
# Corso Git & GitHub - Lezione 2

## Comandi base di Git

**Stefano Faccio, Elisabetta Ferri, Giorgio Micaglio**

Associazione Italiana Studenti di Fisica
Comitato Locale di Trento

08/04/2025

# Overview

1. Inizializzazione della Repository
2. Comandi fondamentali di Git
3. Gestione dei Branch
4. Lavorare con più Branch

WARNING. NON COPIAINCOLLARE COMANDI CHE CONTENGONO IL CARATTERE - DA QUESTO FILE PDF.

# Inizializzazione della Repository

# Inizializzazione della Repository

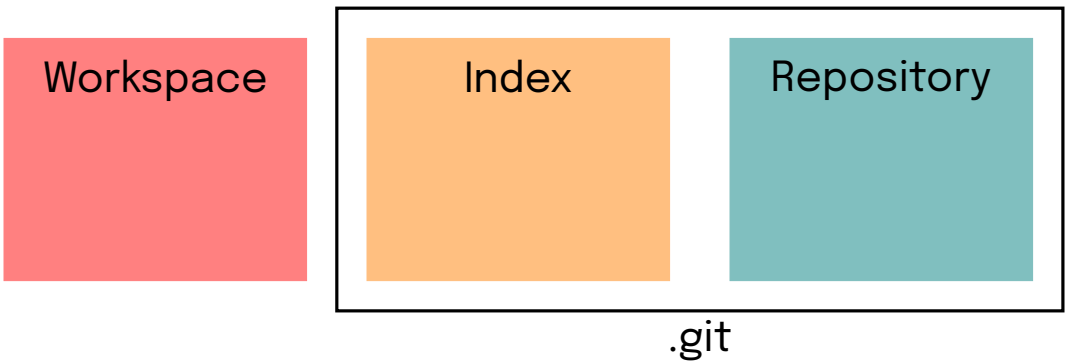
All'inizio del nostro progetto ci troviamo con una cartella. Vuota o piena che sia, questa cartella non verrà controllata da *git* a meno che non sia contenuta a sua volta in una cartella già controllata da *git*.

## Inizializzazione di *git*

Per inizializzare *git* in una cartella e permettergli di tracciarne i cambiamenti, digitare:

```
git init repository\_name
```

```
git init repository_name
```

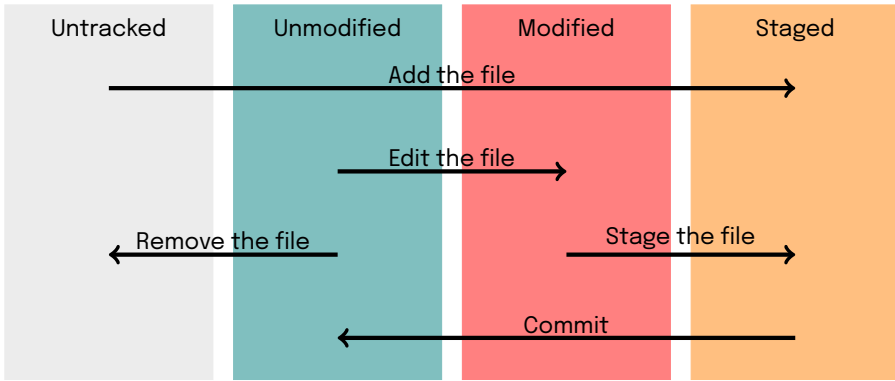


# Inizializzazione della repository

**IMPORTANTE**

Se viene specificato un `repository_name`, git inizializzerà la repository in una cartella chiamata `repository_name` all'interno della cartella in cui ci si trova mentre si chiama il comando.
Per inizializzare la repository nella stessa cartella in cui ci si trova, omettere `repository_name`. Ciò può essere utile nel caso in cui si voglia iniziare a tracciare un progetto già iniziato.

# I possibili stati di un file



# Creazione file e staging

Una volta inizializzata la nostra repository creiamo un file, ad esempio:
*data_analysis.c*

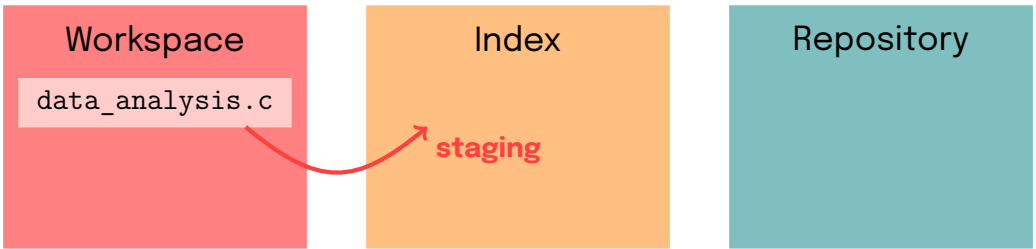
Come ricorderete dall'altra volta, questo file non è ancora tracciato da *git*.

Procediamo dunque con lo staging.

**NB:**

è possibile portare in staging anche i file presenti nella cartella prima dell'inizializzazione della repository.

```
git add data_analysis.c
```



# Commit

Una volta messi i file che ci interessa tracciare nella *Staging area*, è possibile eseguire il commit con il comando:

```
git commit -m 'Messaggio del commit'
```

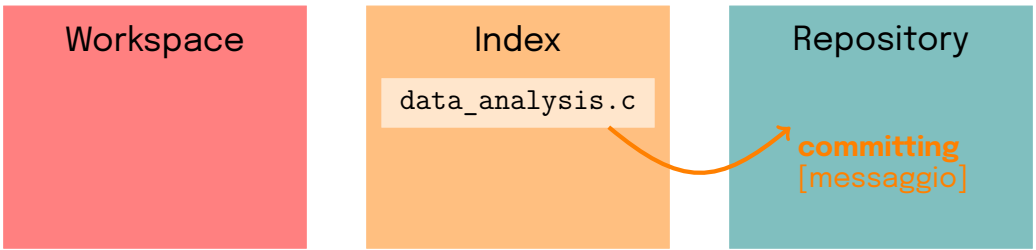
In questo modo *git* salverà lo stato di tali files.

**Domanda**

Quale messaggio del commit secondo voi è migliore?

- 'Implementato il metodo di ordinamento della lista utilizzando l'algoritmo di quicksort.'
- 'Aggiorno dei file per il sorting.'

```
git commit -m 'Messaggio del commit'
```



# Comandi fondamentali di Git

AISF, Comitato Locale di Trento

# Comandi fondamentali - git status

Il comando `git status` ci fornisce una panoramica dello stato attuale della nostra repository.
Ci mostra quali file sono stati modificati, quali sono stati aggiunti al prossimo commit e se ci sono operazioni pendenti da eseguire.

# Comandi fondamentali - git log

Il comando `git log` ci permette di visualizzare la storia dei commit nella nostra repository.
Possiamo vedere chi ha fatto quale modifica, quando è stata fatta e quali messaggi di commit sono stati associati.
Opzioni utili di git log:

- `--pretty=oneline`
- `-- file_name` mostra solo i commit che hanno modificato un certo file
- `--grep blabla` mostra solo i commit che hanno *blabla* nel messaggio di commit.

Per altre opzioni interessanti: pag. 45 del libro Pro Git

# Comandi fondamentali - git diff


- `git diff`: visualizza le differenze tra il nostro stato attuale del codice e la staging area.
- `git diff --staged`: visualizza le differenze tra la staging area e l'ultimo commit
- `git diff branch1 branch2`: visualizza differenze tra branch (ci torneremo).

Ciò ci aiuta a comprendere esattamente cosa è stato modificato nel nostro progetto.

# Note su committing e staging

- Lo staging è facilmente reversibile. Il committing no. Per rimuovere un file dalla staging area:
  `git restore --staged file_name`
- git commit aggiunge cose alla vostra repository, è MOLTO difficile rimuoverle. (`git rm` non rimuove dai precedenti commit).
  Comandamento: non committare file pesanti. (es. grandi dataset)
- Non specificando `-m 'messaggio'`, si apre un editor testo per il messaggio. Utile per scrivere una descrizione più lunga nelle righe dopo la prima.
- `git commit --amend` corregge l'ultimo commit senza crearne uno nuovo e lasciando invariato il messaggio. Utile per piccoli fix (se cambio mezza riga o mi dimentico un file).

# Altre note e comandi utili

Potete creare un file testo chiamato `.gitignore`, con una lista di nomi di file che volete siano IGNORATI da git. In questo file si possono usare wildcards.

**Esempio:**

- `*.pdf` ignorerà tutti i pdf;
- `data_*` ignorerà tutti i file che iniziano con `data_`;
- `data_*.c` ignora tutti i file che iniziano con `dara_` e hanno l'estensione `.c`.

Le wildcards sono molte di più del semplice * e possono rivelarsi molto utili in progetti complessi!

# Altre note e comandi utili

- Potete creare un file testo chiamato `.gitignore`, con una lista di nomi di file che volete siano IGNORATI da git. In questo file si possono usare wildcards.
- Per ripristinare lo stato di un file *modified* al suo stato nell'ultimo commit, `git restore file` ← N.B. questo ELIMINA DEFINITIVAMENTE le correnti modifiche al file!
- Committare spessissimo o pochissimo? C'è un giusto mezzo.
- `git rm` elimina file dalla staging area e dalla working directory. L'opzione `--cached` li lascia salvati sul vostro comptuer, ma diventeranno *untracked*.
- `git mv` usato per rinominare/spostare file, tenendo traccia di questo con git.

# Gestione dei Branch

# Cosa sono i Branch

I branch sono una caratteristica potente di Git che ci permette di sviluppare nuove funzionalità in modo isolato, senza influire sul ramo principale del progetto.
Questo ci consente di sperimentare e testare nuove idee senza compromettere la stabilità del codice principale.

## In pratica

Un branch è come avere un universo parallelo nel quale fare tutte le modifiche al proprio progetto, senza però modificare l'originale.
È molto utile quando si vuole sviluppare una nuova funzione e si desidera inserirla nel codebase solo nel momento in cui risulta stabile.

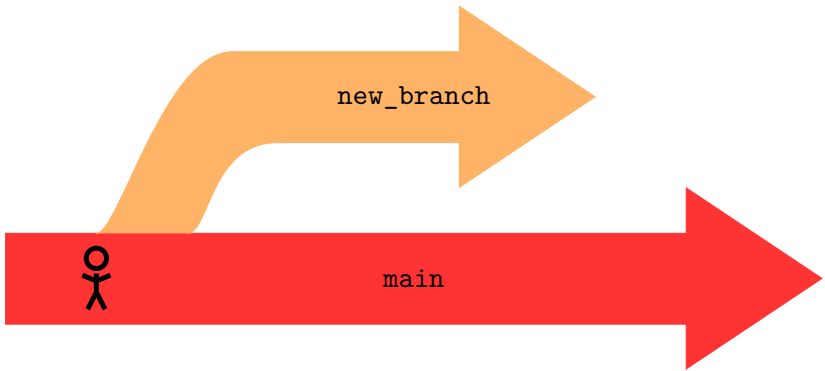
# Inizializzazione di un Branch

Per inizializzare un nuovo Branch è necessario eseguire il comando `git branch new_branch_name`.

**NB:**

Quando inizializzi la repository *git*, viene creata la branch master. Github invece utilizza come branch predefinita la main. In entrambi i casi è possibile, se necessario, cambiare il branch predefinito.

```
git branch new_branch
```

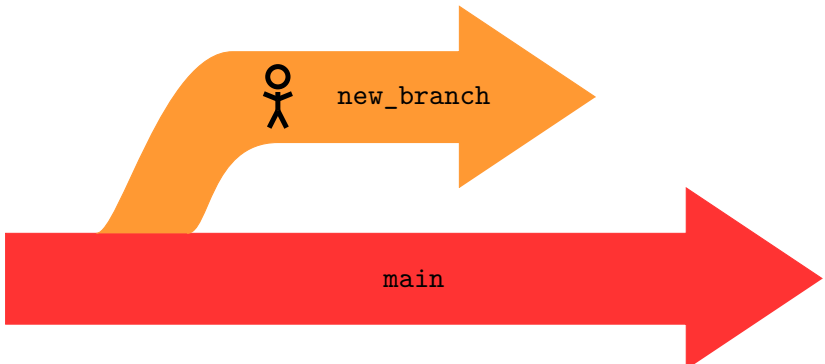


# Come muoversi tra branches

Il comando `git checkout new_branch` sposta il progetto nella branch `new_branch`.

Ciò cambierà i file presenti nella cartella qualora le due branches presentino differenze (verosimilmente sono differenti, altrimenti che le fai a fare).

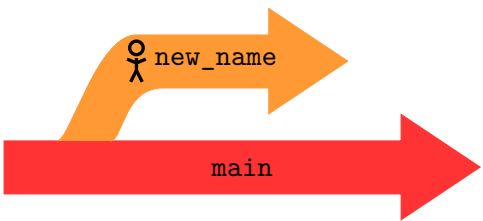
```
git checkout new_branch
```



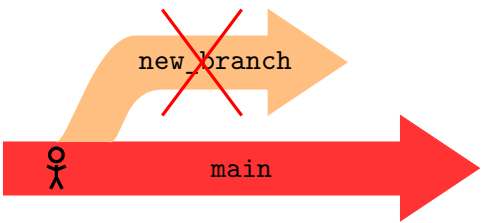
# Operazioni sui Branch

- `git branch -m new_project` cambierà il nome della brach in cui ci si trova in `new_project`;
- `git branch -m old_name new_name` invece cambierà il nome della branch `old_name`;
- `git branch -d new_branch` eliminerà la branch `new_name`.

`git branch -m new_name`




`git branch -d new_branch`



# Lavorare con più Branch

# Lavorare con più Branch

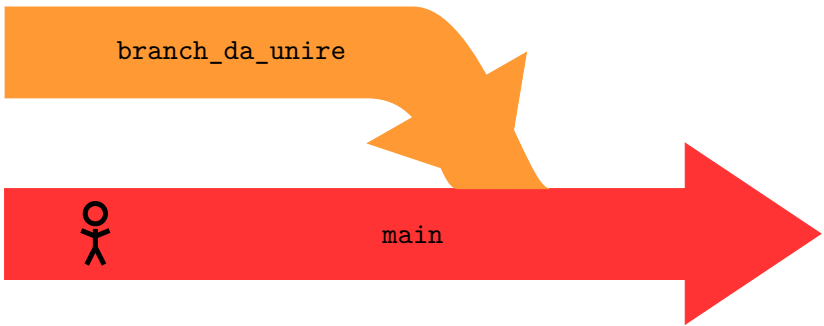
Utilizzando i branch, possiamo sviluppare nuove funzionalità in modo isolato dal ramo principale del progetto. Questo ci consente di lavorare su più aspetti del software contemporaneamente senza interferenze.

Quando abbiamo completato lo sviluppo di una nuova funzionalità su un branch separato, possiamo unire le modifiche al ramo principale del progetto utilizzando il comando

```
git merge nome_branch_da_unire
```

da dentro il ramo principale.

```
git merge branch_da_unire
```



# Risoluzione conflitti

Durante una fusione di branch, potrebbero verificarsi conflitti se le stesse righe di codice sono state modificate in modo diverso in branch diversi. Dobbiamo risolvere manualmente questi conflitti prima di poter completare la fusione.

*Git* evidenzierà tali modifiche in modo da permettere allo sviluppatore di individuarle e dunque risolverle più facilmente.
Il comando `git status` mostra quali file sono in conflitto e ogni file deve essere modificato manualmente.

# Risoluzione dei conflitti

I conflitti vengono indicati come

```
«««« HEAD
Modifiche nel tuo branch
=======
Modifiche provenienti dal branch che stai cervando di unire (merge)
»»»» branch-name
<<<<<<< HEAD
Modifiche nel tuo branch
=======
Modifiche provenienti dal branch che stai cervando di unire
(merge)
>>>>>>> branch-name
```

Una volta modificati i file in conflitto va eseguito lo staging e il committing di questi

# Thank you for your attention

**Stefano Faccio, Elisabetta Ferri, Giorgio Micaglio**

Associazione Italiana Studenti di Fisica
Comitato Locale di Trento

08/04/2025