Introduzione a Git - Usare Git come strumento di collaborazione online

È passato quasi un anno da quando abbiamo iniziato il nostro percorso introduttivo a Git: per cause di forza maggiore non sono riuscito a concludere questa piccola guida, ma sono tornato per portare a termine quanto iniziato.

Ricapitolando: nella prima lezione vi ho spiegato cos'è Git, come installarlo e come configurarlo all'interno del vostro sistema, mentre nella seconda lezione vi ho introdotto alle principali funzionalità di questo version control system.

In questa lezione imparerete a usare Git come strumento di collaborazione online: come lavorare su più rami, come pubblicare il vostro codice, come scaricare gli aggiornamenti e così via.

I rami

In Git, i rami (branch) sono una parte fondamentale del processo di sviluppo. Quando lo sviluppatore vuole aggiungere una nuova feature o correggere un bug crea un nuovo ramo, ovvero una nuova linea di sviluppo indipendente dalle altre. In questo modo i cambiamenti vengono incapsulati all'interno di questo ramo e il nuovo codice non va a rendere instabile il nostro progetto.

Sembra difficile ma non lo è, nella prossima sezione vedremo come funziona attraverso un esempio pratico.

Utilizzo

Immaginiamo di lavorare a un tema WordPress da inserire su ThemeForest. Avremo un ramo, il principale, chiamato master che contiene il codice che verrà messo in produzione. Ipotizziamo ora di voler aggiungere una nuova feature: il portfolio.

Per organizzare il lavoro creo un nuovo ramo chiamato *feature/portfolio*.

A questo punto è come se Git copiasse tutto il contenuto di master all'interno di una nuova cartella dandovi modo di modificare il codice senza andare a compromettere l'intero progetto.

Attraverso il comando *git branch* possiamo ora controllare che il nostro ramo sia presente: Git ci restituirà la lista di tutti i rami esistenti evidenziando il ramo corrente (quello in cui stiamo sviluppando). Nel nostro caso il risultato sarà il seguente:

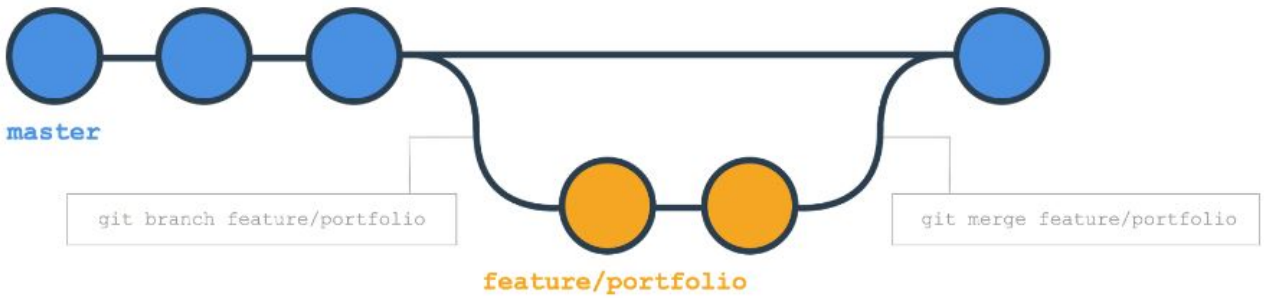
```
$ git branch feature/portfolio
$ git branch
* master
  feature/portfolio
$
```

Il ramo corrente è ancora *master*. Per iniziare a sviluppare all'interno del ramo che abbiamo appena creato dobbiamo eseguire:

Ora possiamo iniziare lo sviluppo della nuova feature utilizzando una linea di sviluppo indipendente da master. Completata e testata la nuova feature, ci basta unire (merge) il ramo *feature/portfolio* al ramo *master* e il gioco è fatto. Per farlo, prima di tutto salviamo tutte le nostre modifiche all'interno del ramo *feature/portfolio* attraverso un semplice *git commit* (vedi lezione 2), successivamente torniamo al branch *master* (*git checkout master*) e infine uniamo la nostra feature al nostro ramo di sviluppo principale attraverso il comando:

Questo ci permette di prendere un ramo di sviluppo creato con il comando *git branch* e integrarlo all'interno di un altro ramo.

Per semplificare questo discorso, ecco uno schema di come funziona il processo appena descritto:



Ora non ci resta che imparare come condividere il codice con i nostri collaboratori.

Sincronizzazione

In Git ogni sviluppatore, all'interno del proprio computer, possiede la copia completa della repository a cui sta lavorando. Gli sviluppatori quindi devono avere un modo per condividere il loro codice così che tutti i loro collaboratori possano avere accesso alle varie modifiche.

I comandi presentati successivamente vi danno la possibilità di gestire le connessioni con altre repository, pubblicare il vostro codice e caricare le modifiche eseguite dai vostri collaboratori.

git remote

Per sincronizzare il codice con gli altri dobbiamo necessariamente connettere la nostra repository a un server remoto che funga da "centro operativo". Il comando *git remote* ci dà la possibilità di creare, visualizzare o cancellare le connessioni a questi server remoti.

Per vedere le connessioni attive all'interno della nostra repository ci basta usare:

che ci restituirà la lista dei server remoti a cui siamo connessi. Solitamente se la repository è stata clonata, vedremo di default un server chiamato *origin* (che è il nome predefinito che Git associa al server da cui cloni).

Se volessimo aggiungere una nuova connessione a un server remoto, non dovremmo far altro che utilizzare questo comando:

Questa funzionalità vi risulterà utile quando inserirete il controllo di versione all'interno di un vostro progetto e vorrete spedire tutto il codice e lo storico a un server remoto come GitHub o BitBucket.

git pull

Immaginiamo che un nostro collaboratore abbia completato una feature e abbia caricato questa feature nella nostra repository remota. Come facciamo a caricare le sue modifiche all'interno della repository su cui stiamo lavorando in locale?

In questo caso utilizzeremo il comando:

Questa funzionalità va a prendere il codice aggiornato nel server remoto, lo carica in locale ed esegue un *merge* automatico. Il merge, tra l'altro, è uno strumento molto potente, infatti poniamo il caso che entrambi abbiate modificato la stessa porzione di codice: Git non sa qual è quella giusta, ma ve lo segnala attraverso un apposito messaggio in modo che voi possiate andare a scegliere cosa mantenere e cosa cancellare.

git push

Ora veniamo all'ultima funzionalità che vi spiegherò oggi: il push. Questo comando vi permette di trasferire le vostre modifiche dalla vostra repository locale alla repository remota, dando quindi la possibilità a tutti i vostri collaboratori di accedere alle vostre modifiche. Per farlo non vi basta che digitare:

Fatto questo Git procederà con l'upload di tutti i cambiamenti che avete effettuato. Il processo può richiedere un tempo variabile a seconda del peso delle vostre modifiche.

Mi raccomando, prestate attenzione quando usufruite di questa funzionalità in quanto, se avete commesso un errore durante la programmazione del codice, questo errore verrà diffuso a tutti i vostri collaboratori. Inoltre ricordatevi sempre di eseguire un *git pull* prima del *push*, altrimenti le modifiche eseguite dai vostri collaboratori andranno perse.

Conclusione

Con questa lezione avete imparato due cose fondamentali, l'utilizzo dei branch e la sincronizzazione del codice tra i vostri collaboratori.

Per ora vi ho spiegato l'utilizzo di queste risorse a livello teorico ma, nella prossima lezione, andremo ad utilizzare quanto imparato finora attraverso un caso studio dove simuleremo l'implementazione e l'utilizzo di un sistema di controllo versione all'interno di un nostro ipotetico progetto.

INTRODUZIONE A GIT: INDICE LEZIONI

- Cos'è e come installarlo
- Quali sono le funzioni principali di Git
- Usare Git come strumento di collaborazione online