SAPIENZA
UNIVERSITÀ DI ROMA

"Sapienza" Università di Roma
Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Informatica

Programmazione WEB

Autore
Vincenzo Bova

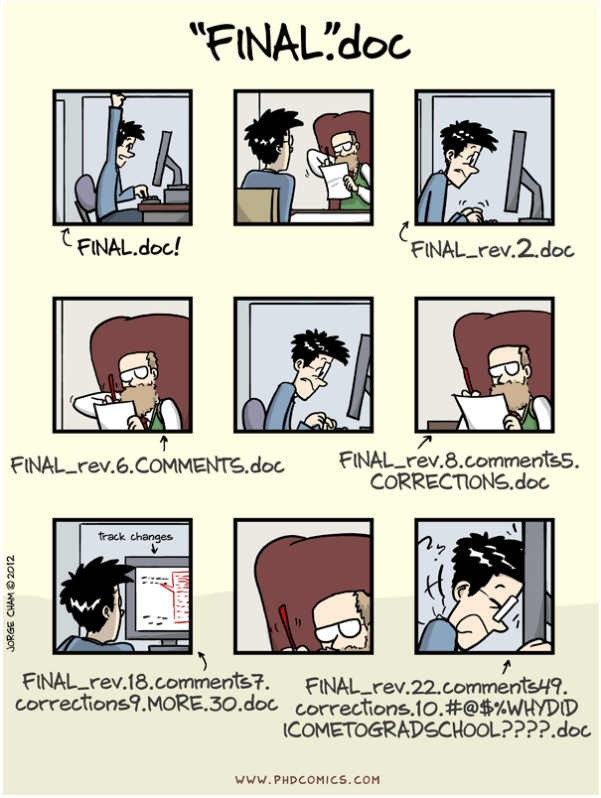
A.A. 2025/2026

Indice

1 Introduzione a Git

1.1 Sistemi di versionamento

Durante lo sviluppo di un progetto c'è spesso la necessità di effettuare revisioni, correzioni o modifiche ai file che lo compongono.



Gestire ciò creando ogni volta nuovi file, tuttavia, comporta evidenti problemi:

- **Duplicazione del contenuto:** che rende il sistema inefficiente e aumenta la difficoltà nel mantenere integrità;
- **Assenza di Naming Convention:** che rende impossibile risalire ad uno storico delle modifiche;
- **Autori incerti**;
- ...

Per ovviare a ciò sono stati creati i **sistemi di versionamento** (git, csv, mercurial, svn...), i quali offrono vari benefici:

- **Gestione delle versioni:** il sistema si occupa automaticamente di etichettare le varie versioni in modo consistente;
- **Tracciamento delle mofiche:** è possibile accedere ad uno storico delle modifiche effettuate;
- **Presenza di metadati:** ogni modifica ha un autore, una data...;
- **Creazione di linee di sviluppo parallele:** è possibile creare una versione parallela del codice per non modificare la versione principale, e poi riunirle integrando i cambiamenti;
- **Sincronizzazione tra computer:** il sistema consente di mantenere il progetto allineato tra più computer.

1.2 Git

Git è un sistema di versionamento distribuito e veloce, creato nel 2005 e capace di gestire progetti di grandi dimensioni. Si basa su un design semplice e utilizza DAG (*Directed Acyclic Graph*) e Merkle trees come strutture dati.

Definizione 1.2.1 : Repository

È un insieme di commit, branch e tag.
Per semplicità assumiamo che un progetto equivale ad un repository.

Definizione 1.2.2 : Working copy

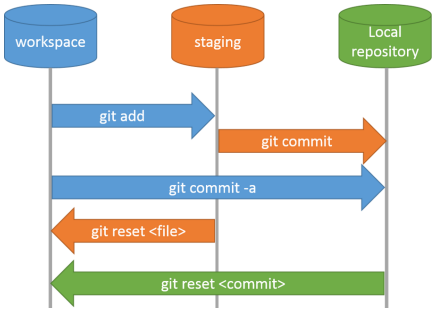
È un insieme di file tracciati nella copia locale.
Un nuovo file non sarà ancora tracciato e bisognerà aggiungerlo.
Quando aggiorniamo i file (*update*), stiamo aggiornando la working copy.

1.2.1 Commit

Un commit è un'istantanea del repository in un determinato momento.
Viene identificato dallo **SHA1** del commit stesso e contiene diversi campi:

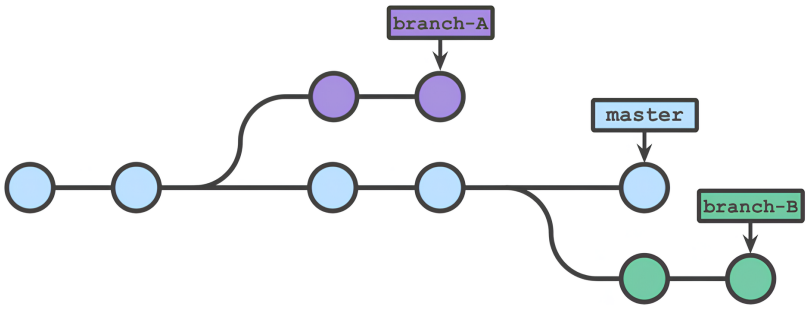
- data + autore, data + commiter
- commento **obbligatorio**
- 0,1 o più genitori
- tree: hash di tutti i file nel commit

In particolare il commit può contenere un sottoinsieme delle modifiche (anche ad un singolo file), le quali devono essere aggiunte alla staging area dei cambiamenti.



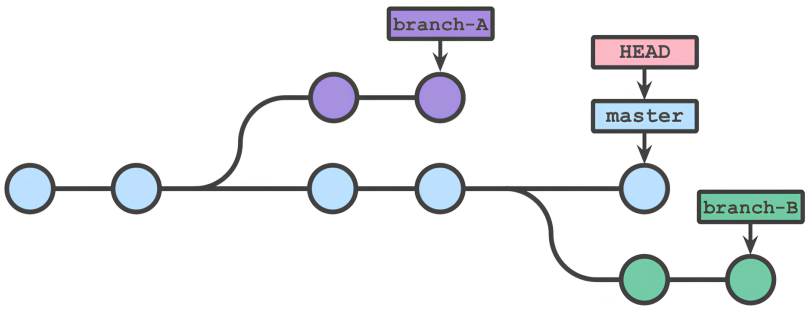
1.2.2 Branch

Un branch è una linea di sviluppo, composta da un insieme ordinato di commit collegati in un DAG, il quale inizia dal primo commit del repository e punta all'ultimo commit. Grazie ai branch è possibile **lavorare parallelamente** a più versioni del progetto.



1.2.3 HEAD

L'HEAD è un puntatore alla posizione attuale rispetto alla storia del repository e può essere aggiornato tramite il comando *checkout*.
Solitamente l'HEAD punta ad un branch o ad un tag, qualora invece puntasse ad un commit si parlerebbe di **Detached HEAD**. Quando ci si trova in questo stato i commit fatti non vengono inseriti in alcun branch, rischiando quindi di andare persi.



1.2.4 Tag

Un tag è un'etichetta per un commit e viene solitamente usato per segnare versioni importandi di un progetto (e.g. *v1.0.0*, *release-2025-09*).

1.2.5 Merge

Il merge è un'operazione che fonde i cambiamenti tra due branch, facendo in modo che la destinazione contenga entrambi i cambiamenti mentre l'origine rimanga immutata. Può avvenire mediante tre strategie:

- **Fast forward:** quando il branch di destinazione non ha commit successivi rispetto a quello che si vuole unire e il sistema sposta semplicemente il puntatore in avanti;
- **Merge commit:** quando i branch hanno sviluppi indipendenti e il sistema crea un nuovo commit con due genitori;
- **Rebase:** il sistema ricrea ogni commit non in comune tra i due branch, mantendo così una storia lineare.

1.3 Comandi Git

Per interfacciarsi con Git vengono messi a disposizione dal sistema diversi comandi:

- `git init`: inizializza un repository creando una subdirectory .git all'interno della directory corrente;
- `git status`: mostra lo stato attuale del repository (file tracciati, file modificati, file nello staging, file non tracciati);
- `git diff`: mostra le differenze tra working directory, staging e commit;
- `git add <file>`: aggiunge un file alla staging area (`git add .` per aggiungere tutti i file modificati);
- `git commit -m "Messaggio"`: crea un commit, registrando le modifiche aggiunte con `git add` nella cronologia del repository;
- `git log`: mostra la lista dei commit effettuati;
- `git branch <nome>`: crea un nuovo branch con il nome indicato, ma **non ci si sposta**;
- `git checkout <nome>`: passa ad un branch esistente spostando l'HEAD;
- `git checkout -b <nome>`: crea un nuovo branch con il nome indicato, per poi **spostarsi** su quest'ultimo (`git checkout -b <nome>` = `git branch <nome>` + `git checkout <nome>`);
- `git fetch`: scarica gli aggiornamenti (commit, branch) dal repository remoto, **senza merge** col tuo branch;
- `git merge <branch>`: unisce la cronologia del branch in cui ci si trova con quella del branch specificato;
- `git pull`: scarica gli aggiornamenti (commit, branch) dal repository remoto, **facendo merge** col tuo branch (`git pull` = `git fetch` + `git merge`);
- `git push`: invia i commit locali al repository remoto, aggiornando il branch remoto corrispondente;

1.4 Git flow