Introduzione a git

Utilizzo di base

Davide Bianchi

26 marzo 2018

Indice

1 Introduzione

Questa guida è dedicata a tutti coloro che vogliono avventurarsi nel mondo di Git. Git è un software di controllo versione distribuito utilizzabile da interfaccia a riga di comando, creato da Linus Torvalds nel 2005.

Consente di gestire il codice tra più sviluppatori senza che questi debbano ogni volta passarsi il codice a mano, ma lasciando questo gravoso compito al tool.

In questa guida ne illustreremo l'utilizzo di base, per consentire a chiunque la legga di imparare abbastanza velocemente il suo utilizzo, focalizzandoci sull'uso di una CLI (anche se sono disponibili numerosi tool che fanno uso di una GUI).

2 Setup dell'ambiente

2.1 Installazione

L'installazione di git è semplice e immediata, infatti, se siete utenti Linux basterà installarlo usando il gestore di pacchetti `apt`: dal terminale basta digitare il misterioso comando:

```
sudo apt install git
```

Per gli utenti Windows e Mac basterà scaricarlo dal sito ufficiale https://git-scm.com/, selezionando il proprio sistema operativo.

2.2 Configurazione

Per rilevare chi inserisce un dato commit, git utilizza uno username, che va fornito prima di cominciare al tool con il comando:

```
git config --global user.name "<username>"
```

È inoltre buona norma specificare l'indirizzo mail dal quale vengono eseguiti i commit, con il comando

```
git config --global user.email "<mail>"
```

3 Utilizzo di base

3.1 Inizializzazione di un progetto

Creazione di un nuovo progetto. Assumiamo che voi ora abbiate un account su GitHub, e vogliate creare un nuovo progetto da rendere disponibile open source (come ognuno dovrebbe fare).

Una volta creata la repository online tramite il sito, sarà necessario collegare la repository online (che da ora in poi si chiamerà *remote*) alla cartella in locale dove il codice sorgente si troverà.

Per fare ciò inizializziamo un repository git locale con il comando

```
git init
```

che genererà una cartella `.git` nella cartella con il codice sorgente. Questa conterrà informazioni relative ai branch e ai commit eseguiti.

Per collegare il remote alla repository locale usiamo il comando

```
git remote add origin <url>
```

che aggiungerà un remote etichettato come `origin` alla repo locale. Al posto dell'url è possibile usare una chiave ssh, con qualche piccolo vantaggio (https://help.github.com/articles/connecting-to-github-with-ssh/).

La fase di inizializzazione termina con l'aggiunta di un file `.gitignore`, che conterrà le informazioni da ignorare quando si eseguiranno dei commit (in dettaglio più avanti).

Partecipazione ad un progetto già avviato. Per partecipare ad un progetto già avviato non occorre inizializzare nulla, git infatti mette a disposizione un comando che *clona* il repository remote in locale, con tutto integrato. Il comando è:

```
git clone <url>
```

3.2 Aggiornamento del repository

Se il repository è mantenuto da più sviluppatori, è possibile che questi abbiano committato e pushato sul remote, e aggiunto nuove caratteristiche al software. Per aggiornare il repository locale con le modifiche pushate da altri, è necessario usare i comandi `git pull` e `git fetch`.

Il comando `git fetch` scarica le informazioni dal remote e **non esegue nessun merge**. Il comando `git pull` invece scarica le informazioni e tenta di eseguire il merge (ovviamente qui si possono generare dei conflitti, spiegati più avanti). Se si aggiorna il progetto con `git fetch`, il merge va eseguito manualmente con `git merge`.

Fatto questo passo, si può procedere alla modifica o aggiunta del codice.

3.3 Il meccanismo di commit

Il commit è l'unità di base di git, sostanzialmente un malloppo di codice che va integrato nel progetto principale. Quando si scrive del codice, git tiene traccia delle modifiche fatte, visualizzabili tramite il comando

```
git status
```

I file che vanno committati vanno prima raggruppati con il comando

```
git add <file1> ... <filen>
git add -A
```

L'opzione `-A`, da usare in alternativa ad un elenco di file, prepara per il commit **tutte** le modifiche eseguite. Ora viene il bello: i commit in locale vanno eseguiti con il comando

```
git commit -m "<messaggio>"
```

dove il messaggio è un sommario delle modifiche apportate. Un commit contiene tutte le modifiche che sono state raggruppate con il precedente `git add`. È buona prassi scrivere messaggi sintetici ma esaustivi di quanto si è svolto.

Per mandare (in gergo nerd *pushare*) i commit al remote si usa il comando

```
git push <etichetta> <branch>
```

L'etichetta rappresenta il nome con il quale il remote è salvato in locale, mentre `branch` è il nome del branch sul quale si vuole pushare (`origin` nel caso precedente)[1].

È possibile saltare la fase di staging dei file modificati quando si intende pushare tutte le modifiche, usando il comando

```
git commit -a -m "<messaggio>"
```

[1] Per evitare confusione, useremo sempre l'etichetta `origin`

3.4 Cronologia dei commit

È possibile visualizzare la cronologia dei commit usando il comando `git log`, con due opzioni che lo possono rendere particolarmente utile, quali:

- `-p` per avere un riepilogo delle modifiche apportate;
- `-n` con *n* numero positivo per limitare l'output del log alle ultime *n* entry.

4 Branching

4.1 Creazione e cancellazione

Un *branch* è un ramo del progetto indipendente da quello principale (*master*), sul quale è possibile committare un codice qualsiasi. I branch tornano utili quando, ad esempio, si sta scrivendo una sezione critica del software e si vuole svilupparla senza rischiare di danneggiare il codice principale sul branch master. Di conseguenza si distinguono due casi:

- il codice sviluppato nel branch è valido e può essere eseguito il *merge* con il master;
- il codice sviluppato contiene errori o per qualche motivo non serve più. In tal caso sarà sufficiente cancellare il branch e il codice principale sul master non subirà nessuna modifica.

Tornano comodi i seguenti comandi:

Creare un branch	`git branch <nome-branch>`
Ottenere una lista dei branch	`git branch`
Cancellare un branch	`git branch -d <nome-branch>`

4.2 Committare su branch

Per eseguire un commit su un branch, è prima necessario spostarsi dal ramo[2] master al branch sul quale si vuole committare, con il comando:

```
git checkout <nome-branch>
```

Una volta eseguito il checkout, è sufficiente ridare i comandi per committare:

```
git add -A
git commit -m "<messaggio>"
git push origin <branch>
```

sostituendo a `branch` il nome del branch sul quale si è committato. Se, una volta committato, si vuole tornare sul master e fare altre modifiche, basterà eseguire il comando `git checkout master`.

4.3 Eseguire un merge tra due branch

Un *merge* è molto semplicemente una fusione tra due rami esistenti. Il codice risultato dipende dal verso del merge.

In generale, per eseguire un merge è necessario utilizzare il comando:

```
git merge <branch-base>
```

Onde evitare disastrose conseguenze per i progetti dei lettori di questa dispensa, si specifica quanto segue: il comando sopra esegue il merge del branch `branch-base` nel branch corrente (quello dove ci si trova al momento attuale). Notare che il merge viene eseguito in locale, per averlo anche su remote basta pushare con i comandi dati in precendenza.

[2] Si usano i termini *ramo* e *branch* indistintamente.

4.4 Conflitti durante il merge

Durante il merge di due branch, potrebbe capitare di imbattersi in un conflitto, con un messaggio che recita all'incirca così:

```
Auto-merging styleguide.md
CONFLICT (content): Merge conflict in styleguide.md
Automatic merge failed; fix conflicts and then commit the result
```

Il problema dei conflitti sorge quando entrambi i codici sono stati modificati. In questi casi git non può sapere quale delle due versioni sia quella da tenere. Nel file sorgente le righe coinvolte da un conflitto sono indicate come segue:

```
If you have questions, please
<<<<<<< HEAD
open an issue
=======
ask your question in IRC.
>>>>>>> <nome-branch>
```

Dove da `<<<<<<< HEAD` a `=======` si ha la versione del codice che è contenuta nel branch corrente, mentre nella seconda parte (fino a `>>>>>>> <nome-branch>`) quella dell'altro branch, con indicato il suo nome.

Per risolvere il conflitto è necessario rimuovere i marker del conflitto e decidere quale delle due soluzioni tenere. Una volta apportate le modifiche si può eseguire il commit con la nuova versione.

5 Annullare azioni

Con git si possono annullare alcune azioni (quasi tutto in realtà), quali disfare dei commit, modificare dei commit esistenti, ecc.

5.1 Modificare dei commit

La modifica dei commit è un procedimento che si svolge spesso, ad esempio quando ci si dimentica di fare una modifica ad un file ed è appena stato fatto un commit. Supponiamo di aver dato i seguenti comandi:

```
git add -A
git commit -m "Added code"
```

A questo punto ci accorgiamo di non aver aggiunto un nuovo file di cui il progetto ha bisogno. Dopo aver aggiunto il file quindi si danno i seguenti comandi:

```
git add forgotten_file
git commit --amend
```

Il risultato sarà un unico commit, ma quello con il file dimenticato andrà a rimpiazzare quello precedente.

5.2 Stashing delle modifiche

Spesso ci si trova nella situazione di dover fare una pull dopo aver svolto delle modifiche, a volte anche corpose. Se si tenta di fare una pull con delle modifiche non committate, si otterrà un seguente messaggio di errore in cui git si lamenta del fatto che non sa come gestire le modifiche

locali. In tal caso è possibile salvare le modifiche apportate, eseguire la pull, e poi riapplicare le modifiche (potrebbe causare dei conflitti).

Con il comando `git stash`, le modifiche attuali vengono salvate su uno stack e rimosse dal codice sorgente. Successivamente è possibile eseguire una pull come spiegato in precedenza, per poi riapplicare le modifiche togliendole dalla cima dello stack con il comando `git stash apply`.

Complessivamente, la sequenza di comandi necessaria ad ovviare il problema originale è la seguente:

```
git stash
git pull [alias] [branch]
git stash apply
```

6 Appendice: principali comandi

6.1 Configurazione della repo

Inizializzazione del repository	`git init`
Includere un remote	`git remote add [label] [url]`
Download di un progetto	`git clone [url]`

6.2 Comandi per i branch

Creazione di un branch	`git branch [name]`
Switch di branch	`git checkout [name]`
Mostrare la lista dei branch	`git branch`
Cancellare un branch	`git branch -d [name]`
Merge di `[branch]` nel branch corrente	`git merge [branch]`

6.3 Committare le modifiche

Preparare i file per il commit	`git add [files]`
Preparare tutti i file per il commit	`git add -A`
Elencare le modifiche da committare	`git status`
Rimuovere i file dalla *stage-area*	`git reset [files]`
Committare i file	`git commit -m "[message]"`
Disfare le modifiche dopo `[commit]` mantenendo le modifiche	`git reset [commit]`
Disfare tutto fino a `[commit]`	`git reset --hard [commit]`

6.4 Aggiornare la repo locale

Scaricare la cronologia dal remote	`git fetch [alias] [branch]`
Scaricare la cronologia e incorporare le modifiche	`git pull [alias] [branch]`
Pushare i file sul remote	`git push [alias] [branch]`

6.5 Navigazione della cronologia

Elencare la cronologia	`git log`
Mostrare le modifiche di un commit	`git show [commit]`
Mostrare le modifiche tra branch	`git diff [branch1] [branch2]`