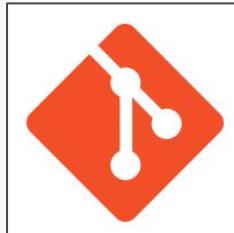


Git - Version Control

Tracking delle modifiche e condivisione del codice



Perchè serve?

Per rispondere (tra le altre) alle seguenti situazioni:

- Quale modifica avevo appena fatto?
- Come mi piacerebbe poter tornare a prima quando almeno funzionava tutto!
- Cosa avevo scritto quando quel div era posizionato bene?
- ...e *tante altre domande figlie delle frustrazione*

Git è uno strumento che, opportunamente utilizzato, permette di rispondere alle domande sopra e molto di più.

Git

E' un ***distributed version control software***:

- Tiene traccia delle modifiche
- Permette di annullare le modifiche e tornare ad uno “stato” precedente
- Permette di collaborare in più persone sul codice
- Se collaborativo, permette di vedere chi ha fatto quali modifiche



Git

- Iniziato a sviluppare da *Linus Torvalds* nel 2005 per aiutare lo sviluppo del *Kernel Linux*
 - Chi è Linus?
 - Cos'è Linux?
 - Cos'è un Kernel?
- Esistevano già altri VCS ma più *primitivi*, git è stato costruito per avere le seguenti caratteristiche:
 - Speed
 - Simple design
 - Strong support for non-linear development (thousands of parallel branches)
 - Fully distributed
 - Able to handle large projects like the Linux kernel efficiently (15+ million LOC)
- E' lo standard *de facto* del VCS:
 - Se lavorerete in una azienda che usa *subversion*, devono esserci dei motivi convincenti, altrimenti...RUN!

Ma non è un po' troppo complicato?

Una doverosa premessa

SI è complicato, soprattutto all'inizio e non sembra che ne valga la pena MA...

NON E' COSI'

Git è l'unico strumento in grado di garantire che il nostro lavoro non venga perso, che ci sia uno stato funzionante sempre recuperabile e che sia possibile condividere / chiedere aiuto sul codice in maniera efficiente

"[DVCS is] possibly the biggest advance in software development technology in the [past] ten years"
Joel Spolsky - Trello, Excel, StackOverflow Q&A



Git - Init

Workflow

- **Inizializzazione del repository**
 - Da fare solo all'inizio di ogni progetto, significa: "ok, questa è la cartella che dovrai guardare"
 - Da questo momento ogni modifica sarà tracciata

Repository (o repo): è l'insieme di tutti i file del progetto e la loro storia

Git - Staging

Workflow

- **Inserimento dei file nella sezione di staging**
 - Git controlla tutti i file e segnala quali sono quelli che sono stati modificati, rispetto all'ultimo stato
 - E' nostro compito selezionare quelli che logicamente fanno parte della stessa modifica inserendoli nell'area di staging
 - (esempio: per aggiungere un footer ho aggiunto il tag footer modificando il file index.html e aggiunto il selettore footer nel file style.css. Index.html e style.css sono entrambi da inserire nell'area di staging per la modifica "aggiunta del footer")

Staging: è l'area di sosta comune per tutti i file che concorrono alla stessa modifica.

Git - Commit

Workflow

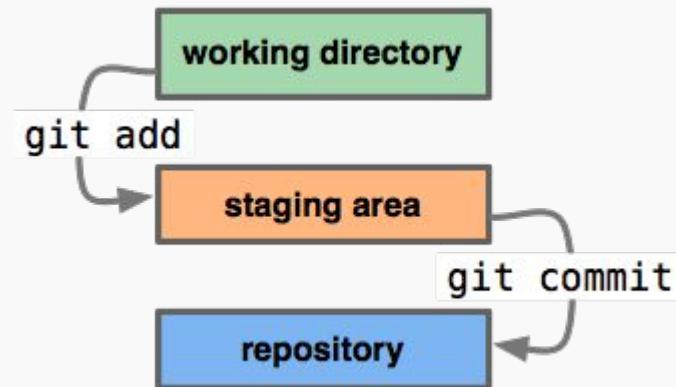
- **Creazione di un commit**
 - Quando tutti i file necessari a fare una modifica sono stati modificati e inseriti nell'area di staging, si può creare un commit.
 - Git prende tutte le informazioni dei file nell'area di staging e crea un “*punto di recupero*” del progetto.
 - Questo punto di recupero è un *commit* e sarà possibile tornare allo stato del progetto al momento del commit in qualsiasi momento

Commit: è l'atto di creare uno stato recuperabile del progetto.

Può essere sia un verbo che un nome (“ho commitato” oppure “quel commit”

Git - Recap

Modifiche -> Stage area -> Commit



Git clients - Git UI / Git CLI

Come si fanno queste operazioni?

- **Git** è il nome del software che gestisce i file, tiene traccia delle modifiche etc.. ma come si fanno le operazioni di stage, commit, ... ? Con un *git client*
- Un git client è un software che interagisce con git per fare le operazioni, ce ne sono diversi, di diverso tipo:
 - **git-cli**: il client ufficiale, utilizzabile solo da riga di comando (*command line interface*)
 - Diversi **git client stand-alone con interfaccia grafica**, utilizzabili come un qualsiasi altro programma (*smartgit*, *git tower*, *source tree*, ...)
 - Git client integrati negli IDE: in tutti è installabile, in alcuni è già installato (hai mai provato a cliccare su “*0 files*” in basso a destra su Atom?)

Git clients - La scelta consigliata

- **Git client integrato nell'IDE:** per le operazioni semplici quotidiane (staging, commit)
- **Standalone git client:** per le operazioni più complesse (merging, branching, rebasing, ...).
La mia scelta è *smartgit* (<https://www.syntevo.com/smartgit>) free for non-commercial use e utilizzabile su Windows, MacOS, Linux

Esempio Live

Il nostro primo commit:

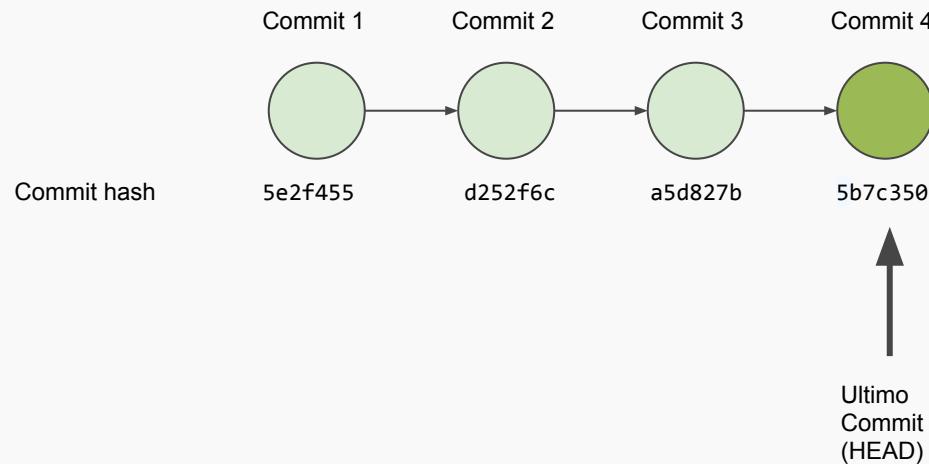
- CLI
- Smartgit

Git - Reset

- Per tornare ad un commit precedente bisogna eseguire una operazione di reset, indicando l'hash del commit a cui si vuole ritornare
 - *Hard Reset*, tutte le modifiche successive al commit vengono eliminato
 - *Reset*, tutte le modifiche successive al commit vengono messe in staging

Reset: operazioni per riportare il repository ad uno stato precedente

Git - Commits



Git - Remote

- E' possibile inviare la copia del nostro repo ad un server esterno
 - Per sicurezza (se il codice fosse solo sul nostro PC e questo si rompesse?)
 - Per facilitare la condivisione del codice con altre persone

Repository remoto (o remote): è una copia del repo locale su un server esterno (remoto)

Git - Remote



GitHub

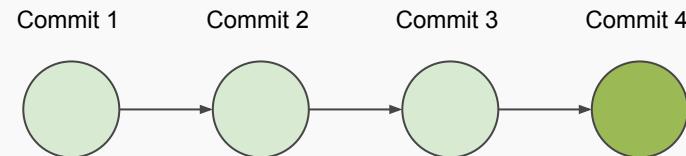
Il più famoso repository remoto git.

Oltre ad essere ottimo per tenere al sicuro il proprio codice, github aggiunge funzionalità social per condividere codice, seguire le attività di sviluppo di altri progetti e contribuire

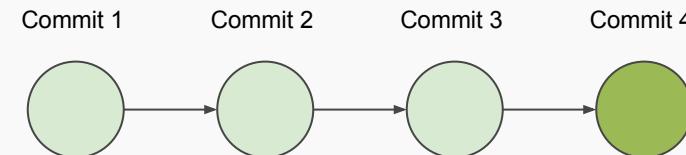
Inoltre, per i developer, è una sorta di portfolio dei propri progetti (come *behance* per i creativi) utilizzato anche da HR per fare scouting o valutare i candidati.



Git - Remote



Repo Locale



Repo Remoto
su Github

Git e GitHub

Come li utilizzeremo

GitHub diventerà il nostro unico modo di scambiarci progetti e consegnare gli esercizi
(niente più .zip e file via slack, yay!)

Ogni studente dovrà utilizzare git sul proprio pc per gestire il proprio codice e sincronizzare il proprio repo con github.

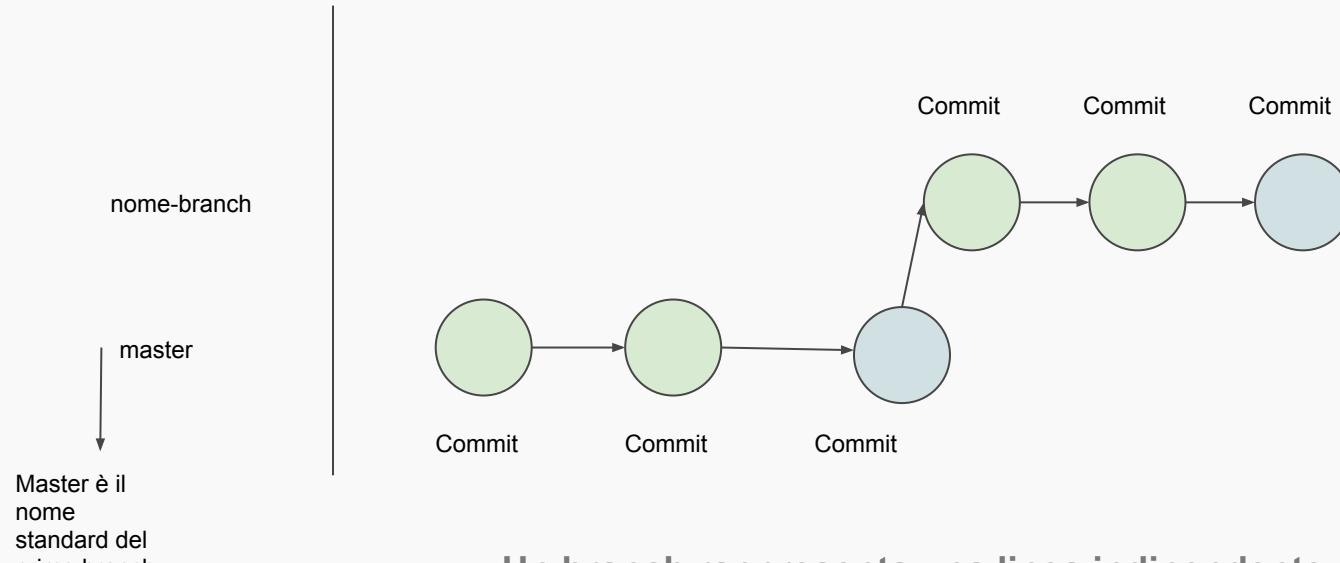
Questo porterà alcuni vantaggi:

- Per gli studenti, tutti i vantaggi detti per git
- Per il professore, la possibilità di accedere al codice dello studente in maniera semplice e di fornire supporto in maniera più veloce

Setup Live

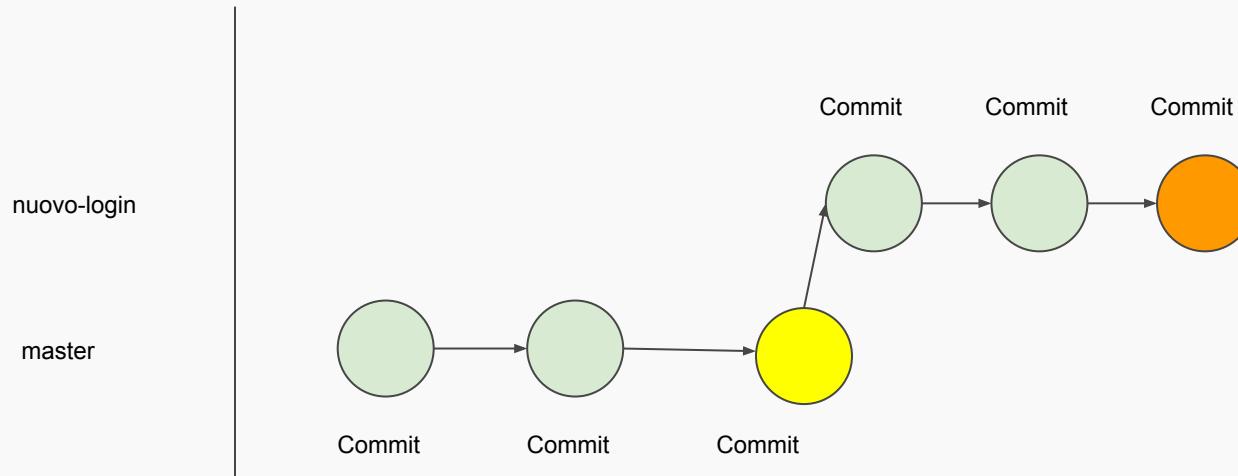
Creiamo tutti Live un account github
e impostiamo Atom

Git - Branching



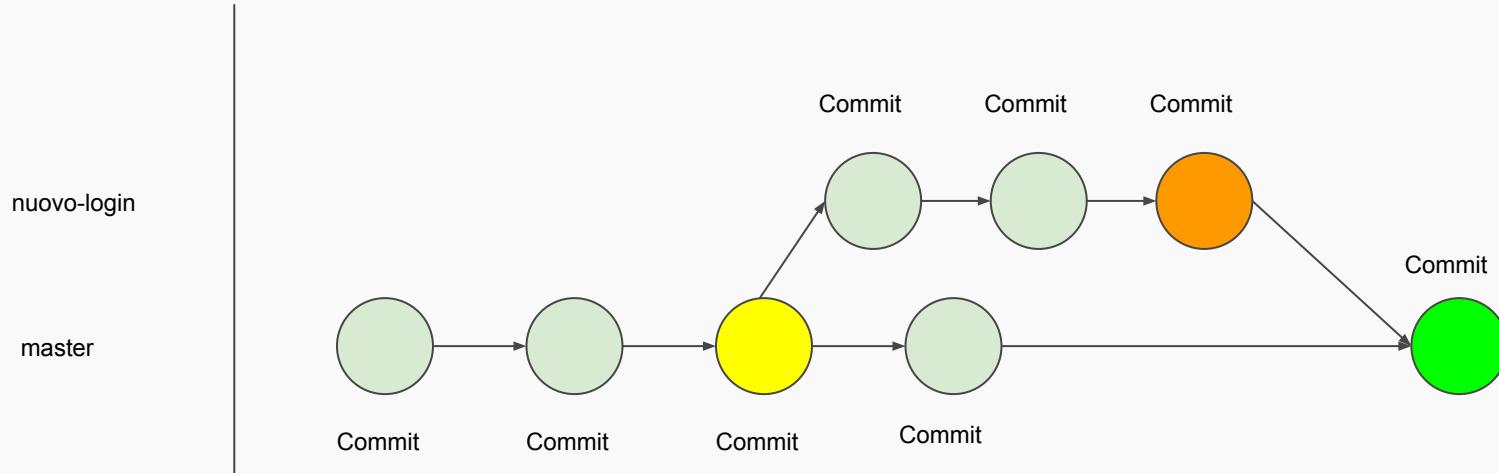
**Un branch rappresenta una linea indipendente di sviluppo.
E' possibile passare da una linea all'altra in ogni momento**

Esempio branching



Il mio sito è online e ora voglio lavorare su un nuovo processo di login che so richiederà alcuni giorni di tempo per essere completata. Creo un nuovo branch che chiamo “nuovo-login” creando una *nuova versione* del progetto. Quando lavorerò sul nuovo login farò modifiche al branch nuovo-login. Se devo fare modifiche alla versione online del sito, mi basterà cambiare branch (tornando a master) e riporterò il progetto senza le modifiche fatte in nuovo-login (*nodo giallo*)

Git - Merging



Le funzionalità di *nuovo-login* sono completate e sono pronte per andare sul sito. Non c'è più bisogno di mantenere due versioni diverse del progetto quindi le nuove funzionalità possono essere inserite (*mergiate*) nella versione master. Git lo fa automaticamente tranne quando viene modificata la stessa riga in due modi diversi, in qual caso è l'utente a scegliere quale versione prevale

Consigli per la lettura



Git Base

<https://git-scm.com/book/it/v1/Per-Iniziare-Basi-di-Git>



Git - La guida tascabile

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

