# Sistema di Controllo Versione Git

## Ripasso

Un sistema di controllo versione è un sistema che permette di tenere traccia delle successive versioni di un progetto (ad esempio di un progetto di sviluppo di un software o un testo complesso).

I sistemi di controllo versione, come ad esempio Git e Mercurial, consentono di collaborare in rete a livello globale (cioè da tutto il mondo, purché si disponga di una connessione a Internet).

Git è probabilmente il sistema di controllo versione attualmente più utilizzato. É usato, ad esempio, per lo sviluppo del kernel Linux e del brower Chrome.

GitHub è il servizio web più utilizzato tra quelli che mettono a disposizione il sistema di controllo versione Git.

Alcuni concetti sono particolarmente importanti in Git. Il ***repository*** (letteralmente deposito) è, come suggerisce il nome, un luogo (generalmente una directory sul disco) che Git utilizza per conservare i file di progetto e la ***history*** (cioè la storia) del repository stesso. Eventuali file che non si vogliono tracciare ma che per qualche motivo è opportuno che risiedano nella cartella di progetto possono essere ignorati da Git semplicemente elencandoli nel file .gitignore, la cui creazione è un'operazione a carico dell'utente.
Per history si intende l'insieme dei ***commit***, cioè degli stati del repository (e quindi del progetto) che si è scelto di tracciare. La history è gestita dal sistema Git tramite un insieme di file che si trovano nella directory nascosta .git, che si trova a sua volta nella directory di progetto.
Attraverso questi file nascosti Git definisce una struttura a lista concatenata nota come ***branch master*** (ramo di sviluppo principale) dove la ***head*** (testa) corrisponde all'ultimo commit, cioè al più recente.

Un progetto Git può essere creato localmente sul proprio computer e poi registrato su un servizio web come GitHub per la collaborazione in rete oppure creato direttamente attraverso il servizio web, come abbiamo fatto nella terzultima lezione del corso col repository hello-world.

Nel primo caso, l'utente crea la propria ***copia locale*** del progetto. Per creare anche una copia centralizzata, ad esempio del repository hello-world su un server di GitHub, sarà necessario dare i seguenti comandi:

```
git remote add origin https://github.com/teledon/test.git
git push -u origin master
```

Si noti l'uso dei nomi origin e master nei precedenti comandi.

Nel secondo caso il servizio web crea direttamente la copia centralizzata, nota come ***origin*** su uno dei propri server e, per poter lavorare sul progetto, sarà necessario ottenere una copia locale del repository. Questa operazione si chiama ***clone*** e si effettuata tramite il comando

```
git clone https://github.com/teledon/hello-world.git
```

Verrà creata una directory locale di nome hello-world con i file di progetto e la cartella nascosta .git.

Chi crea un progetto Git ne è automaticamente l'***owner***, cioè il proprietario e ha tutti i diritti sul progetto. In GitHub è possibile aggiungere altri ***contributors***, cioè contributori o sviluppatori, ai quali si possono assegnare vari livelli di privilegi. Volendo, l'owner può condividere o cedere la proprietà del repository ad altri contributors. In realtà, la possibilità di contribuire a un progetto, se questo è pubblico, non è limitato al solo gruppo dei contributori ma chiunque, da qualsiasi parte del mondo, può richiedere ai contributori ufficiali di inserire nel progetto le proprie modifiche, come vedremo più avanti.

In un tipico scenario di lavoro Git esiste una copia centralizzata (origin) e una o più copia locale del progetto. Ogni contributore lavora sulla propria copia locale. Prima di iniziare una modifica al progetto, è buona norma che il contributore importi nella propria copia locale eventuali modifiche che altri utenti hanno effettuato e applicato al repository centralizzato. Questa operazione si chiama ***pull***. Quando si effettua un'operazione di pull, si possono presentare vari scenari:

- La copia centralizzata coincide con la copia locale. In tal caso non succede nulla.
- Le novità (commit di altri utenti) su origin *non sono in conflitto* con le modifiche locali (commit locali) e vengono importate automaticamente tramite un'operazione detta ***fast forward merge*** da Git.
- Le novità su origin *sono in conflitto* con le modifiche locali. In questo caso l'utente è invitato a ***risolvere i conflitti***. Una volta risolti i conflitti, l'utente deve registrare questo ulteriore lavoro con un ulteriore commit.

A questo punto l'utente effettua le modifiche che ritiene necessarie e le registra tramite uno o più commit sulla propria copia locale.
L'ultimo atto consiste nel condividere i nuovi commit presenti nella propria copia locale con il repository centrale. Questa operazione si chiama ***push***.

Riguardo a come effettuare le modifiche, questo dipende se si è contributori ufficiali del progetto o meno.

Pur non esistendo un obbligo in tal senso, i contributori ufficiali sono invitati a creare un ramo di sviluppo alternativo tramite un'operazione detta ***branch*** per evitare di sporcare il ramo master, che dovrebbe contenere solo commit consistenti (cioè stati di sviluppo del progetto perfettamente funzionanti). Una volta terminata la modifica del progetto sviluppata sul ramo alternativo lo sviluppatore effettua una ***pull request*** per incorporare le modifiche effettuate. Questo apre una fase di analisi che coinvolge tutti i contributori. In caso di valutazione positiva, le modifiche sono accettate e assimilate nel branch target, che è generalmente il master.

Se non si è contributori ufficiali, ma si vuole comunque contribuire allo sviluppo del progetto, o avviare uno sviluppo alternativo, si deve innanzitutto effettuare un'operazione detta ***fork*** del progetto originale. Questa operazione crea una copia di origin (cioè del repository centrale). Dal punto di vista del nuovo progetto ottenuto tramite fork, l'origin del progetto madre sarà riferito come ***upstream***. In altri termini, upstream coincide con l'origin del progetto da cui si è effettuato il fork.
A questo punto il nuovo progetto può avere vita propria e non interagire in alcun modo col progetto upstream. Questo è in realtà uno dei motivi d'essere dell'operazione fork. Tuttavia, da un progetto fork è possibile aprire una ***pull request verso upstream***. La gestione della pull request avviene allo stesso modo di una pull request effettuata per effettuare il merge di un branch da parte di un contributore di progetto.

Si noti, infine, che il lavoro di sviluppo vero e proprio da parte dei contributori/resto del mondo avviene prevalentemente sulle copie locali del repository. Qui, in locale, alcuni comandi sono particolarmente importanti. Tra questi, il comando

```
git branch
```

consente di listare i branch presenti nel repository, mentre il comando

```
git checkout
```

consente di cambiare branch di lavoro. Questo comando può essere usato anche per visitare (spostare il puntatore head su) un commit differente da quello corrrente. Il comando

```
git status
```

permette di ispezionare lo stato delle modifiche effettuate. In particolare, i file listati nell'area *not staged for commit* sono i file di progetto modificati rispetto all'ultimo commit ma non ancora incorporate nella history tramite un commit o segnalati per essere inserite in un commit. Il comando

```
git add
```

permette di selezionare i file modificati presenti nell'area *not staged for commit* che si desidera diventino parte del prossimo commit. Una volta aggiunti, tali file risulteranno nell'area *staged for commit*. Il comando

```
git commit
```

crea un commit e lo inserisce in testa al branch corrente. Infine, il  comando

```
git log --graph --all
```

lista la storia di tutti i commit effettuati considerando le varie diramazioni.