UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dispense per il Laboratorio di Strutture Dati e Algoritmi

Federico Bolelli

Esercitazione 05: Introduzione a *git*

Ultimo aggiornamento: 30/03/2023

Perché ci interessa?

- Immaginate di dover consegnare entro oggi il codice di un progetto per passare un esame. Il software funziona ed è pronto per la sottomissione.
- Mentre state eseguendo le ultime verifiche scoprite che c'è un piccolo bug e decidete di sistemarlo.
- Durante la correzione del bug modificate accidentalmente un pezzo di codice funzionante. Quel codice era stato scritto diversi giorni prima e non vi ricordate più com'era e come dovrebbe essere.
- …
- Sono le 23.58

Perché ci interessa?

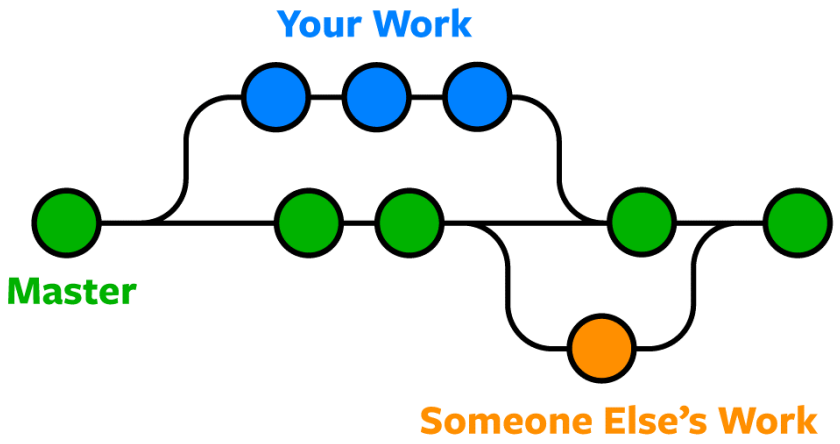
- e avete appena realizzato che CTRL + Z non risolverà i vostri problemi.

Perché ci interessa?

- Sono ormai cinque mesi che state lavorando alla vostra tesi.
- Il codice è pronto, vi manca solo l'elaborato e il gioco è fatto, ma una sera mentre state giocando a Minecraft il computer si spegne.
- Il disco è rotto.
- Era un po' che pensavate di fare una copia del codice su chiavetta, ma non avete mai trovato il tempo e la voglia di farlo.

Perché ci interessa?

- Tiene traccia delle modifiche apportate ai file, fornendo una fotografia nel tempo di ciò che è stato fatto.
- Se necessario è possibile tornare a versioni specifiche di alcuni determinati file, o di tutto il progetto.
- Semplifica la collaborazione tra più persone consentendo di unire tutte le modifiche effettuate.



Ma Cos'è Git?

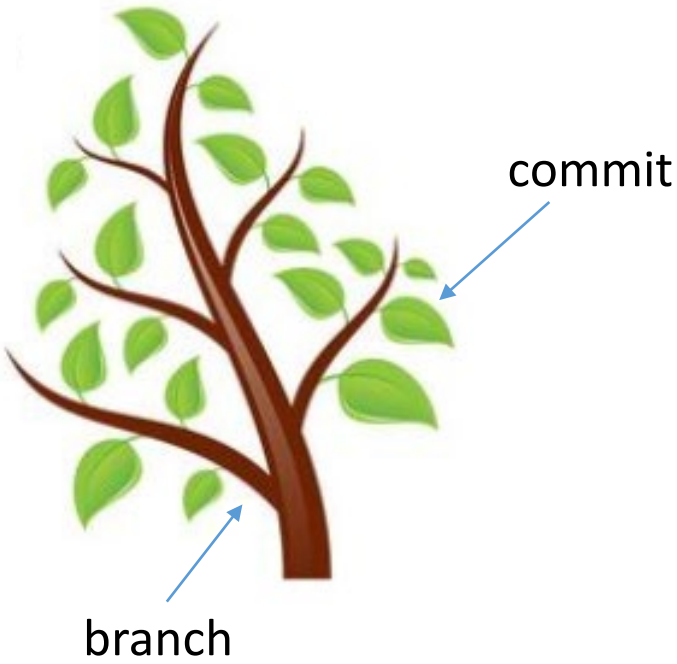
- *git* è un progetto open-source originariamente sviluppato da Linus Torvalds nel 2005.
- Si tratta di una utility da linea di comando.
- Potete immaginare *git* come qualcosa che lavora al di sopra del vostro file system e che si occupa di gestire dei file.
- Nello specifico *git* è un Distributed Version Control System (DVCS).

Version Control System

- Un sistema di controllo delle versioni si occupa di memorizzare i cambiamenti su un file o un gruppo di file nel tempo, così che sia possibile recuperare una specifica versione in qualunque momento.
- I sistemi di controllo delle versioni possono essere:
  - **Locali**: gestiti localmente sulla singola macchina;
  - **Centralizzati**: gestiti in remoto su un server centralizzato per favorire la collaborazione tra più sviluppatori;
  - **Distribuiti**: un server remoto gestisce la copia principale, ma ogni client ha tutte le informazioni sulla storia del progetto. Favorisce collaborazione e tolleranza ai guasti.

git

- Potete immaginare git come qualcosa che lavora al di sopra del vostro file system e che si occupa di gestire dei file.
- Questo qualcosa è una struttura ad albero dove ogni **commit** crea un nuovo nodo nell'albero.
- Quasi tutti i comandi git servono per navigare questo albero e modificarlo opportunamente.
- Come detto l'obiettivo è quello di gestire un progetto (insieme di file) e tracciarne le modifiche nel tempo.



Partiamo da Qualche Definizione

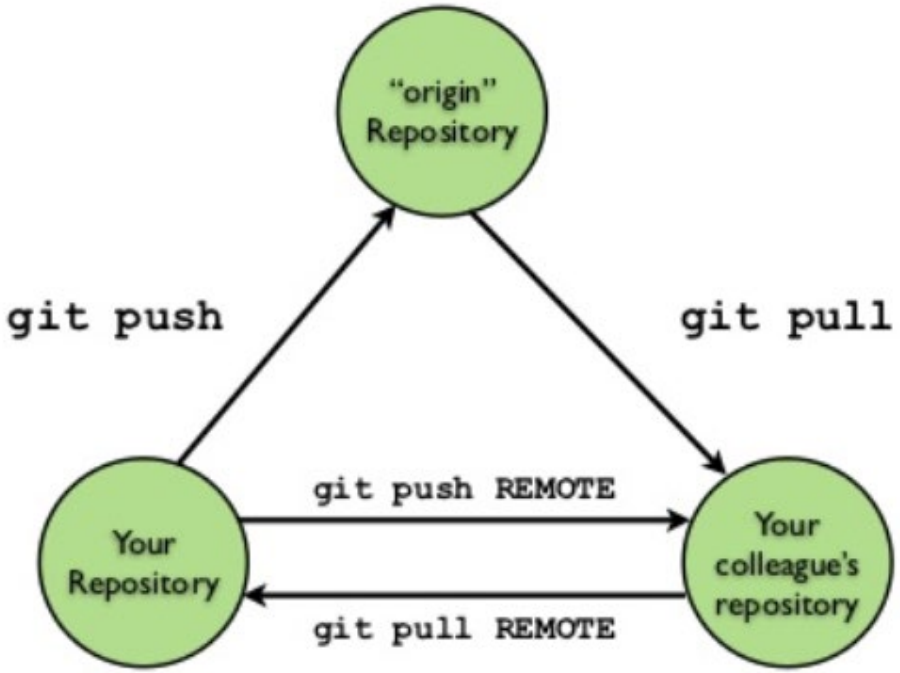
- Il **working tree** è una cartella nel nostro file system che rappresenta un progetto. Questa cartella può contenere file e sottocartelle ed ha un **repository** associato.
- Il **repository** è una collezione di **commit** e **branch** salvati nella cartella **.git** all'intero del **working tree**.
- Una **commit** è una fotografia del working tree in un determinanto istante di tempo. La **commit** è identificata da un numero di revisione.
- **HEAD** è un riferimento all'ultima commit effettuata sul branch corrente.

Numero di Revisione

- Ogni operazione di commit è identificata da un identificatore globalmente univoco.
- L'identificatore è un *hash SHA-1* di ognuna delle cose importanti che costituiscono l'operazione di commit.
- Le più importanti sono:
  - Il contenuto dei file, non solo le differenze rispetto alla versione precedente.
  - La data della commit.
  - Il nome e l'indirizzo email di colui che ha eseguito la commit.
  - Il messaggio di log.
  - L'ID delle operazione di commit precedenti.

Un Passo Indietro

- Abbiamo detto che git è un DVCS: un server remoto gestisce la copia principale, ma ogni client ha tutte le informazioni sulla storia del progetto. Favorisce collaborazione e tolleranza ai guasti.
- Il modello può essere rappresentato come nello schema qui sotto.
- **origin** è la copia principale del repository gestita dal server remoto.
- Esistono diverse piattaforme che si occupano di gestire la copia principale di un repository git.



Servizi di Hosting

GitHub: https://github.com/

GitLab: https://about.gitlab.com/

Bitbucket: https://bitbucket.org/

SourceForge: https://sourceforge.net/

GitHub

- Per creare un nuovo account: https://github.com/join?source=header-home
- Per richiedere i *benefit education*: https://education.github.com/discount_requests/new

- Una volta completata la registrazione, fate il login e create il vostro primo repository.
- Vediamo come …

Installare git

- Utenti Windows:
  - https://git-scm.com/download/win
- Utenti Unix:
  - …

Iniziamo

- Una volta installato **git** potete utilizzare le sue funzionalità tramite il prompt dei comandi di Windows.
- Impostare nome ed email da utilizzare nelle commit:
  - `git config --global user.name "Bugs Bunny"`
  - `git config --global user.email bugs@gmail.com`
- Clonare un repository remoto o creare un nuovo repository locale:
  - `git clone`
  - `git init`
- Facciamo qualche prova …

I File in git

- Aggiungere o modificare un file nella cartella di un progetto (working tree) non significa aggiungere il file o le modifiche al repository.
- Un file può trovarsi in diversi stati:
  - **untracked**: il file si trova nel working tree ma NON fa parte del repository;
  - **tracked**: il file si trova nel working tree e fa parte del repository;
- Un file **tracked** può essere:
  - **staged**: pronto per essere inserito in una commit;
  - **unstaged**: le modifiche non saranno aggiunte alla prossima commit;
- Per vedere lo stato dei file in un repository si può utilizzare il comando `git status`.

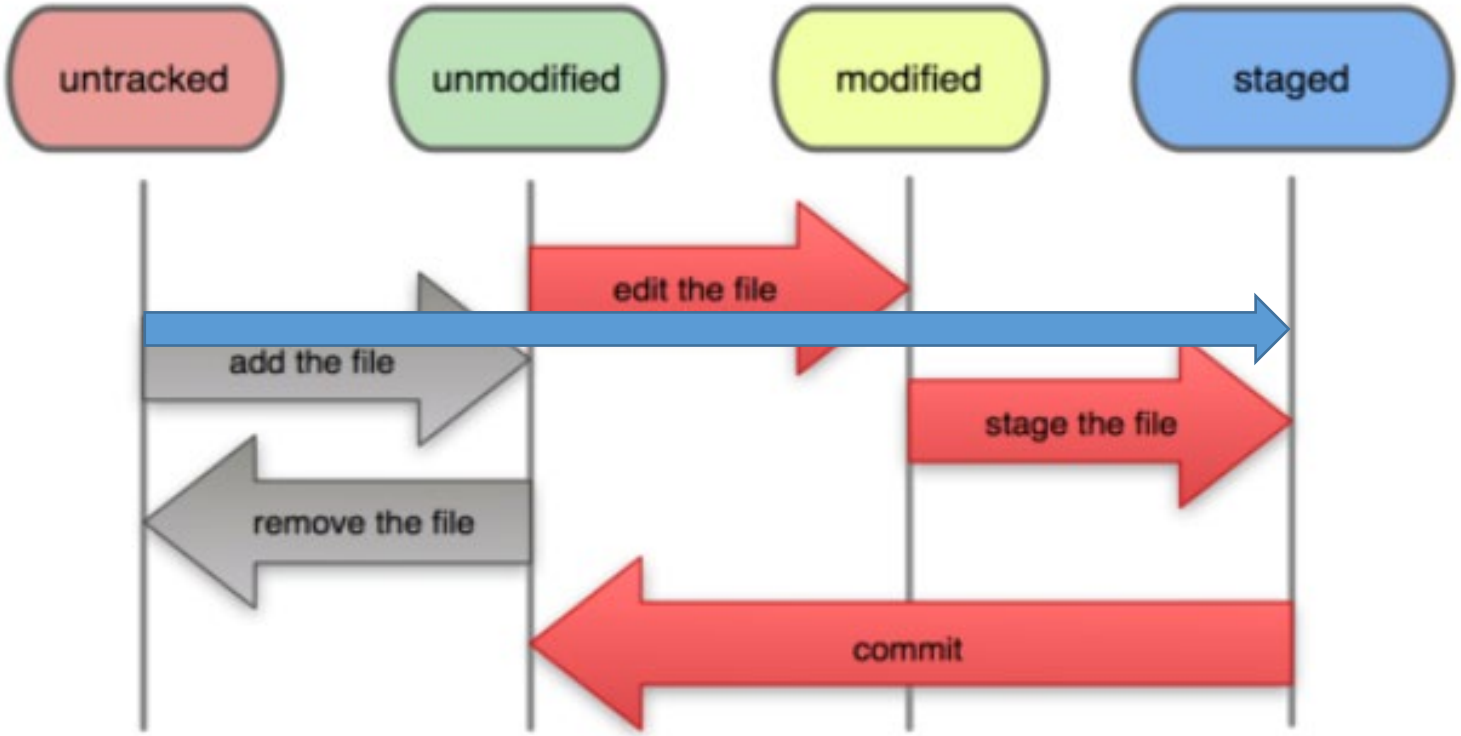
Add and Commit

- `git add <path>`: permette di portare uno o più file allo stato staged.
- `git commit –m <message>`: crea una nuova commit contenente tutte le modifiche ai file in stato staged.
- **ATTENZIONE! commit = locale**
- `git push`: permette di caricare le commit locali nell'origin.
- `git pull`: permette di scaricare le commit nell'origin in locale.
- Le commit sono veloci ed economiche.
- **Fate quante più commit possibile!**

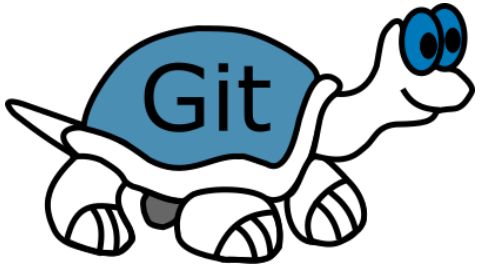
Ricapitolando

- La modifica/aggiunta/rimozione di un file cambia il working tree.
- Per poter *committare* i cambiamenti dobbiamo prima di tutto aggiungerli all'**index** (`git add <path>`) o rimuoverli dall'**index** e dal working tree (`git rm <path>`).
- `git status` mostra lo stato corrente dell'index.
- `git commit -m <message>` effettua la commit dei cambiamenti salvati nell'**index** e pulisce l'**index** subito dopo.
- Volendo, si può saltare un passaggio e *committare* tutte le modifiche non salvate nel working tree, senza bisogno di aggiungerle all'**index**: `git commit -a -m <message>`.
- Ci serve comunque `git rm` per rimuovere i file.

Il Ciclo di Vita di un File



TortoiseGit



- Prima di proseguire semplifichiamoci la vita!
- Scarichiamo e installiamo TortoiseGit: https://tortoisegit.org/
- Di cosa si tratta:
  - È un'interfaccia di Windows per Git;
  - Non è integrata con un IDE specifico come Visual Studio o altri, quindi molto versatile.
  - Ci permette di fare tutto (o quasi) quello che possiamo fare da linea di comando, ma in maniera *user-friendly*
- Vediamo come …

Branching and Merging

- I **branch** vengono utilizzati per distinguere task differenti e non sono altro che delle commit speciali con nome.
- Di default si lavora sempre sul branch `master` (o `main`)
- `git branch` mostra su quale branch sto lavorando (*) ed elenca tutti quelli disponibili.
- È possibile creare nuovi **branch** sul repository locale:
  - `git branch <branch_name>`
- e spostarsi da un **branch** all'altro (sempre in locale):
  - `git checkout <branch_name>`

Branching and Merging

- È possibile *pushare* tutte le commit del **branch** corrente locale in un **branch** remoto:
  - `git push <remote_branch_name> <local_branch_name>`
- O *pullare* un **branch** remoto in locale:
  - `git checkout <remote_branch_name>`

- Per effettuare il **merge** dei cambiamenti effettuati su un branch locale nel master locale devo:
  - `git checkout master`
  - `git merge <branch_name>`

Merge Conflicts

- A volte *git* non riesce a risolvere autonomamente i conflitti che si vengono a creare in commit diverse dello stesso file.
- Quindi? Dobbiamo farlo a mano!
- I file in conflitto conterranno delle sezioni <<<< e >>>> che indicano appunto i conflitti che dobbiamo risolvere.

```
<<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=======
<div id="footer">
  thanks for visiting our site
</div>
>>>>>>> SpecialBranch:index.html
```

branch 1's version

branch 2's version

- Quello che dobbiamo fare è trovare queste sezioni ed editarle opportunamente.

.gitignore

- Il file *.gitignore* specifica i file untracked del working tree che devono essere ignorati.
- I file ignorati non entreranno mai a far parte del repository.
- Vediamo alcune regole per la costruzione del .gitignore:
  - Una riga vuota non corrisponde a nessun file.
  - La barra / viene utilizzata come separatore di cartelle. I separatori possono verificarsi all'inizio, al centro o alla fine della regola di *ignore*.
  - Se una / si trova alla fine di una regola, verranno ignorate solo le cartelle che soddisfano quella regola.
  - Un asterisco "*" corrisponde a tutto tranne che a una barra. Il "?" corrisponde a qualsiasi carattere tranne "/".

.gitignore

- O ancora:
  - La notazione dell'intervallo [a-zA-Z] può essere utilizzata per fare match con un carattere qualsiasi negli intervalli specificati.
  - Un "**/" iniziale indica che la regola che segue si riferisce a qualunque percorso nel working tree.
  - Un "/ **" finale significa che deve essere ignorato il contenuto della directory che fa match con la regola che precede "/ **"
  - Una barra seguita da due asterischi consecutivi e un’altra barra corrisponde a nessun o più cartelle intermedie. Ad esempio, "a / ** / b" corrisponde a "a / b", "a / x / b", "a / x / y / b" e così via.