

Formazione specialistica per sviluppatore front-end





Hello!

Alessandro Pasqualini

Full-stack developer, appassionato di programmazione e sviluppo software in generale, amo le serie tv e i gatti

Versionamento del software: GIT



Cos'è il versionamento

I sistemi di controllo di versione sono strumenti software che permettono ad un team di gestire modifiche al codice durante un periodo di tempo (tipicamente la vita dell'intero progetto).

Questi sistemi tengono traccia di ogni modifica apportata al codice in uno speciale database.

Se viene commesso un errore, i developer possono "riportare indietro l'orologio" e comparare vecchie versioni del codice senza causare ulteriori problemi.



Versionamento NON è backup

L'idea fondamentale del controllo di versione è mantenere diverse revisioni della stessa unità di informazione (es. codice).

L'idea di backup è mantenere al sicuro l'ultima versione dell'informazione (es. codice). Le vecchie versioni possono essere sovrascritte o eliminate.



Repository

Il concetto di repository è centrale e fondamentale quando si parla di VCS.

Un repository è simile ad una cartella del pc dove sono contenuti tutti i file del progetto e tutte le loro revisioni.

Una modifica ad un file in terminologia VCS è chiamata revisione.

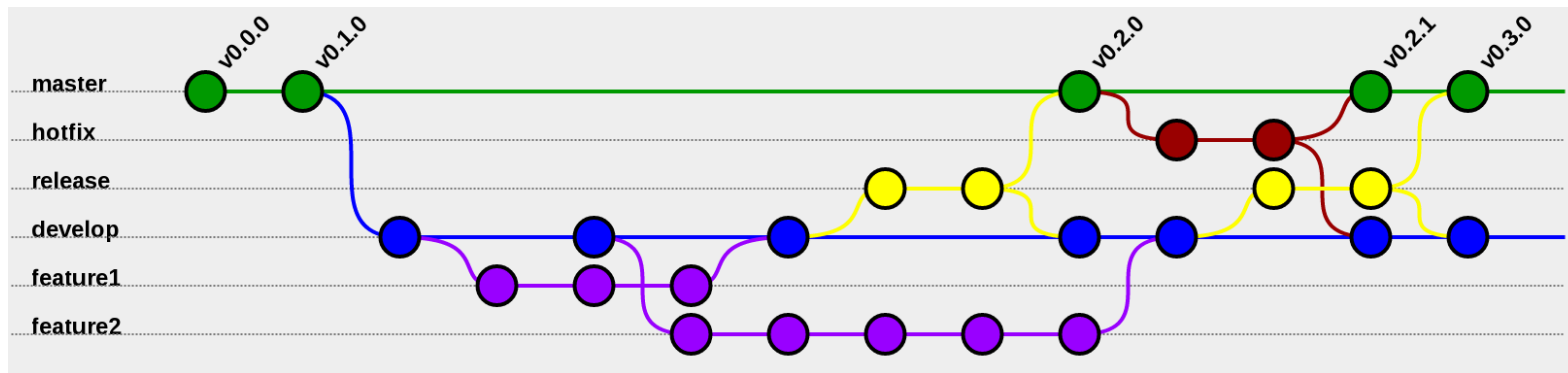


Commit

Quando si apportano delle modifiche a dei file del progetto è necessario salvarle e darci una descrizione.

Nella terminologia Git, salvare le modifiche equivale a creare un **commit** (dall'inglese commettere).

Quindi un commit è un insieme di modifiche apportate ad uno o più file, insieme ad un messaggio testuale che descrive le modifiche fatte. (Il messaggio è obbligatorio).





"Clonare" un repository

Quando si lavora con un repository è necessario "scaricare" una copia in locale in modo da poterci lavorare.

Fare un "clone" di un repository equivale a scaricare una copia locale dello stesso, mantenendo un collegamento alla "sorgente" dalla quale è stato scaricato.

Il comando da eseguire è:

```
git clone url_del_repository
```

Git creerà una cartella chiamata con lo stesso nome del repository in cui scaricherà tutto il suo contenuto (e tutte le revisioni dei file).



Aggiungere un file al repository

Per aggiungere un file al repository:

```
git add nome_del_file
```

Il file è semplicemente stato aggiunto, ma non è ancora parte di un commit, quindi **le sue modifiche non sono tracciate**.

Aggiungere un file significa dire a Git che quel file verrà inserito all'interno di un commit a breve.



Vedere lo stato del repository

Per vedere lo stato del repository e quindi conoscere quali file sono stati modificati, aggiunti, cancellati, etc, ma non committati:

```
git status
```

Git ci mostra lo stato corrente e quindi solo quello che è stato modificato dal commit precedente (le modifiche precedenti).



Creare il nostro primo commit

Una volta aggiunti i file modificati è necessario salvare le modifiche creando un commit ("save point").

```
git commit -m "Messaggio descrittivo"
```

Il messaggio deve essere una descrizione, seppur concisa, delle modifiche contenute nel commit (le modifiche fatte ai file che inseriamo nel commit).

Sono PROIBITI messaggi inutili o "tanto perché è obbligatorio il messaggio"



Caricare il commit su Github

Una volta che abbiamo creato il nostro commit dobbiamo caricarlo online, ad esempio su Github in modo che le altre persone possano scaricarlo.

```
git push
```

Se non abbiamo commit da caricare Git ci dice che non ha apportato modifiche al repository online.



Git step by step

Primo passo obbligatorio è clonare il repository, altrimenti non abbiamo il codice su cui lavorare!

Successivamente questi step si ripetono a ciclo:

1. Modificare, creare, eliminare file (quindi fare il lavoro sul codice)
2. Aggiungere a Git i file modificati da includere nel commit
3. Creare il commit insieme al messaggio di descrizione
4. Fare il push del commit
5. Tornare al punto (1) e ricominciare il ciclo

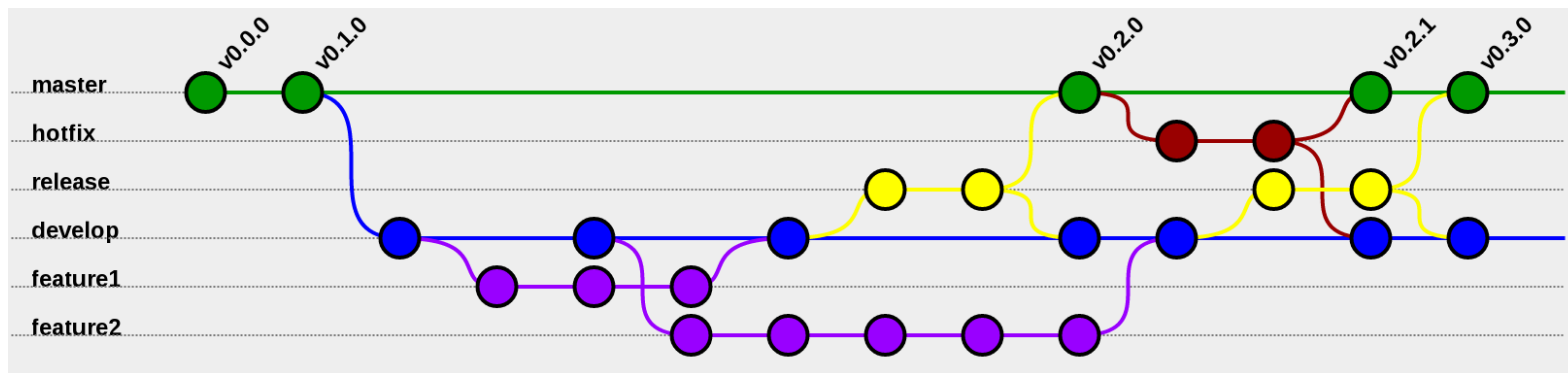


Branch

Dall'inglese significa ramificazione.

Durante lo sviluppo è possibile che sia necessario intraprendere diverse strade: pensate all'implementazione di una nuova funzionalità del nostro software.

In Git un branch è una ramificazione dello sviluppo, una strada alternativa di sviluppo.





Branch

Nei repository esiste sempre un branch "speciale" chiamato **master** che rappresenta la linea di sviluppo principale.

Tutte i branch (ramificazioni del codice) possono successivamente essere inclusi nel branch master (quello principale), abbandonati e quindi eliminati dal repository, etc.

Nei casi più semplici si usa solamente il branch master.



Come creare un branch

Git ci mette a disposizione un semplice comando per creare un nuovo branch:

```
git checkout -b nome_del_nuovo_branch
```

Un branch è una ramificazione del codice e quindi all'interno di questo nuovo "ramo" troviamo tutte le modifiche (commit) effettuati fino al momento in cui è stato creato il branch.

Successivamente i branch seguiranno "strade" differenti e quindi avranno commit differenti (avranno una storia divergente).



Perchè usare diversi branch?

Usare branch differenti ha diversi vantaggi anche se introduce alcune complessità.

E' comune contenere nel branch **master** il **codice attualmente in produzione**, mentre spostare su un branch differente (a volte chiamato develop) lo sviluppo normale. Quando il codice contenuto nel secondo branch viene ritenuto abbastanza maturo da essere messo in produzione, i due branch vengono uniti (merge).

Questo permette, in caso di bug in produzione (gli unici che hanno priorità assoluta), di poter essere risolti conoscendo esattamente quale codice è installato nel server e che ha causato il problema.



Perchè usare diversi branch?

Un'altra "metodologia" molto comune è utilizzare un branch differente per ogni membro del team.

Questo permette che le modifiche (magari non complete) compiute da un membro non "intralcino" il lavoro svolto dal resto del team. Solo quando le modifiche sono complete (e testate!) possono essere unite al codice in master.

Il tutto funziona correttamente (e senza problemi) se i vari membri del team lavorano su funzionalità che "non si intersecano" tra di loro.

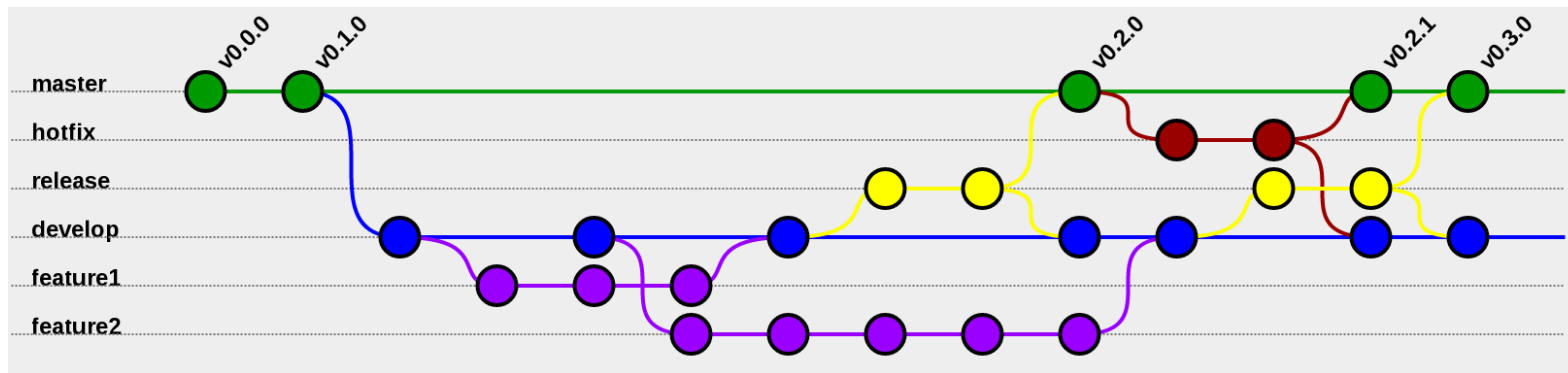


Perchè usare diversi branch?

I branch sono ramificazioni del codice e le loro storie divergono (commit diversi) e quindi **le modifiche effettuate su un branch non incidono sugli altri branch.**

Questo permette di poter eseguire delle "prove" di sviluppo di un branch separato.

Se queste modifiche vengono alla fine ritenute "buone" allora possono essere unite al branch principale, altrimenti eliminando il branch vengono eliminate anche le modifiche ripristinando il codice originale.





Come vedere tutti i branch

Per vedere quali branch sono presenti in un repository è possibile eseguire il comando:

```
git branch
```

C'è un però: questo comando mostra solamente i branch "locali", ovvero i branch presenti sul pc. Per vedere tutti i branch, compresi quelli online:

```
git branch -a
```



Come muoversi tra branch

Quando siamo su un branch possiamo entrare in un altro branch semplicemente eseguendo il comando

```
git checkout nome_del_branch
```

E' **quasi** lo stesso comando che eseguiamo per creare un nuovo branch: l'unica differenza è l'assenza dell'attributo `-b`



Come eliminare un branch

Per eliminare un branch locale è possibile eseguire il comando:

```
git branch -d nome_del_branch
```

Questo comando viene eseguito da git solamente se il branch è stato unito (merge) in un altro branch. Per forzare git e obbligarlo ad eliminare un branch in ogni caso (USARE CON CAUZIONE):

```
git branch -D nome_del_branch
```



Come eliminare un branch

Il comando

```
git branch -d nome_del_branch
```

elimina solamente il branch locale ma non elimina il branch online (su Github per esempio). Per eliminare il branch online dobbiamo usare una sintassi leggermente differente:

```
git push --delete origin nome_del_branch
```



Come eliminare un branch

Eliminare un branch è un'operazione **irreversibile**: una volta eliminato non c'è possibilità di recuperarlo.

Se il branch viene eliminato solamente localmente è possibile scaricarlo dal repository online (Github) ma se viene eliminato anche online non c'è possibilità di recupero.

Prima di eliminare un branch verificate bene quello che state facendo!



Come caricare un branch online

Quando creiamo un branch in locale esso non viene automaticamente caricato online (Github). Per caricare un branch online dobbiamo eseguire:

```
git push --set-upstream-to origin/nome_del_branch
```



Come caricare un branch online

Dobbiamo ricordarci questo comando "complicato"? No git ci aiuta e ci suggerisce il comando corretto se eseguiamo solamente:

```
git push
```

Successivamente possiamo caricare online i nostri commit come al solito eseguendo solamente il comando:

```
git push
```



Come unire due branch

Quando abbiamo completato il lavoro su un branch dobbiamo unire le modifiche nel branch principale (di solito **master**)

Nella terminologia di git questa operazione si chiama **merge**.

Eseguire un merge è un operazione che può portare ad un conflitto: stessi file e stesse righe modificate nel branch destinazione (master) e nel branch da unire (il nostro branch).



Come unire due branch

Prima di unire due branch dobbiamo spostarci sul branch di destinazione del merge (quello in cui vogliamo unire le modifiche dell'altro branch):

```
git checkout nome_del_branch_destinazione
```

Successivamente possiamo eseguire il merge dell'altro branch dentro quello in cui siamo in questo momento (quello destinazione):

```
git merge nome_del_branch_con_le_modifiche
```



Se si presentano conflitti

Se nell'operazione di merge si presenta un conflitto git ci avverte automaticamente e ci indica quali file sono "andati in conflitto".

```
MacBook-Pro-di-Alessandro-2:cartella senza titolo alessandro$ git merge test
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
MacBook-Pro-di-Alessandro-2:cartella senza titolo alessandro$ █
```

```
<<<<<<< HEAD
modifica sul branch master
=====
modifica effettuata sul branch 'test'
>>>>>>> test
```

Il contenuto del file in conflitto viene modificato come quello in parte in modo da mostrarci le righe che hanno generato il conflitto.



Come risolvere un conflitto

```
<<<<<< HEAD
modifica sul branch master
=====
modifica effettuata sul branch 'test'
>>>>>> test
```

Il contenuto del file in conflitto viene modificato come quello in parte in modo da mostrarci le righe che hanno generato il conflitto.

Per risolvere un conflitto è sufficiente decidere quale riga tenere eliminando la riga (e le righe <<<< e >>>> aggiunte da git), aggiungere il file e creare il commit normalmente.

VSCode ci aiuta a risolvere i conflitti in modo "visuale".



Thanks!

Any questions?

Per qualsiasi domanda contattatemi:
alessandro.pasqualini.1105@gmail.com