



# Introduzione a Git e GitHub

**Progetto ICT Forma**

Istituto Statale di Istruzione Superiore "Gobetti - Volta"



Istituto Tecnico  
Informatica e Telecom  
**GobettiVolta**



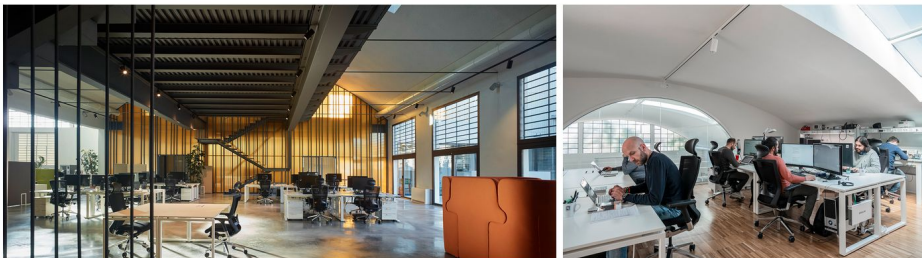
# Ciao!

## Massimiliano Atzori

- lavoro in Develer
- programmo dal 1999
- mi occupo di coordinare progetti software

## Come contattarmi

- social [@amaxis](#) (Twitter)
- email [massimiliano.atzori@gmail.com](mailto:massimiliano.atzori@gmail.com)



50+

Dipendenti

18

Anni di esperienza

300+

Progetti realizzati

100+

Clienti

The logo for Develer, featuring the word "develer" in a bold, blue, sans-serif font. Above the "e" and "l" is a horizontal bar with a teal segment on the left and an orange segment on the right.

# Cosa faremo

1. Installiamo Git e vediamo com'è organizzato GitHub
2. Breve introduzione a Git
3. Primi passi con Git: clone, add, commit
4. Portiamo le modifiche in remoto: push e pull
5. Conflitti e merge
6. Riepilogo e consigli utili



Photo by Brendan Steeves on Unsplash

# Cosa portare a casa

1. Capire che i programmatori non lavorano **mai** da soli
2. Scoprire che esistono Git e GitHub
3. Imparare i comandi base più semplici
4. Iniziare a studiare i concetti più avanzati
5. Collaborare ad un progetto open source



# 1. Installare Git e account su GitHub

---

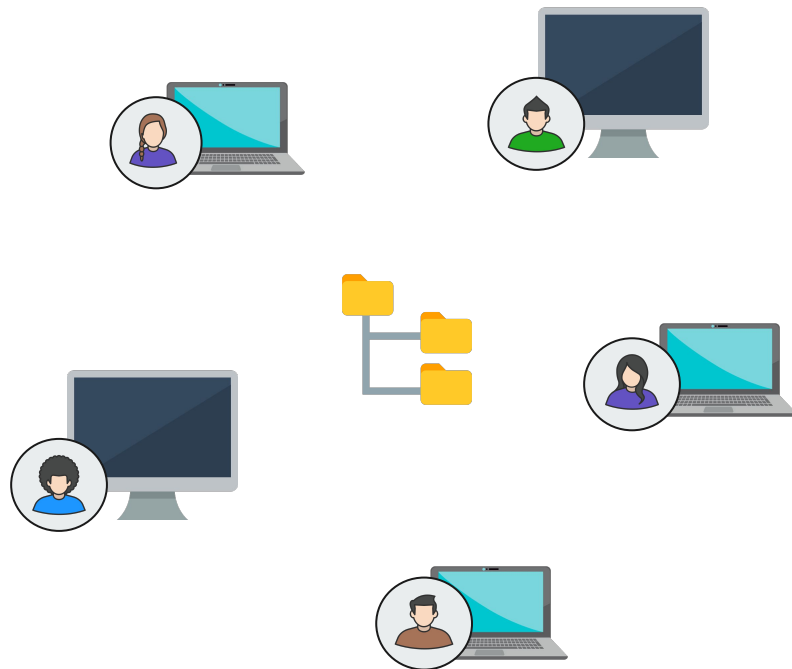
# Cos'è Git?

È un **software di controllo di versione (VCS)**

Serve a:

- Memorizzare il codice sorgente dei programmi
- Riportare i file ad uno stato precedente
- Riportare un progetto ad uno stato precedente
- Confrontare modifiche fatte nel tempo
- Capire chi ha modificato cosa

In poche parole, è un insieme di strumenti per evitare di pestarsi i piedi (il più possibile)





Linus Torvalds, Aprile 2005 (4 giorni!)

Junio Hamano, Luglio 2005 - oggi

Il nome significa “stupido” in slang e Linus lo prende da una canzone dei Beatles (*I'm so tired*)



Linus Torvalds



Junio Hamano



# Installare Git sulla propria macchina

## Installare Git

Windows <https://git-scm.com/download/win>

MacOS <https://git-scm.com/download/mac>

Linux 

```
sudo apt-get install git
```

Debian

```
sudo yum install git
```

Fedora





Permette di creare repository Git online

GitHub nasce nel 2008

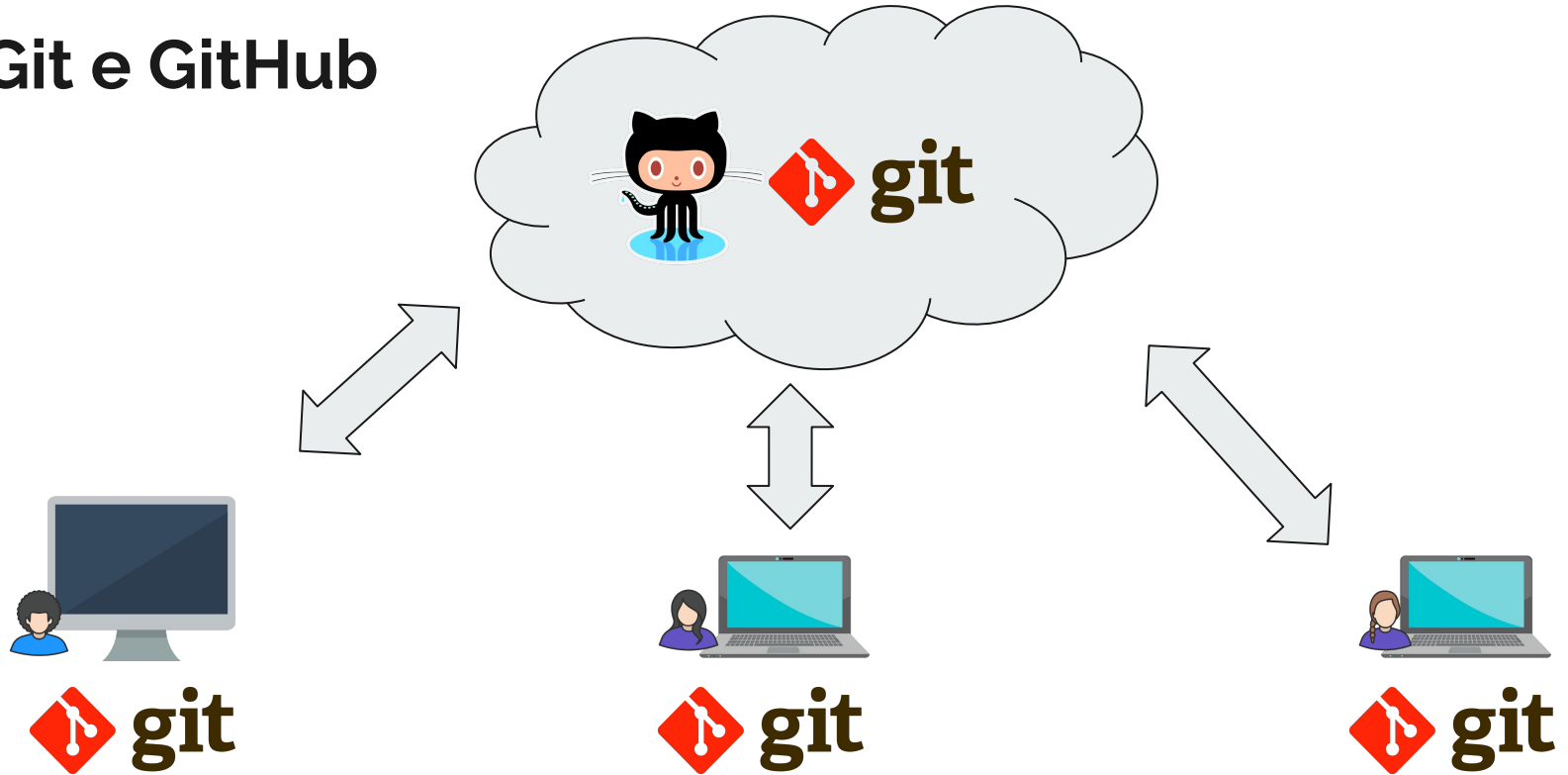
Viene acquisito da Microsoft nel 2018

Gratuito per progetti Open Source

<https://github.com/>

A screenshot of the GitHub website's sign-up page. The page has a dark background with a light-colored sign-up form on the right. The form contains fields for Username, Email, and Password, each with a placeholder text. Below the Password field is a note: "Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)". At the bottom of the form is a green button labeled "Sign up for GitHub". Above the button is a line of text: "By clicking 'Sign up for GitHub', you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails." The top of the page features a navigation bar with links: "Why GitHub?", "Enterprise", "Explore", "Marketplace", and "Pricing". There is also a search bar and "Sign in" and "Sign up" buttons.

# Git e GitHub





# Come si usano Git e GitHub

1. Scriviamo del software con quello che ci pare (Eclipse, VSCode, Sublime Text, il blocco note e simili)
2. Quando siamo arrivati ad un punto “stabile” o significativo, chiediamo a Git di **registrare le modifiche**
3. Git registra le modifiche e le rende **permanenti**
4. Quando siamo pronti, chiediamo a Git di inviare le modifiche permanenti su GitHub
5. Altre persone prenderanno le nostre modifiche da Github e le useranno

## 2. Breve introduzione a Git

---

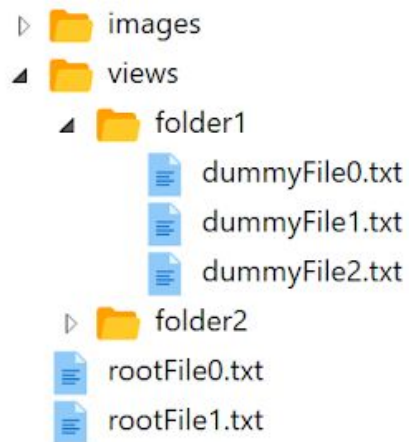
# Repository

Un **repository** è un **contenitore di file e cartelle**. In genere, questi file e cartelle, nel loro insieme, costituiscono un **progetto software**.

Es. creiamo un progetto con Eclipse: tutto il codice che scriviamo lo possiamo mettere in un repository

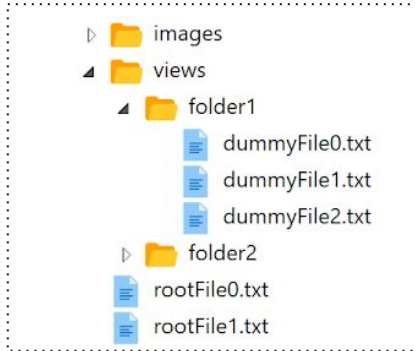
Un software può stare in più repository

Progetto  
software

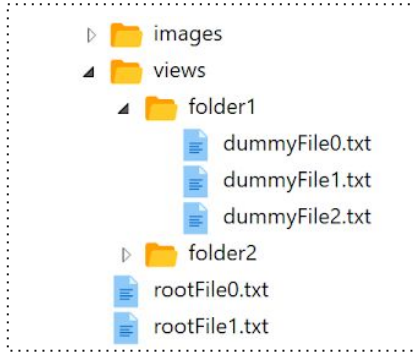


git-gobetti-volta

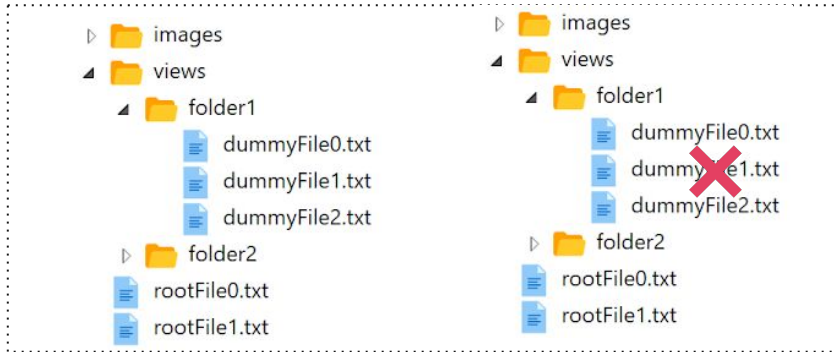
# Repository



# Repository

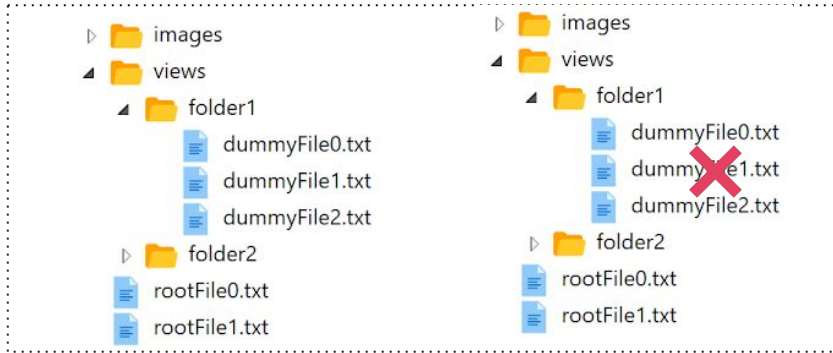


# Repository

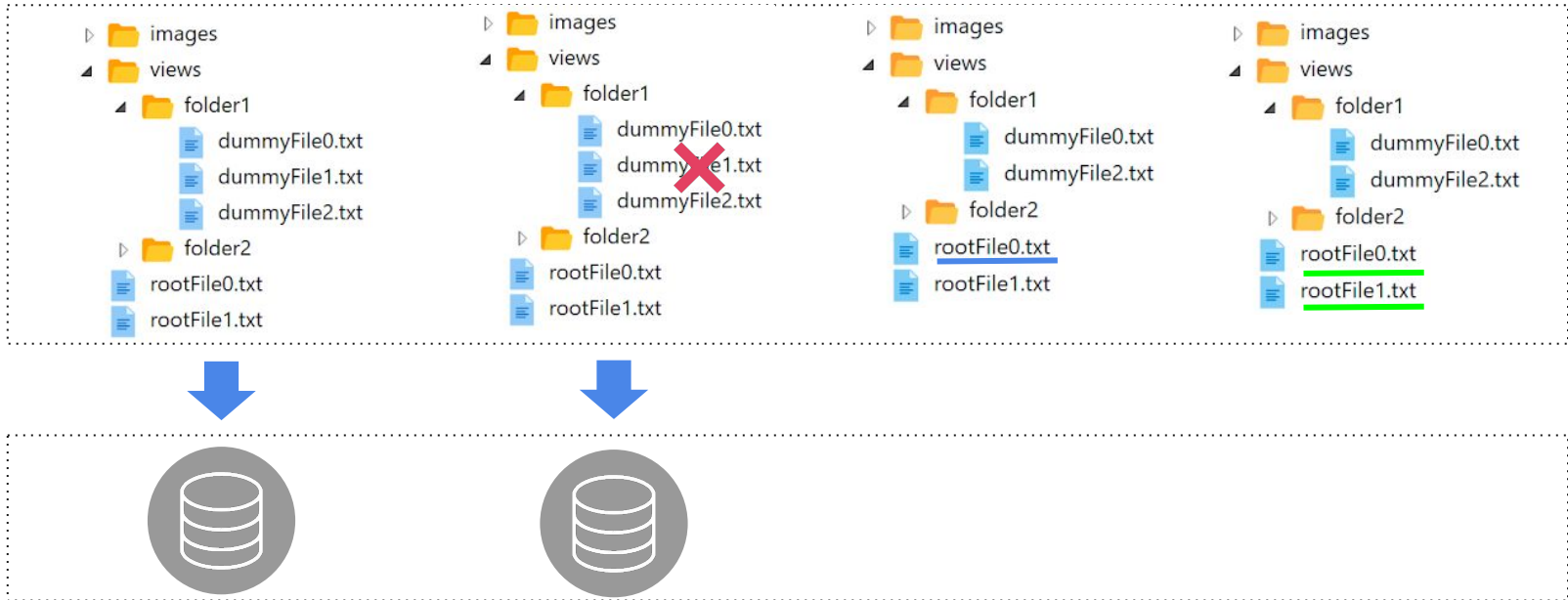




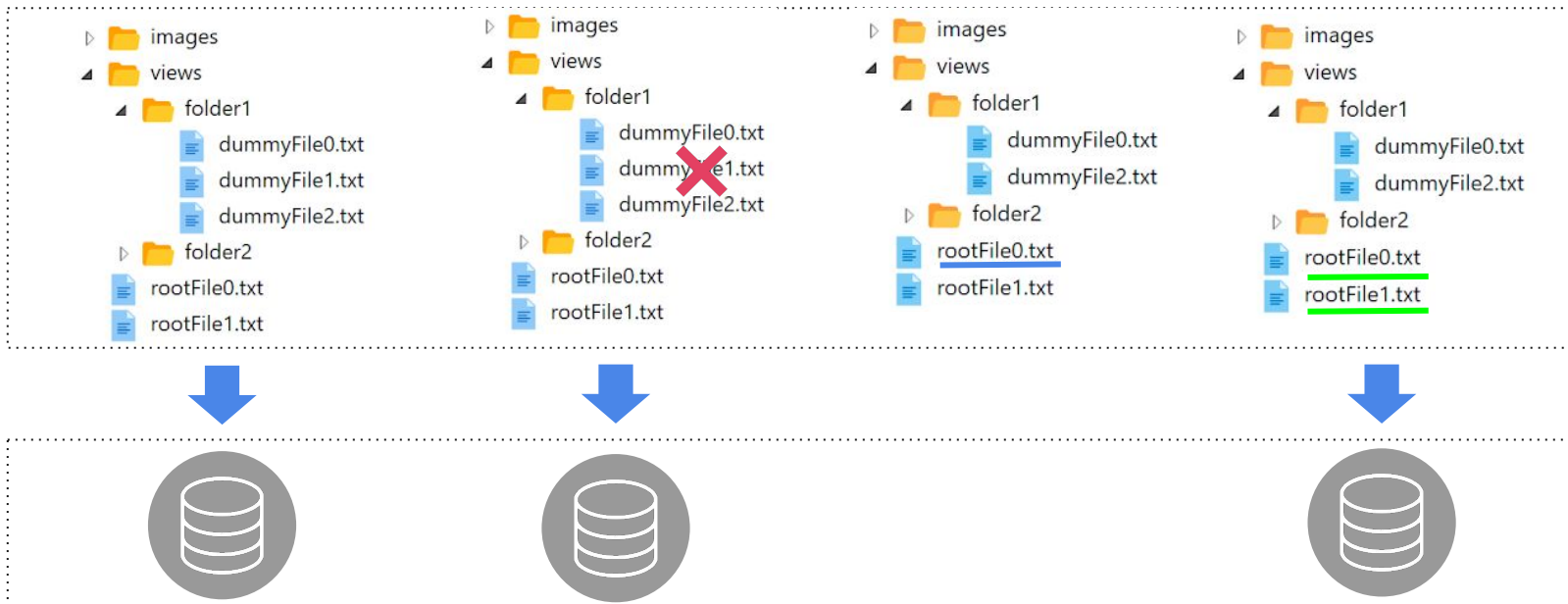
# Repository



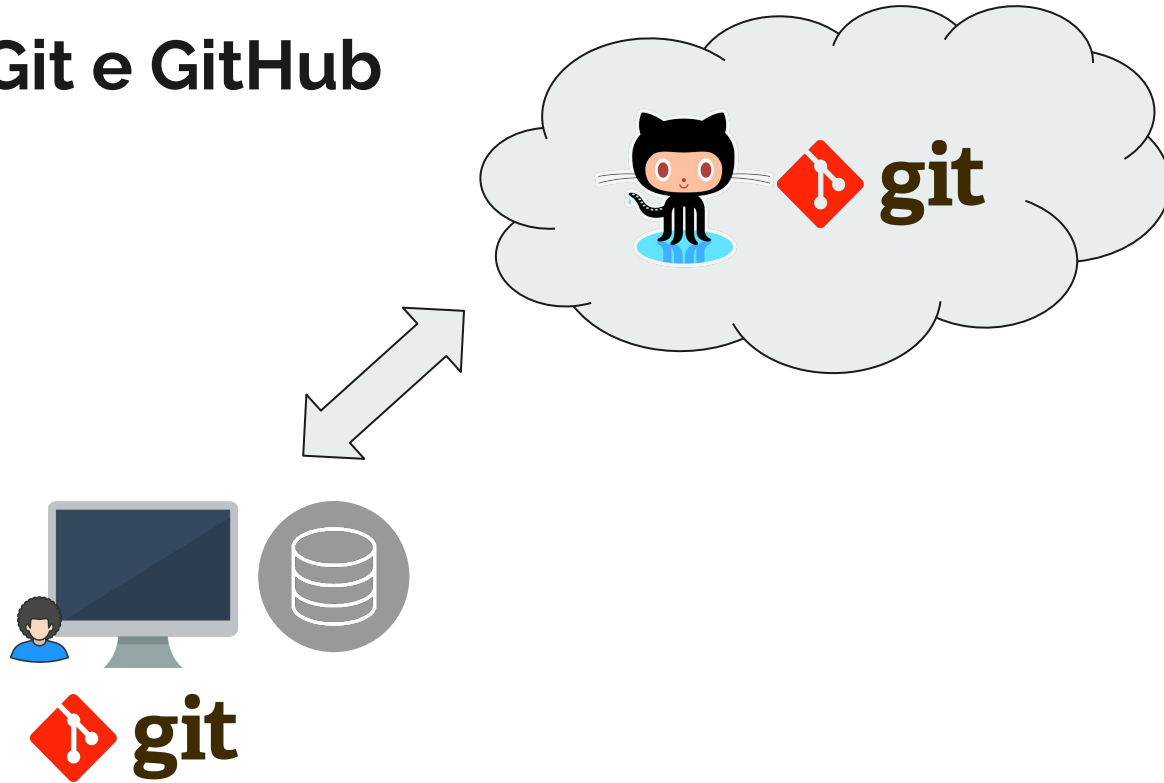
# Repository



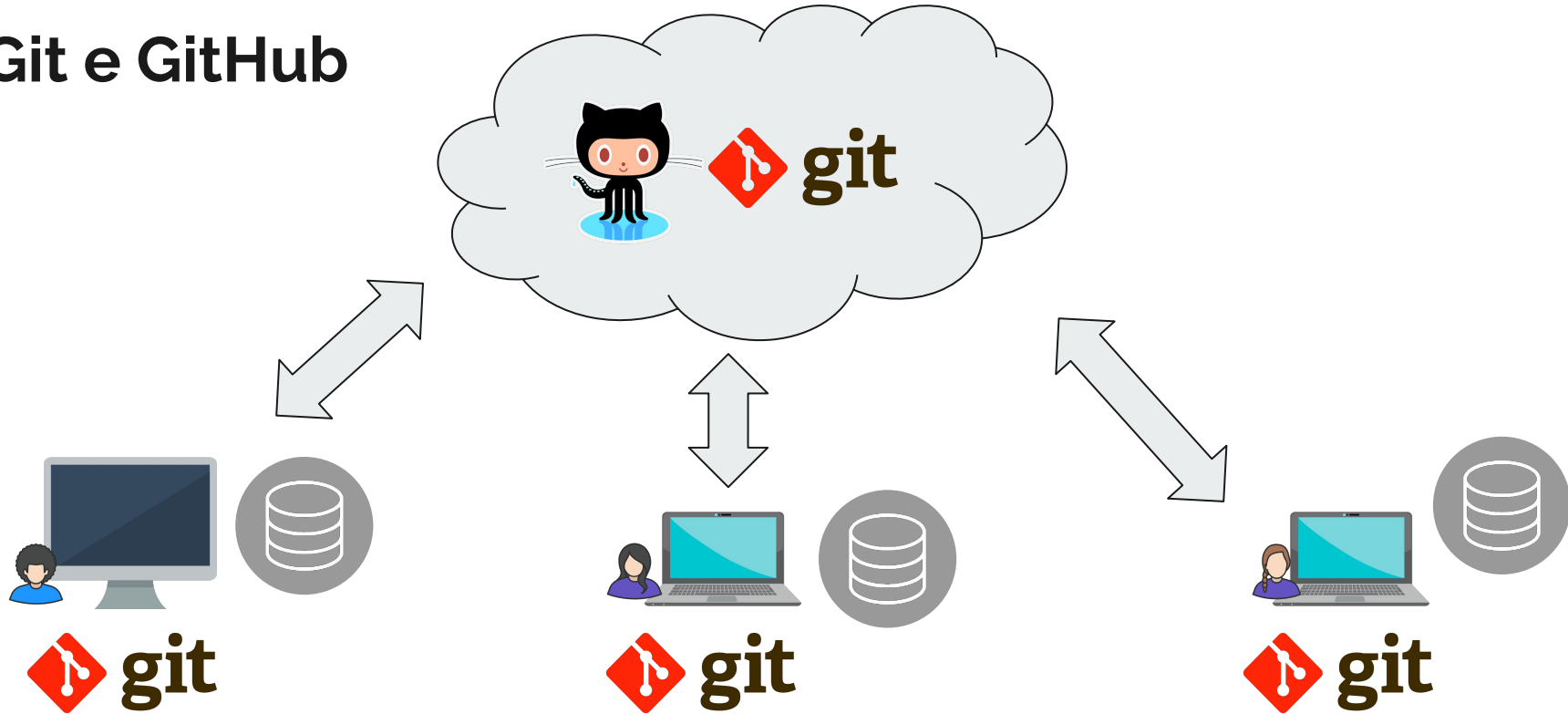
# Repository



# Git e GitHub



# Git e GitHub



# Primi passi con Git: clone, add, commit

---



# Primi passi con Git

Si può “parlare” con Git in **tre modi**:

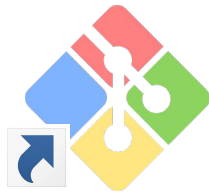
- da linea di comando
- tramite programmi con interfaccia grafica (TortoiseGit, SourceTree, **GitHub desktop** e simili)
- direttamente dall’IDE (Eclipse, VSCode, SublimeText, QtCreator e simili)

**Lo useremo solo da linea di comando (o terminale)**

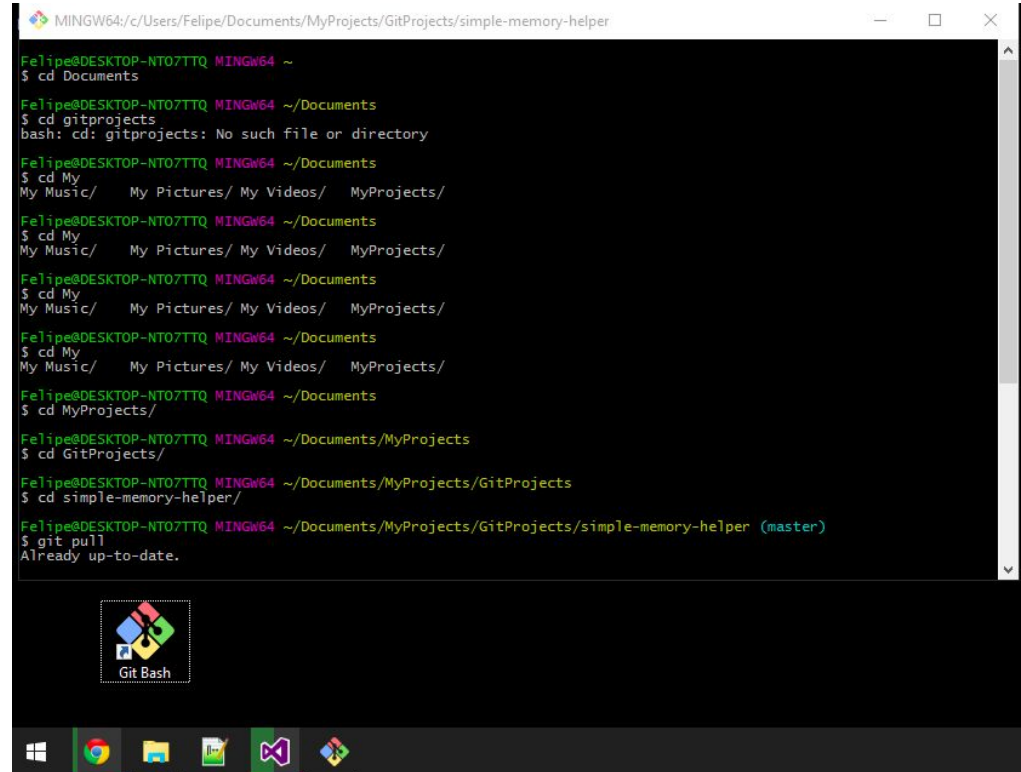
- Terminale (Linux, MacOS)
- Git Bash (Windows)



# Windows: Git Bash



Git Bash

A screenshot of a Windows desktop environment. A Git Bash terminal window is open, displaying a series of commands and their outputs. The window title bar shows the path "MINGW64:/c:/Users/Felipe/Documents/MyProjects/GitProjects/simple-memory-helper". The terminal output shows the user navigating through directories: from the home directory to Documents, then to a subdirectory named "My", and finally to "MyProjects/GitProjects/simple-memory-helper". The user then runs "git pull", which reports "Already up-to-date." The Windows taskbar is visible at the bottom, showing icons for the Start menu, Google Chrome, File Explorer, a folder icon, and the Git Bash application. A small Git Bash icon is also visible on the desktop background below the terminal window.

```
Felipe@DESKTOP-NT07TTQ MINGW64 ~  
$ cd Documents  
  
Felipe@DESKTOP-NT07TTQ MINGW64 ~/Documents  
$ cd gitprojects  
bash: cd: gitprojects: No such file or directory  
  
Felipe@DESKTOP-NT07TTQ MINGW64 ~/Documents  
$ cd My  
My Music/ My Pictures/ My Videos/ MyProjects/  
  
Felipe@DESKTOP-NT07TTQ MINGW64 ~/Documents  
$ cd My  
My Music/ My Pictures/ My Videos/ MyProjects/  
  
Felipe@DESKTOP-NT07TTQ MINGW64 ~/Documents  
$ cd My  
My Music/ My Pictures/ My Videos/ MyProjects/  
  
Felipe@DESKTOP-NT07TTQ MINGW64 ~/Documents  
$ cd MyProjects/  
  
Felipe@DESKTOP-NT07TTQ MINGW64 ~/Documents/MyProjects  
$ cd GitProjects/  
  
Felipe@DESKTOP-NT07TTQ MINGW64 ~/Documents/MyProjects/GitProjects  
$ cd simple-memory-helper/  
  
Felipe@DESKTOP-NT07TTQ MINGW64 ~/Documents/MyProjects/GitProjects/simple-memory-helper (master)  
$ git pull  
Already up-to-date.
```





# Terminale (o linea di comando)

Un programma linea di comando, come Git Bash, è molto utile per eseguire programmi e script. Vediamo alcuni comandi utili:

```
pwd
```

Stampa la directory corrente

```
ls -la
```

Stampa il contenuto della directory corrente

```
cd <nuova directory>
```

Cambia la directory corrente

```
clear
```

Cancella l'output del video

Non vi precipitate a scrivere comandi! Li vediamo prima insieme

---

**Proviamo!**



# Git e terminale

Per dare comandi a Git

```
git <comando> <opzioni>
```

Si possono dare tanti comandi a Git: per esempio

- **Operazioni sul repository:** aggiungi e togli file dal repository, registra le modifiche fatte ai file,
- **Operazioni sullo stato:** dimmi qual è la situazione
- **Operazioni sulle registrazioni:** chi ha fatto una registrazione



# Primi comandi

Vediamo i primi comandi che si danno a Git per una prima configurazione:

Verificare la presenza di Git

```
git --version
```

Configurare nome e email

```
git config --global user.name "Massimiliano Atzori"  
git config --global user.email massimiliano@develer.com
```

Output con colori

```
git config --global color.ui.auto
```

Verificare la configurazione

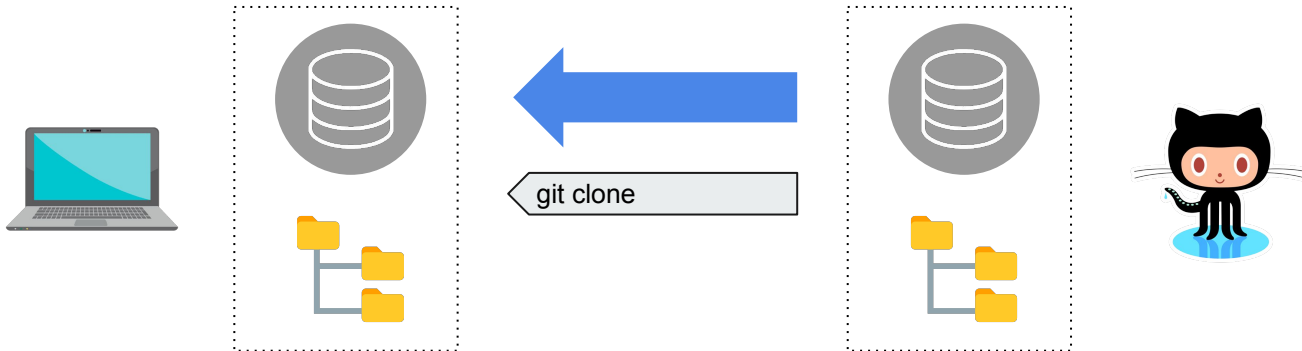
```
git config --list
```

---

**Proviamo!**

# Clonare un repository da GitHub

```
git clone https://github.com/amaxis/git-gobetti-volta.git
```





# Cos'è successo?

Vediamo cosa abbiamo scaricato: **cambiamo la directory**

```
cd git-gobetti-volta
```

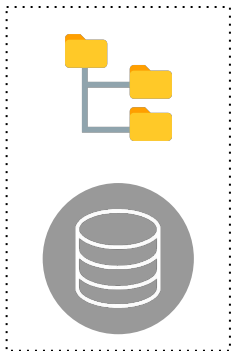
Esaminiamo il contenuto

```
ls
```

Possiamo ottenere lo stesso risultato aprendo la cartella dei file scaricati con Esplora Risorse

# Cos'è successo?

Con il comando clone, abbiamo scaricato da GitHub due oggetti:



L'ultima versione dei **file** e **directory** del progetto

Il **repository completo**, dove ci sono le informazioni su tutte le modifiche registrate

Da qui in poi, possiamo lavorare normalmente con i file e poi “comunicare” a Git quando è il momento di registrare delle modifiche in modo permanente



# Le aree di lavoro di Git



Workspace



Repository



Remote  
repository



# Le aree di lavoro di Git



Workspace



Index (staging area)



Local repository



Remote repository



# Le aree di lavoro di Git



Workspace

- Sono i file su cui lavoriamo



Index (staging area)

- Modifiche **pronte** per essere registrate



Local repository

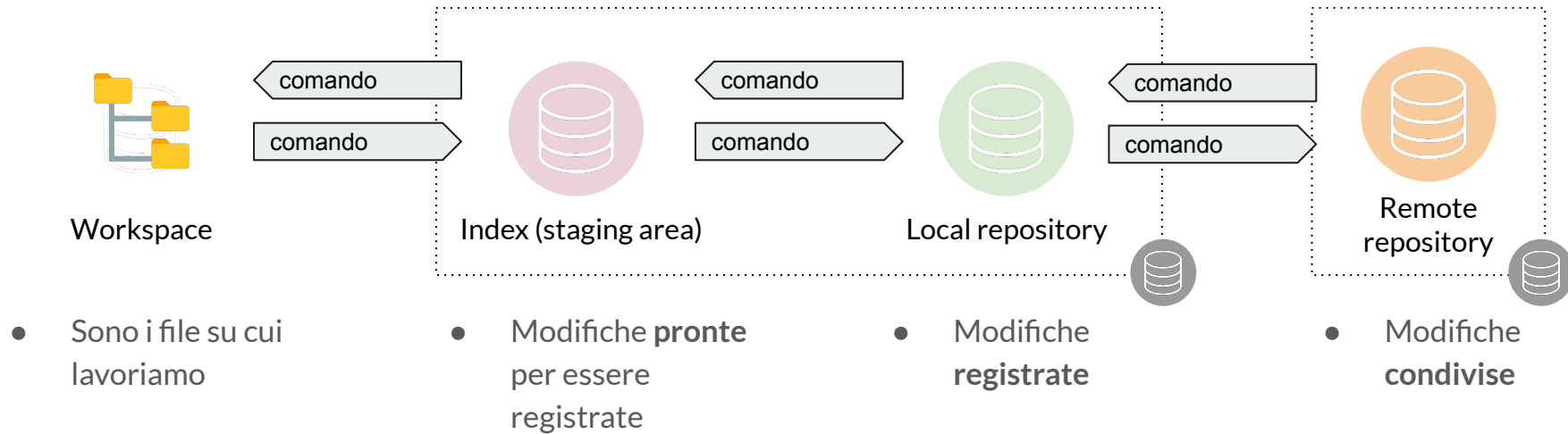
- Modifiche **registrate**



Remote repository

- Modifiche **condivise**

# Le aree di lavoro di Git





# Controllare lo stato del nostro repo

```
git status
```

Cosa ci dice

- se abbiamo fatto modifiche ai file rispetto al contenuto del local repository
- in quali stage si trovano i file che abbiamo modificato
- se ci sono conflitti da risolvere
- se ci sono altre situazioni particolari



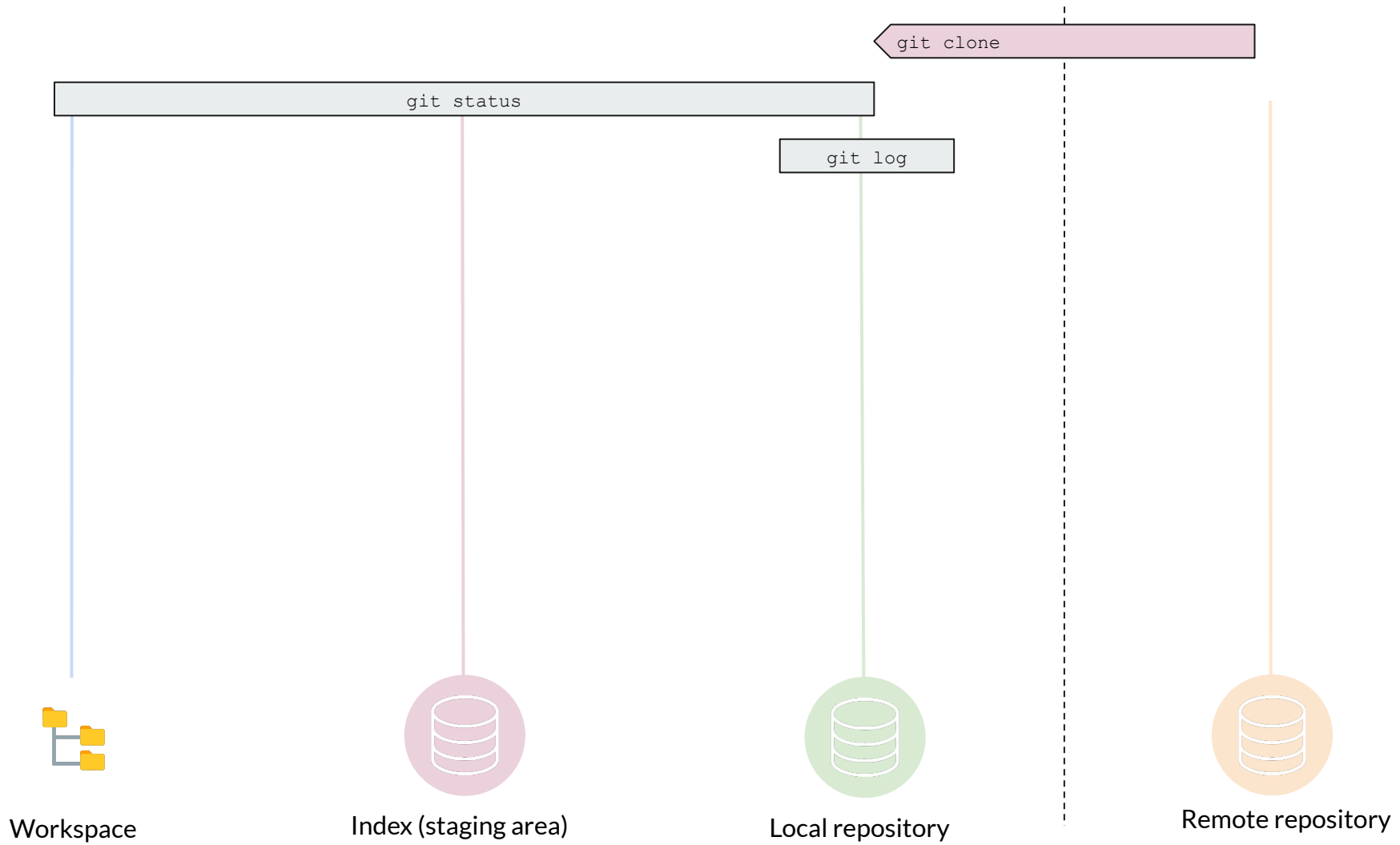
# Controllare lo stato del nostro repo

```
git log
```

```
git log --oneline
```

Cosa ci dicono

- ci fa vedere la lista delle modifiche permanenti fatte al repository locale
- `--oneline`: mi fa vedere una lista più compatta



---

**Proviamo!**



**Modificare i file e portarli in  
remoto**

—

# Le aree di lavoro di Git



Workspace

- Sono i file su cui lavoriamo



Index (staging area)

- Modifiche **pronte** per essere registrate



Local repository

- Modifiche **registrate**



Remote repository

- Modifiche **condivise**



# Aggiungiamo un file nel repository Git locale

Apriamo il file

```
git-gobetti-volta/java/CalcoloArea.java
```

Cambiamo il messaggio della riga 14

Salviamo le modifiche

Vediamo lo status

```
git status
```



# Aggiungiamo un file nel repository Git locale

Apriamo il file

```
git-gobetti-volta/java/CalcoloArea.java
```

Cambiamo il messaggio della riga 14

Salviamo le modifiche

Vediamo lo status

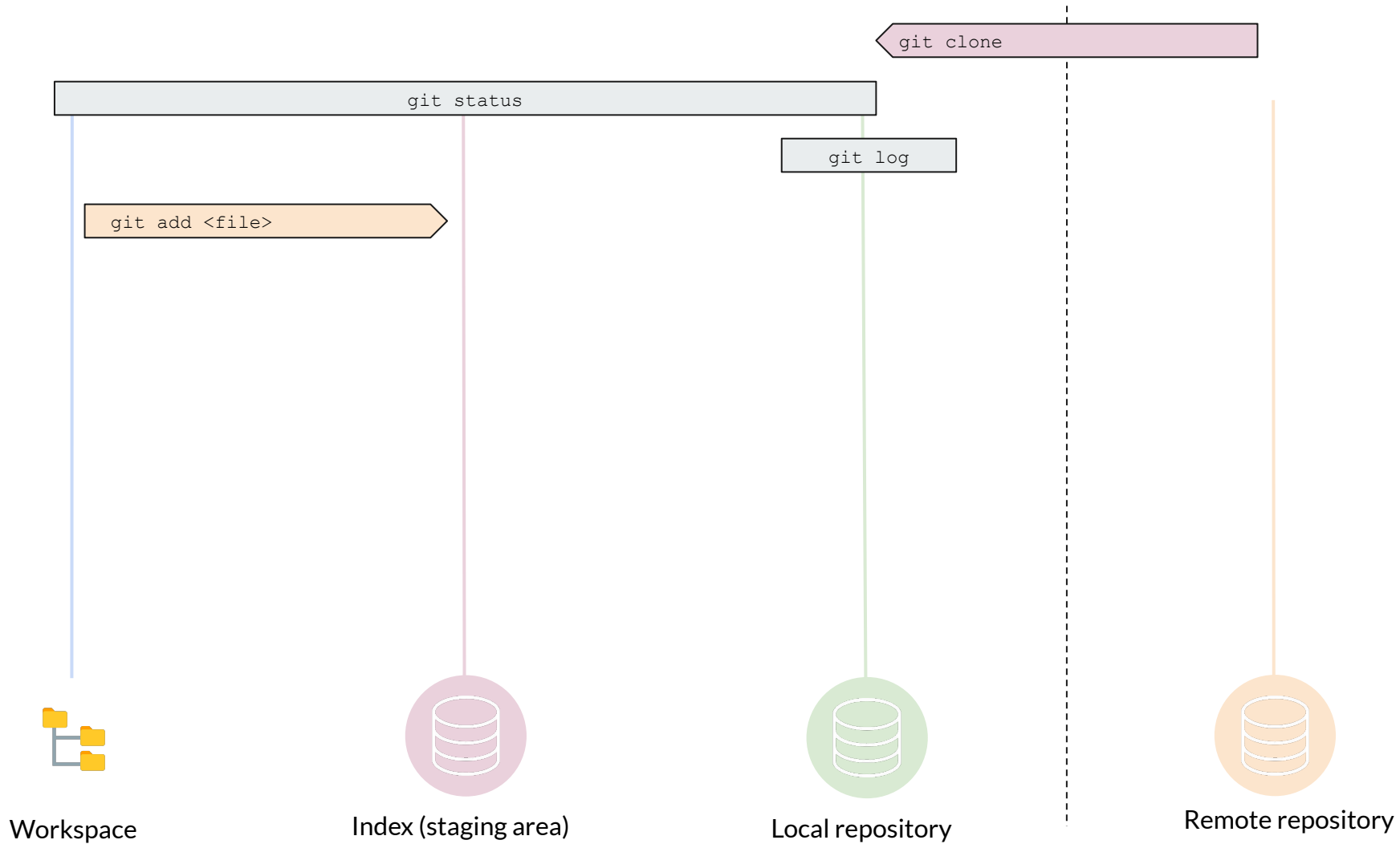
```
git status
```

Chiediamo a Git di aggiungerlo nello index

```
git add java/CalcoloArea.java
```

Vediamo lo status

```
git status
```





## Proviamo a togliere il file dallo staging

Chiediamo a Git di togliere il file dall'Index

```
git reset java/CalcoloArea.java
```

Vediamo lo status

```
git status
```



## Proviamo a togliere il file dallo staging

Chiediamo a Git di togliere il file dall'Index

```
git reset java/CalcoloArea.java
```

Vediamo lo status

```
git status
```

Aggiungiamolo di nuovo

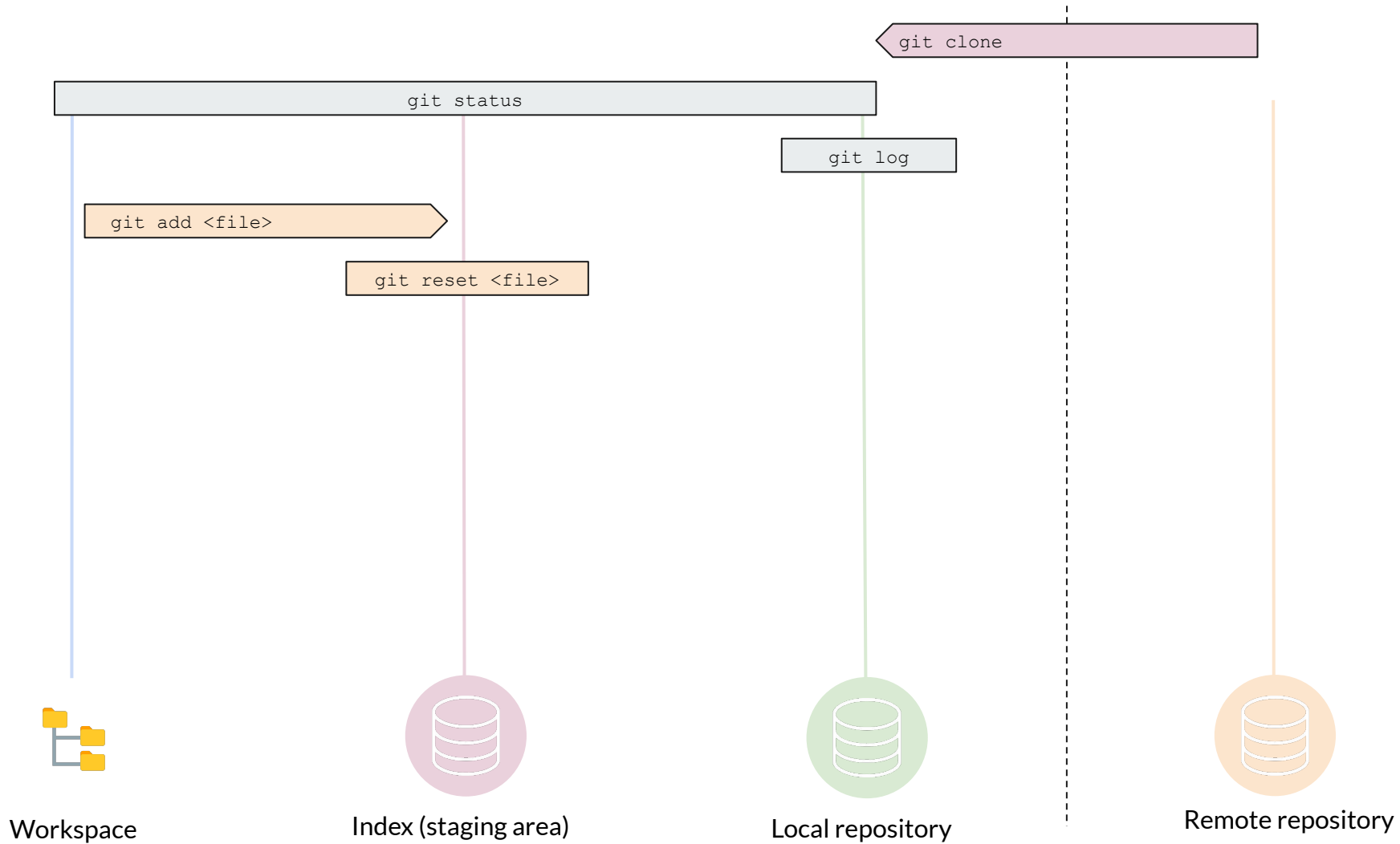
```
git add java/CalcoloArea.java
```

Vediamo lo status

```
git status
```

Vediamo la situazione del repository locale

```
git log --oneline
```







# Aggiorniamo il repository locale

Vediamo lo status

```
git status
```

Aggiorniamo il repository

```
git commit -m "Cambio testo input per utente"
```



# Aggiorniamo il repository locale

Vediamo lo status

```
git status
```

Aggiorniamo il repository

```
git commit -m "Cambio testo input per utente"
```

Vediamo lo status

```
git status
```

Vediamo la situazione del repository

```
git log --oneline
```



## Cos'è successo?

1. Abbiamo modificato un file
2. Lo abbiamo aggiunto nella staging area con il comando `add`
3. Abbiamo creato un **commit** con il comando `git commit`, aggiornando il repository locale

Le modifiche al repository, per il momento, si trovano ancora sul nostro computer.

Possiamo controllare la lista dei commit con il comando `git log`



# I commit

I commit ci raccontano la storia delle modifiche al nostro codice

Vanno sempre avanti, non si cancellano mai: sono **snapshot permanenti**

Il messaggio di commit è obbligatorio, deve descrivere cos'è successo nelle modifiche: si scrive al presente in prima persona singolare. Meglio se in inglese.

Tutte le modifiche sono ancora in locale sulla nostra macchina nel Local Repository

Per vedere la lista dei commit

```
git log
```

```
git log --oneline
```



# Differenze tra commit

Lista dei commit

```
git log --oneline
```

```
794b5be (HEAD -> master, origin/master, origin/HEAD) Correggo errori nei file Java
0b00b55 Aggiorno presentazione con modifiche per i file Java
eb1618f Aggiungo le classi Java di esempio
e053449 Aggiungo PDF con la presentazione del modulo
d812873 Aggiorno README.md con una descrizione e le classi
fd50b95 Initial commit
```

Per vedere le differenze tra commit

```
git diff fd50b95 d812873
```

## Aggiorno README.md con una descrizione e le classi

[Browse files](#)

🔒 master



amaxis committed 5 days ago

1 parent [fd50b95](#)

commit [d8128735d9566161651a473f9ea5458442e8b91b](#)

Showing 1 changed file with 7 additions and 1 deletion.

Unified Split

▼ 8 README.md



... @@ -1,2 +1,8 @@

1 1 # git-gobetti-volta

2 - Repository per il corso Git al Gobetti-Volta

2 +

3 + Questo è un repository pubblico per gli esercizi del modulo su Git e Github dell'Istituto Statale di Istruzione Superiore "Gobetti - Volta".

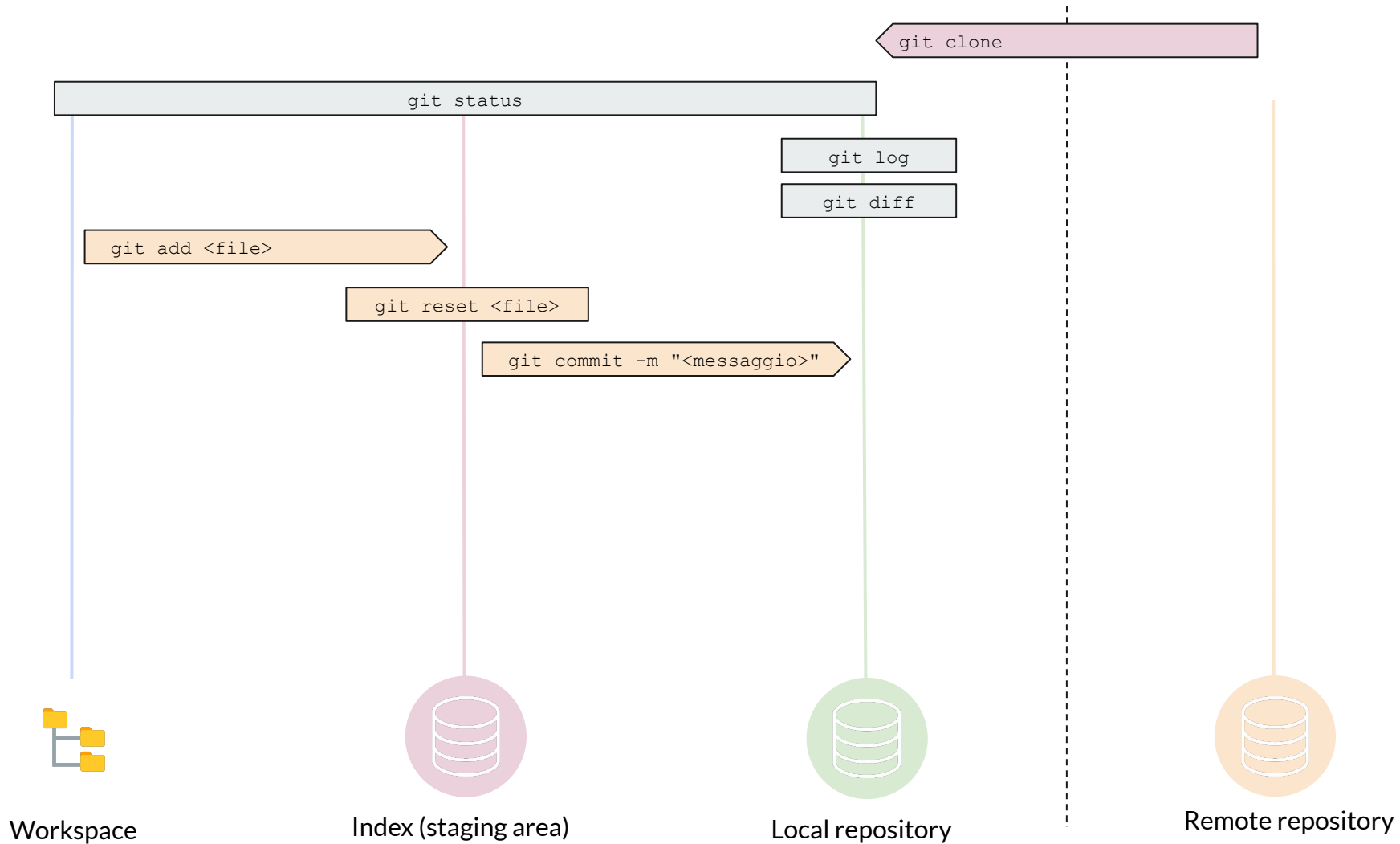
4 +

5 + Classi:

6 + \* 4A IT

7 + \* 4B IT

8 + \* 4C TL





# Portare le modifiche in remoto

Per portare le modifiche (commit) sul repository remoto

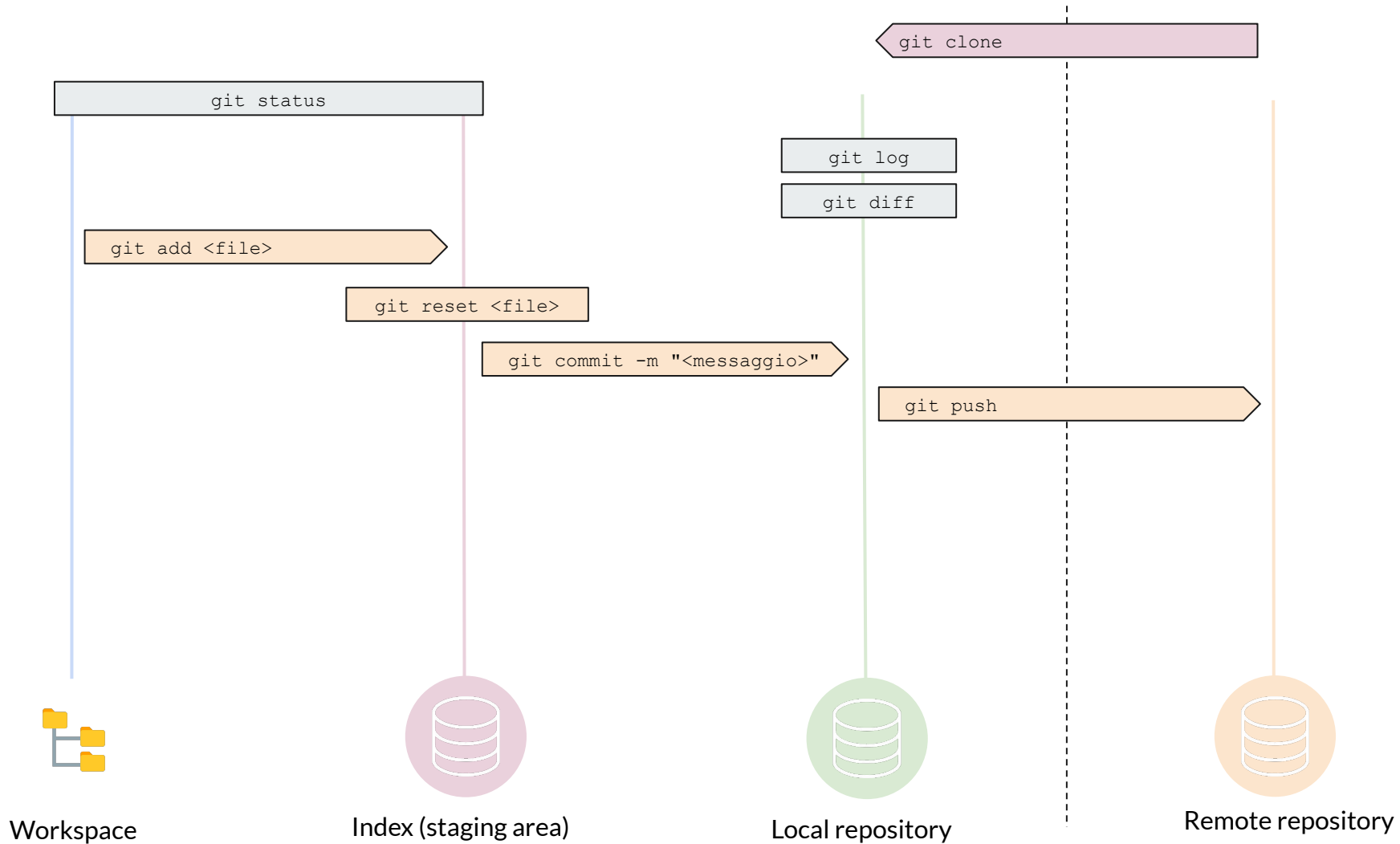
```
git push
```

Porta i commit aggiunti nel local repository sul remote repository

Attenzione

- per l'operazione di push, ci vogliono le autorizzazioni sul repository remoto (nel nostro caso, un account su GitHub)
- portare i commit sul remote può causare **conflitti**







# Un primo semplice flusso di lavoro con Git

Clono un repository

```
git clone
```

Aggiungo, modifico, elimino dei file e li preparo per il commit

```
git add
```

Tengo d'occhio lo stato dei file

```
git status / git log
```

Quando arrivo ad un punto importante, registro le modifiche nel repository locale

```
git commit
```

Quando sono pronto a condividere le modifiche, le porto sul repository remoto

```
git push
```



## Altri comandi utili

Annulla le modifiche registrate nell'index

```
git reset
```

Riporta **un file** dal local repository al workspace

```
git checkout <file>
```

Riporta tutti i file dal local repository al workspace

```
git reset --hard
```

Differenze

- la prima annulla le modifiche fatte nell'index, **il contenuto dei file nel Workspace rimane invariato**
- la seconda riporta i file dal local repository a workspace, perciò **il contenuto di un file viene sovrascritto**
- la terza riporta tutti i file dal local repository al workspace, perciò **il workspace viene riportato esattamente allo stato dell'ultimo commit**



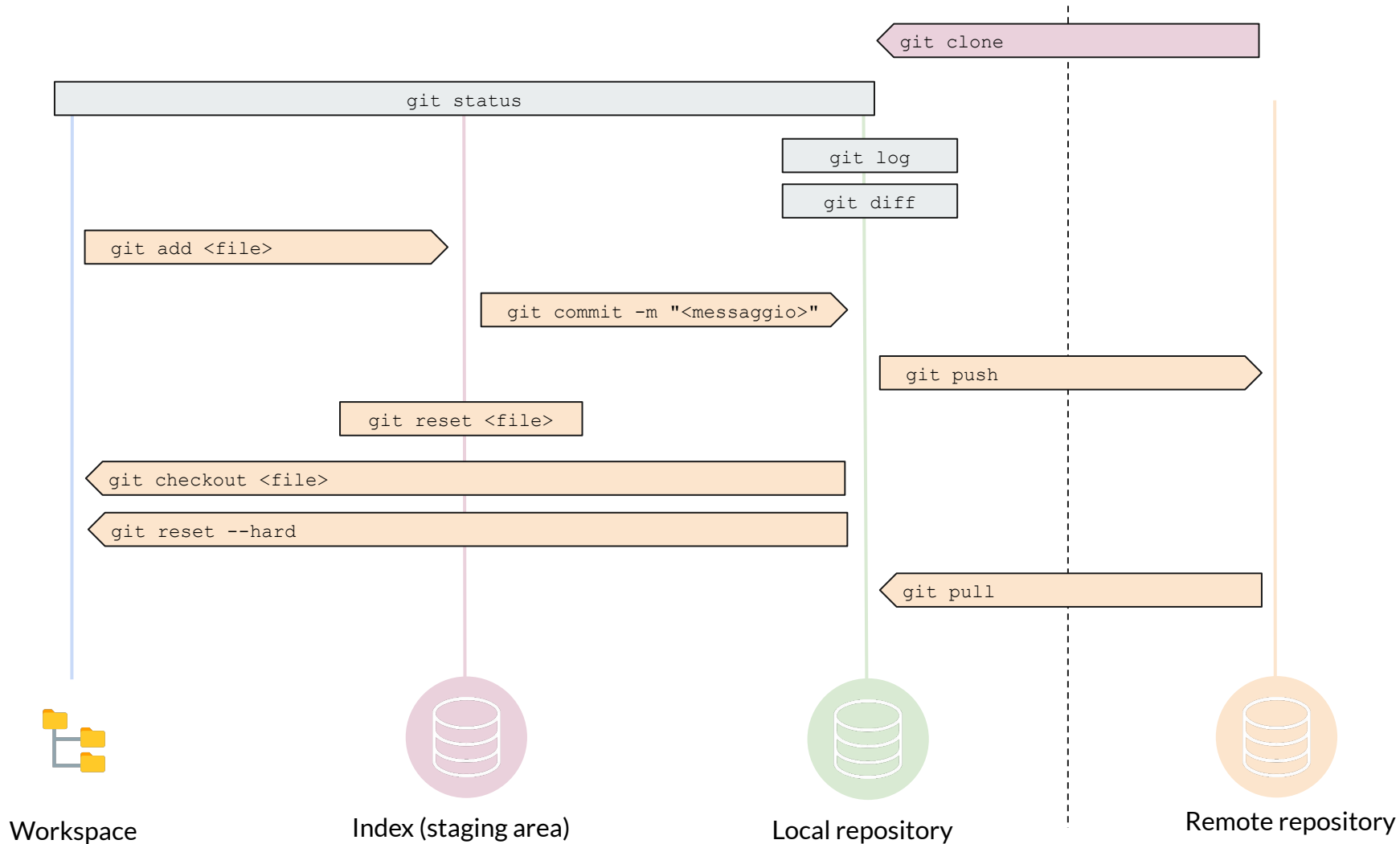
## Altri comandi utili

Prende i **nuovi commit** dal repository remoto

```
git pull
```

Da notare

- il comando `git clone` fa una copia esatta del repository e lo collega a quello remoto
- il comando `git pull` si esegue su un repository già collegato ad uno remoto e copia solo i commit che non si hanno già



---

**Proviamo!**

# Repository e commit: GitHub

---

Repository per il corso Git al Gobetti-Volta

Edit

[Manage topics](#)

📄 6 commits

🌿 1 branch

📦 0 packages

🏷 0 releases

👤 1 contributor

Branch: master ▾

New pull request

Create new file

Upload files

Find file

Clone or download ▾



amaxis Correggo errori nei file Java

Latest commit 794b5be 2 days ago

[Presentazione](#)

Aggiorno presentazione con modifiche per i file Java

3 days ago

[java](#)

Correggo errori nei file Java

2 days ago

[README.md](#)

Aggiorno README.md con una descrizione e le classi

5 days ago

📖 README.md



# git-gobetti-volta



Branch: master ▾

## Commits on Feb 19, 2020

## Correggo errori nei file Java



amaxis committed 19 minutes ago



794b5be



## Commits on Feb 18, 2020

## Aggiorno presentazione con modifiche per i file Java



amaxis committed yesterday



0b00b55



## Aggiungo le classi Java di esempio



amaxis committed yesterday



eb1618f



## Commits on Feb 16, 2020

## Aggiungo PDF con la presentazione del modulo



amaxis committed 3 days ago



e053449





Search or jump to...



[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)



amaxis ▾

## Repositories



New

Find a repository...



[amaxis/git-gobetti-volta](#)



[develersrl/abb-iotop-analyzer](#)



[develersrl/corob-pos](#)



[develersrl/liit](#)



[develersrl/checkmate-ui](#)



[develersrl/ep.appliance-manager](#)



[develersrl/ep.common](#)

Show more

## Your teams



ramiel forked [ramiel/i18next-parser](#) from [i18next/i18next-parser](#) 10 days ago

### i18next/i18next-parser

Code parser for i18next

JavaScript ★ 147 Updated Feb 16



Star



ramiel created a repository [ramiel/parser-test](#) 11 days ago

### ramiel/parser-test

Temporary repository to show an issue

Updated Feb 11



Star



ramiel starred [facebook/react](#) on 16 Jan

### facebook/react

A declarative, efficient, and flexible JavaScript library for build-



Star

## Explore repositories

### tensorflow/build

Build-related tools for TensorFlow

Shell ★ 13

### tensorflow/community

Stores documents used by the TensorFlow developer community

★ 630

### tensorflow/text

Making text a first-class citizen in TensorFlow.

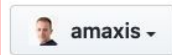
Python ★ 544

Explore more →

## Create a new repository

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

Owner



Repository name \*

progetto-ict



Nome del repository

URL per il clone

```
git clone https://github.com/amaxis/progetto-ict
```

Great repository names are short and memorable. Need inspiration? How about **solid-c**

Description (optional)



**Public**

Anyone can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

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

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

Importante, altrimenti non si può fare immediatamente il clone

Add .gitignore: **None**

Add a license: **None**



Create repository

# Conflitti e merge

---

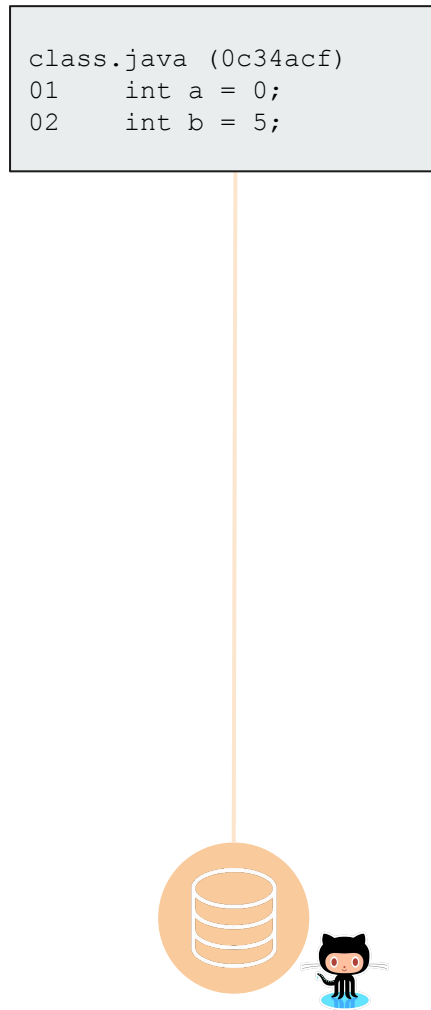
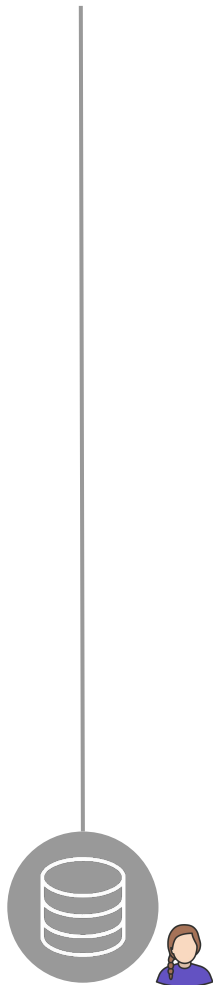
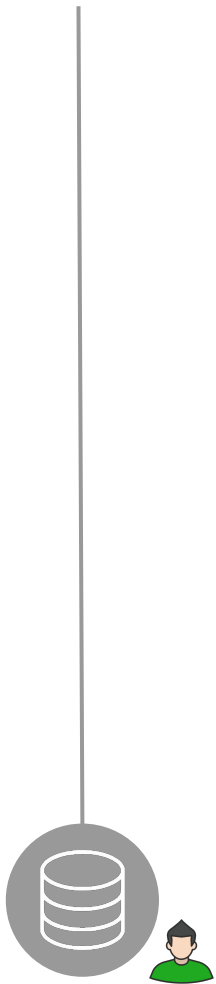


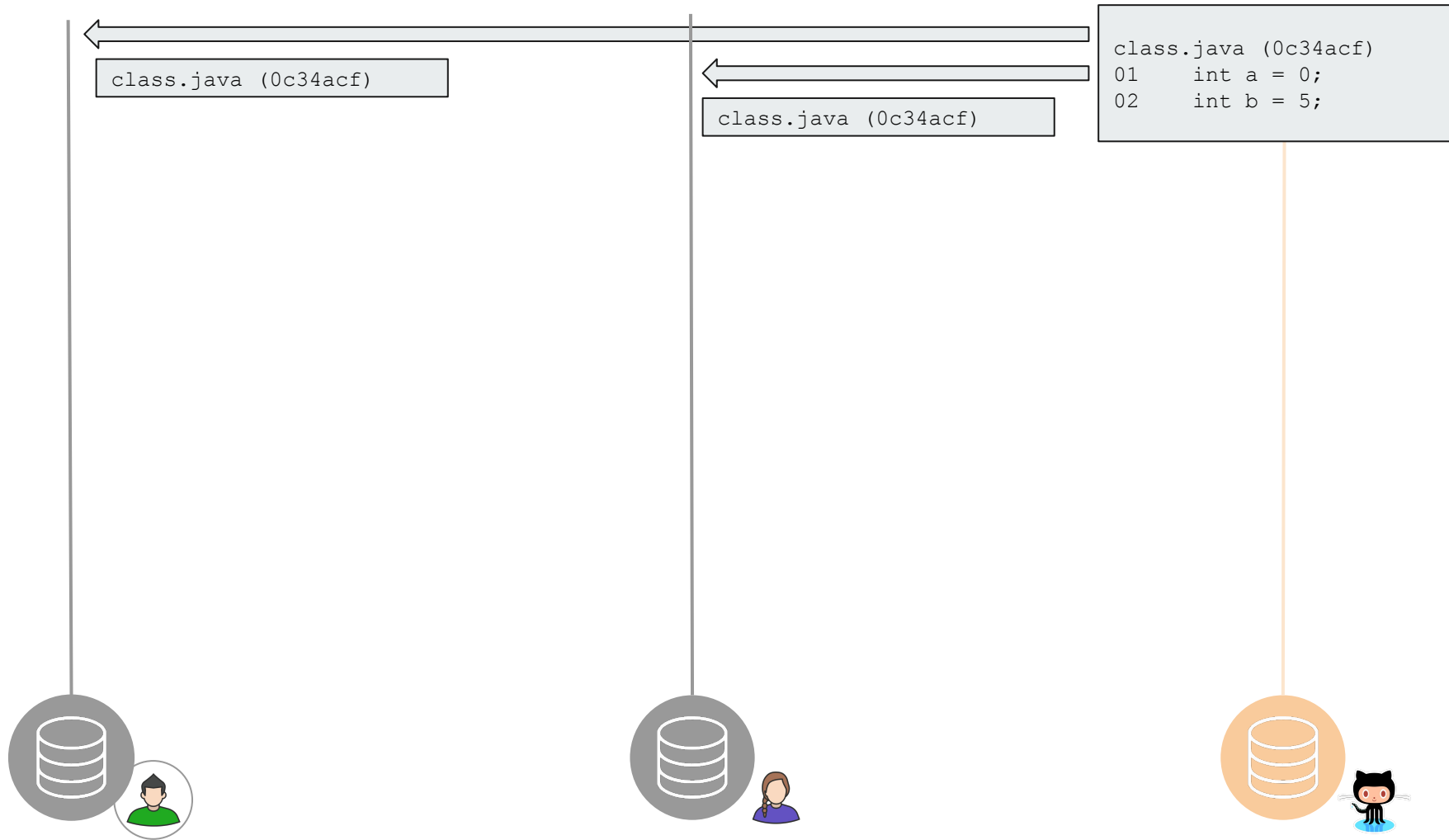
# Git e i conflitti

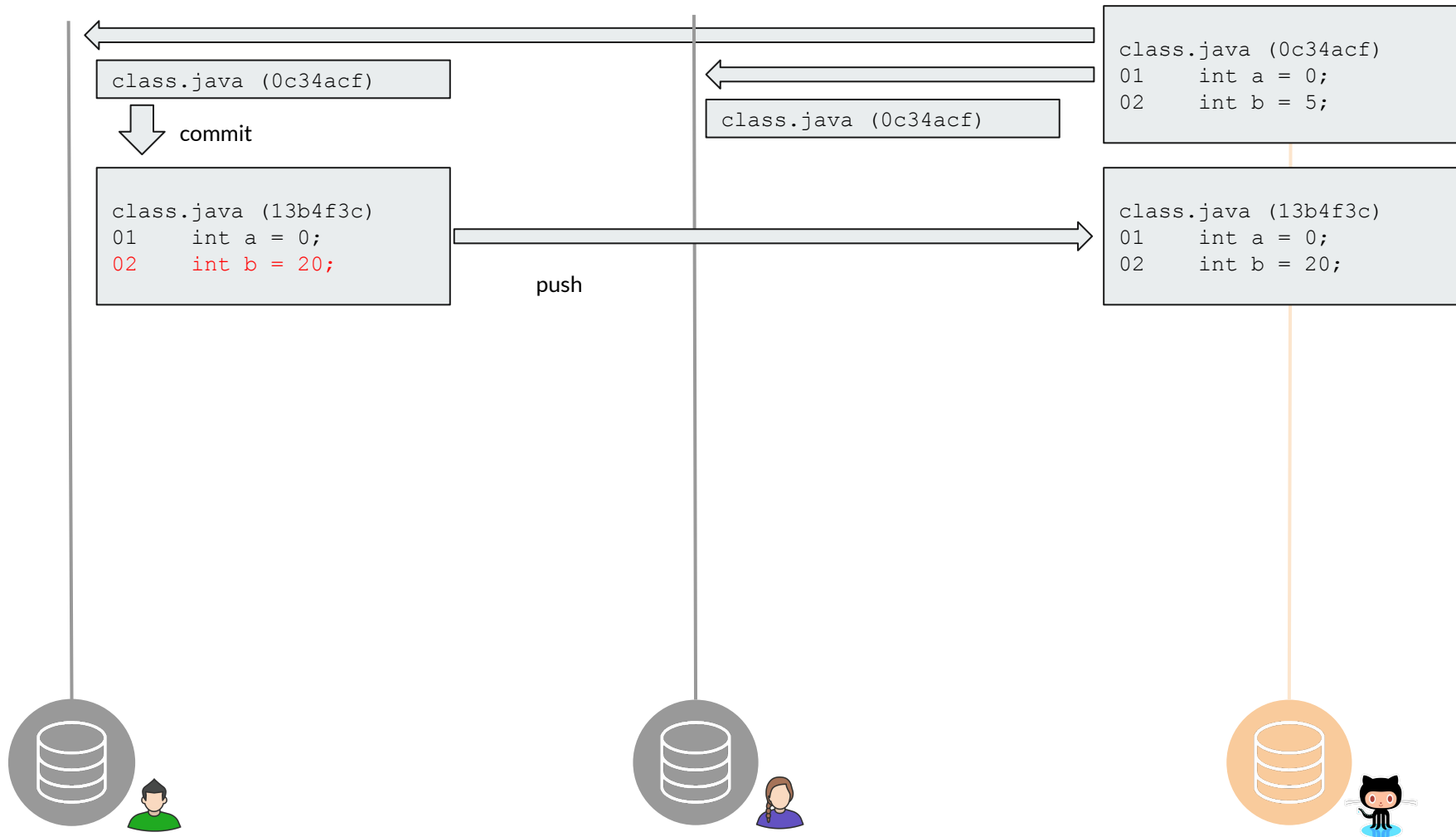
Come abbiamo visto, Git esegue i nostri comandi: quando gli chiediamo di creare un commit, lo fa.

Ogni commit contiene delle modifiche ai file: cosa succede se due utenti diversi modificano le stesse parti di un file?

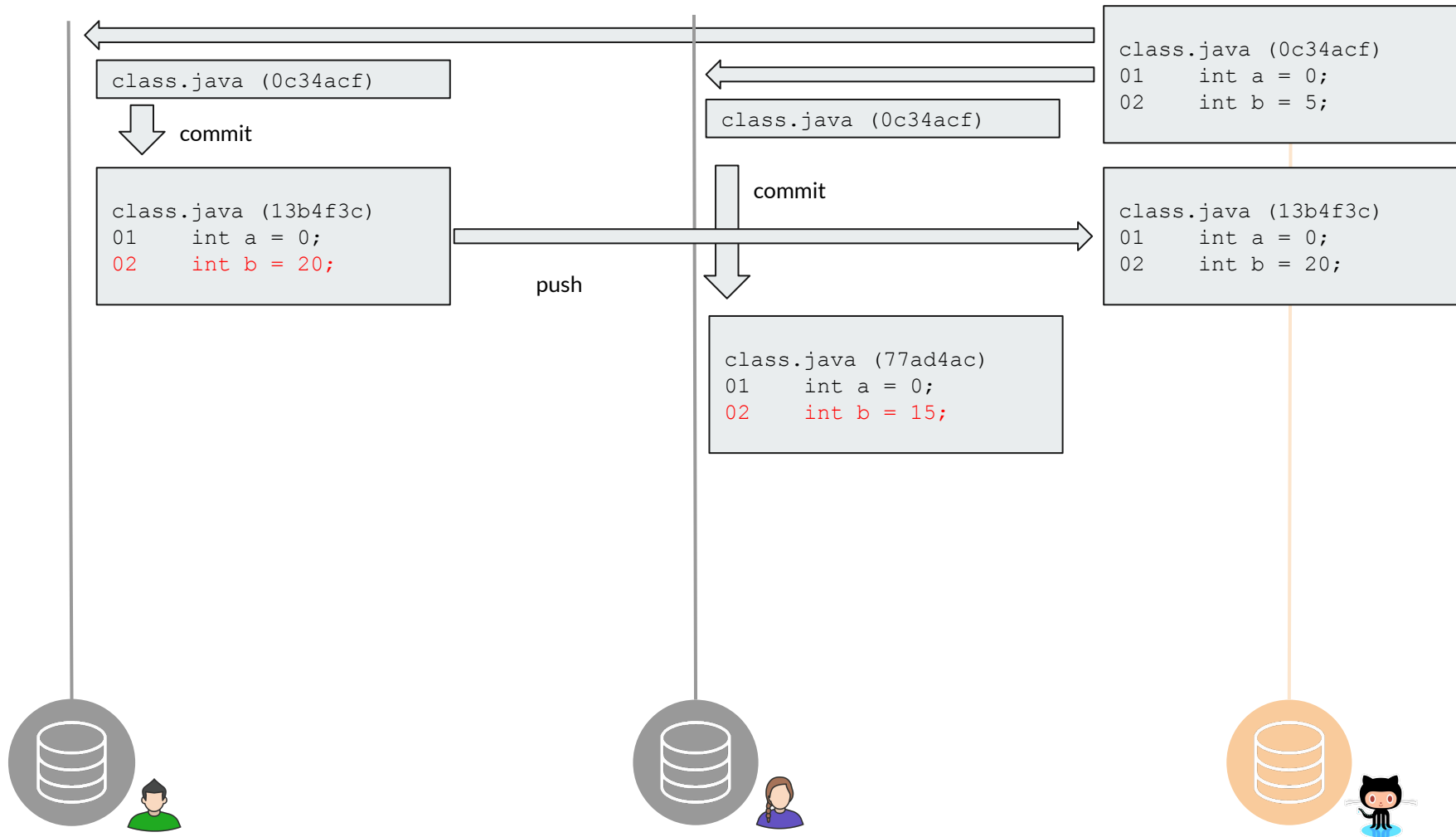
Proviamo a vedere un esempio.

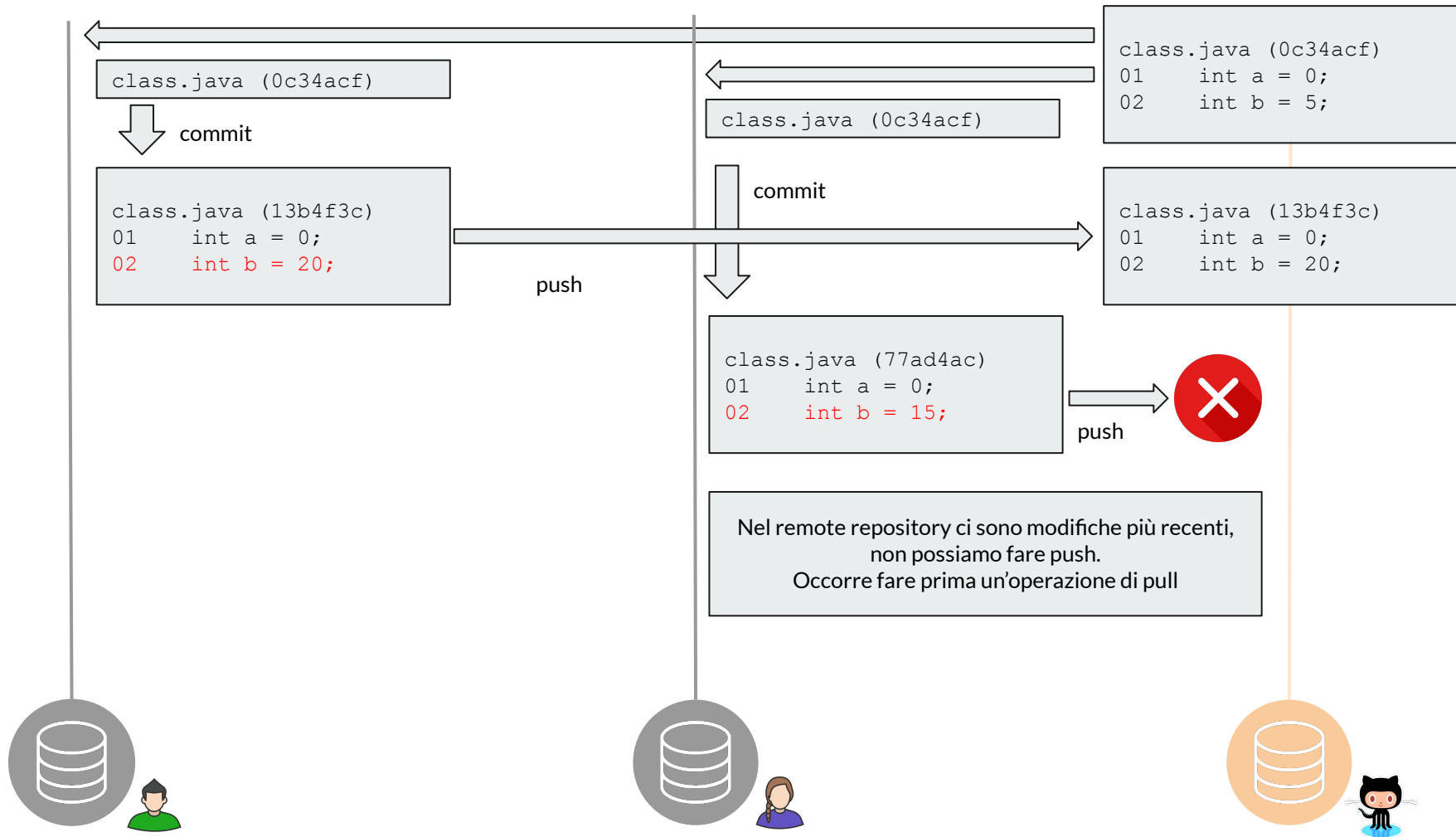


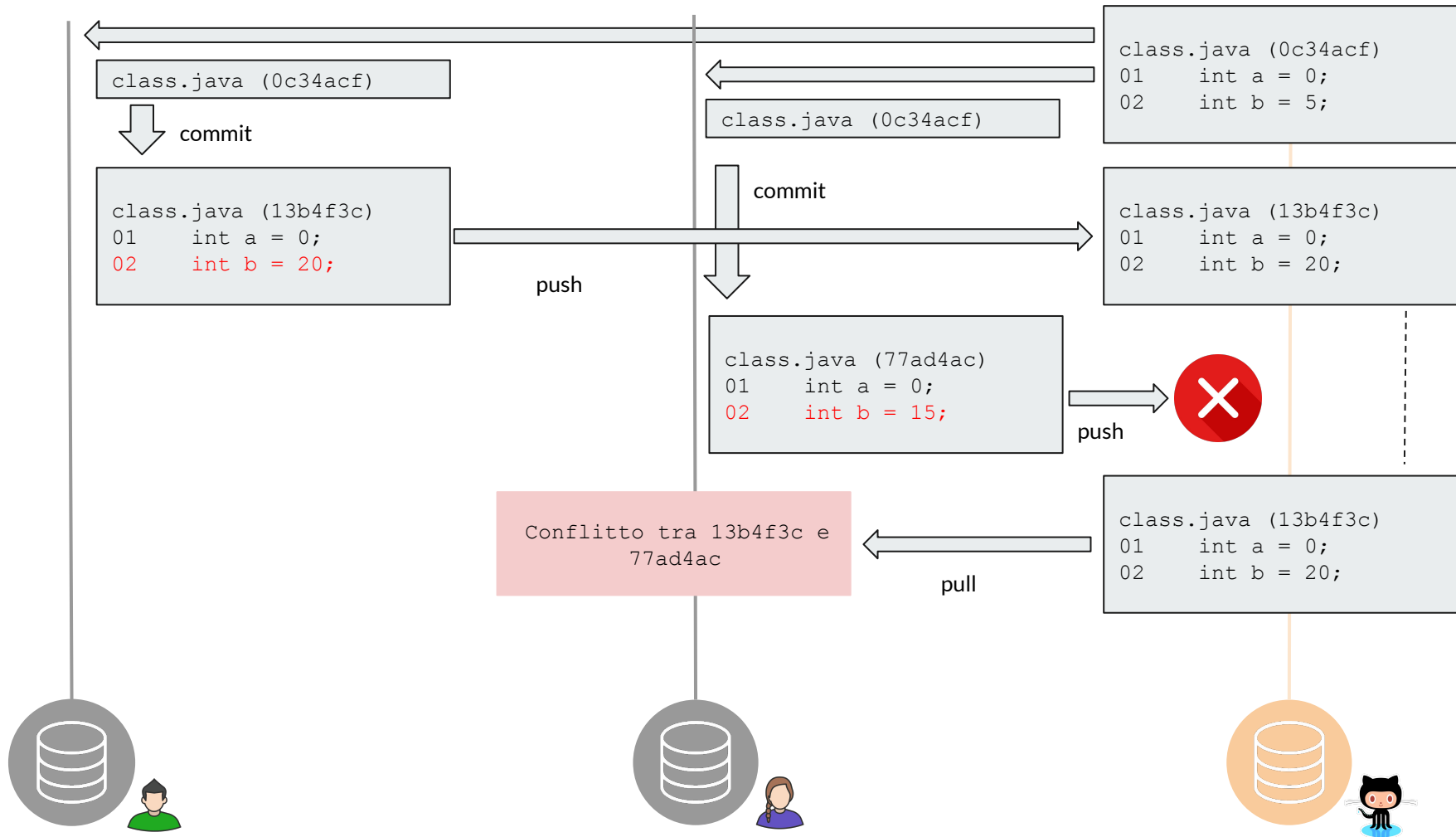














## Cos'è successo?

Git ha identificato un **conflitto**: il primo utente ha fatto un commit e il secondo utente non ha aggiornato il local repository. Subito dopo il secondo ha provato a fare una modifica a partire da un commit **meno recente** e ha modificato alcune righe del commit successivo presente su GitHub.

Non tutte le modifiche sono classificate automaticamente come conflitti: vanno bene quelle che non si danno fastidio (per esempio, aggiunte o modifiche di file completamente diversi)

Ogni volta che si fa un'operazione di pull, **si possono verificare dei conflitti**

Quando c'è un conflitto, Git modifica i file che hanno dato origine al conflitto per chiedere esplicitamente all'utente di risolverli



# Cosa fare in caso di conflitto

Quando c'è un conflitto, Git scrive nei file quali parti sono in conflitto. Nel caso di prima, le righe in conflitto sono nel file `class.java`

```
02 int b = 20;  
02 int b = 15;
```

```
class.java (13b4f3c)  
01 int a = 0;  
02 <<<<<< HEAD  
03 int b = 20;  
04 =====  
05 int b = 15;  
06 >>>>>> 77ad4ac
```

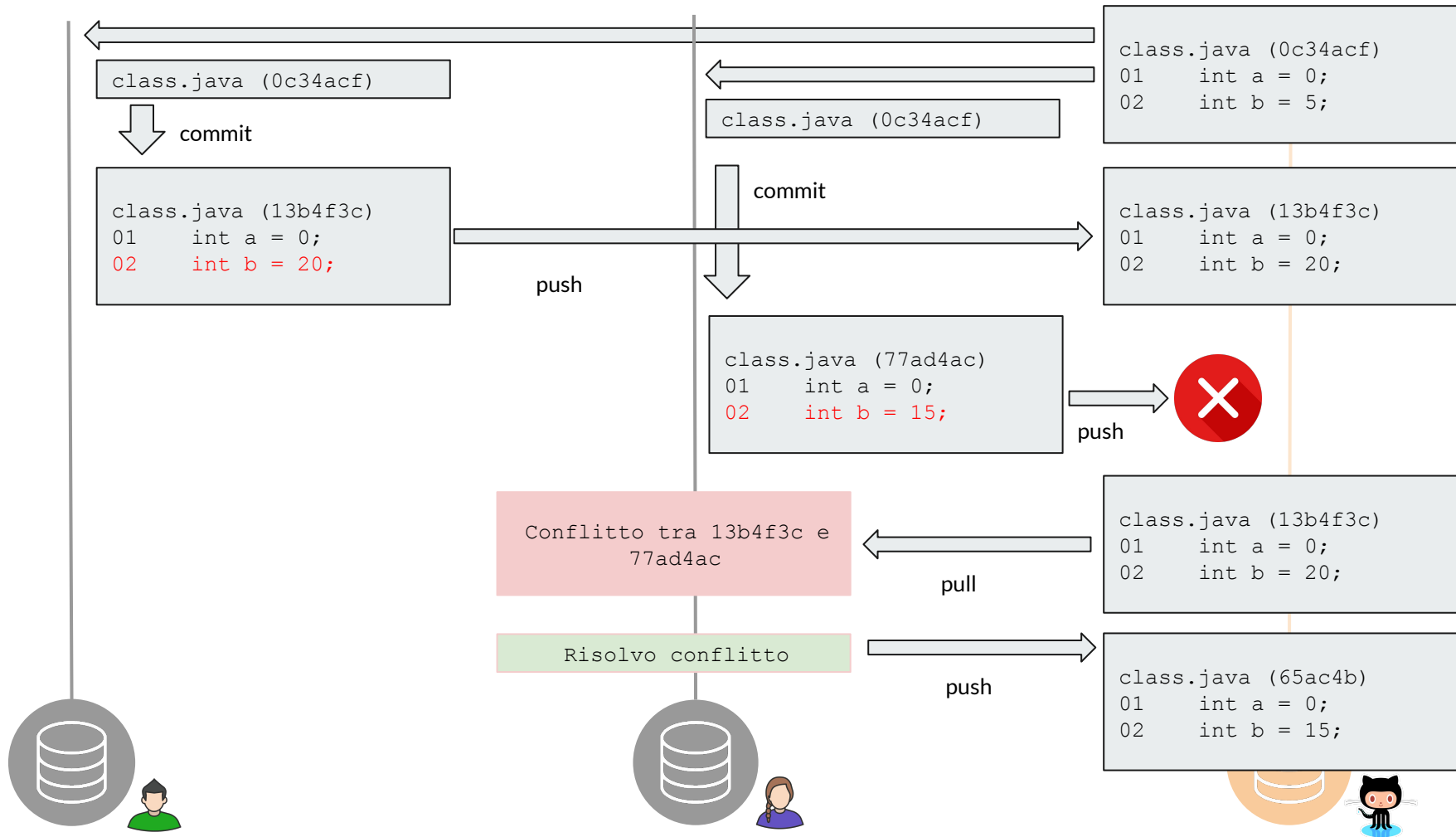
# Cosa fare in caso di conflitto

Git non risolve automaticamente i conflitti perché non è in grado di dire chi ha ragione. La responsabilità di risolvere il conflitto è della persona a cui è capitato: questa persona non deve per forza promuovere la sua versione corretta, ma deve capire cos'è successo e qual è la soluzione da adottare

```
class.java (13b4f3c)
01 int a = 0;
02 <<<<<< HEAD
03 int b = 20;
04 =====
05 int b = 15;
06 >>>>>> 77ad4ac
```



```
class.java
01 int a = 0;
02 int b = 15;
```





# Esempio

1. Modifico un file su GitHub

```
README.md
```

2. Lo modifico anche in locale

```
README.md
```

3. Lo aggiungo in staging

```
git add README.md
```

4. Lo aggiungo al repository (faccio un commit)

```
git commit -m "Modifica nome istituto"
```

5. Provo a fare un push delle modifiche

```
git push
```

6. Il push fallisce





# Esempio

7. Provo a prendere le nuove modifiche

```
git pull
```

8. Si origina un conflitto

9. Lo risolvo (cambio i file)

10. Aggiungo in staging

```
git add README.md
```

11. Faccio un commit

```
git commit -m "Modifica nome istituto"
```

12. Riporto le modifiche su GitHub

```
git push
```

---

**Proviamo!**



# Riepilogo

1. Installiamo Git e creiamo un account su GitHub
2. Breve introduzione a Git
3. Primi passi con Git: clone, add, remove, commit
4. Portare le modifiche in remoto
5. Conflitti e merge
6. Riepilogo e consigli utili

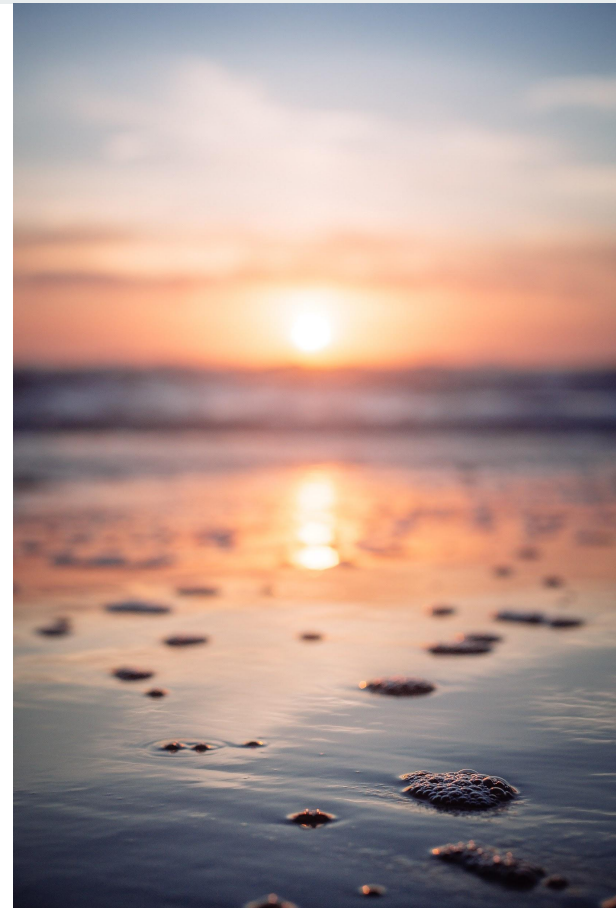


Photo by Cole Keister on Unsplash

# Cosa manca

Branch, tag, remotes

Merge e rebase

Git tools (comandi avanzati)

Git workflow



Photo by Joshua Earle on Unsplash



## Link e risorse in italiano

Libro e sito ufficiale

<https://git-scm.com/docs>

<https://git-scm.com/book/en/v2>

Slide e video

<https://channel9.msdn.com/Series/Introduzione-a-Git>

<https://www.slideshare.net/valix85/introduzione-a-git-ita-2017>

<http://marklodato.github.io/visual-git-guide/index-it.html>

Questa presentazione

<https://github.com/amaxis/git-gobetti-volta>

# **Grazie**

Massimiliano Atzori

massimiliano.atzori@gmail.com

@amaxis