Il database sottostante Git

Arianna Masciolini

4 luglio 2018

Indice

1 Introduzione

Un *Version Control System* (VCS) è un sistema che tiene traccia delle modifiche apportate ad uno o più file in modo da garantire all'utente la possibilità di accedere alle versioni precedenti dei file suddetti in qualsiasi momento.

I primi sistemi di controllo di versione, locali, nacquero con l'idea di risolvere i problemi legati a quello che potrebbe essere definito versioning "manuale", consistente nel conservare più copie dei file d'interesse: la forte suscettibilità a errori e lo spreco di spazio su disco. Tra questi VCS locali, RCS ha goduto ha lungo di grande popolarità: esso salva su disco, in un particolare formato, una serie di *patch*, ossia le differenze tra una versione e l'altra dei file, in modo tale da poter ricostruire lo stato in cui era ognuno di essi in qualsiasi momento, applicandovi una dopo l'altra le varie patch.

Successivamente, ci si pose il problema di permettere a più persone di collaborare a distanza. Per risolverlo nacquero i sistemi centralizzati di controllo di versione (CVCS), come CVS, Subversion, e Perforce. In questi sistemi, per il resto analoghi ad RCS e simili, tutte le versioni dei file controllati sono salvate su un unico server e rese così disponibili ai diversi utenti. Anche questo approccio presenta però problematiche importanti, dovute al fatto che il server centrale rappresenta un punto di vulnerabilità per l'intero sistema.

I DVCS (*Distributed VCS*), di cui Git, Mercurial, Bazaar e Darcs sono gli esempi più noti, risolvono questo problema: i membri del gruppo non si limitano a scaricare la più recente versione dei file, ma copiano l'intero repository, cosicchè ogni client costituisce un backup completo del progetto.

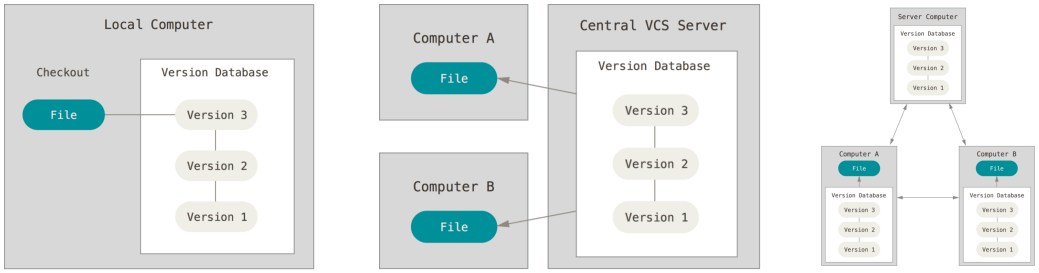


Figura 1: Confronto tra un VCS locale, un CVCS e un DVCS.

In particolare, Git -ad oggi il VCS più diffuso [2]- si differenzia tanto dagli altri sistemi distribuiti quanto dai loro predecessori per il modo in cui memorizza i dati: non come una serie di *patch* legate ai vari file, ma come una serie di "istantanee" di un filesystem in miniatura, accessibili tramite puntatori. L'obiettivo di questa relazione è, per l'appunto, descrivere l'approccio adottato da Git in tale ambito, in modo da comprendere quali ne siano

i vantaggi rispetto alle soluzioni adottate dai sistemi concorrenti. Nella sezione successiva verranno brevemente richiamati alcuni concetti di base riguardo il funzionamento di Git, utili per rendere più chiare e sintetiche le spiegazioni più strettamente legate al suo modello di branching e al sottostante database.

2 Nozioni di base su Git

Git, nato nel 2005 per favorire lo sviluppo del kernel Linux, è un sistema di controllo versione completamente distribuito che deve il suo successo al proprio modello di branching, che lo rende ottimale per la gestione di progetti open source, che hanno quasi sempre uno sviluppo non lineare.

Git come filesystem in miniatura Come si è accennato nella sezione precedente, Git adotta un approccio radicalmente differente rispetto a quello tradizionale, il cosiddetto controllo versione *delta-based*, basato su una lista di *patch*, ossia di modifiche, legate ai singoli file. Un repository Git è infatti una sorta di filesystem Unix-like semplificato, di cui viene fatta una sorta d'"istantanea" resa accessibile mediante un puntatore ogniqualvolta lo stato del progetto viene salvato.

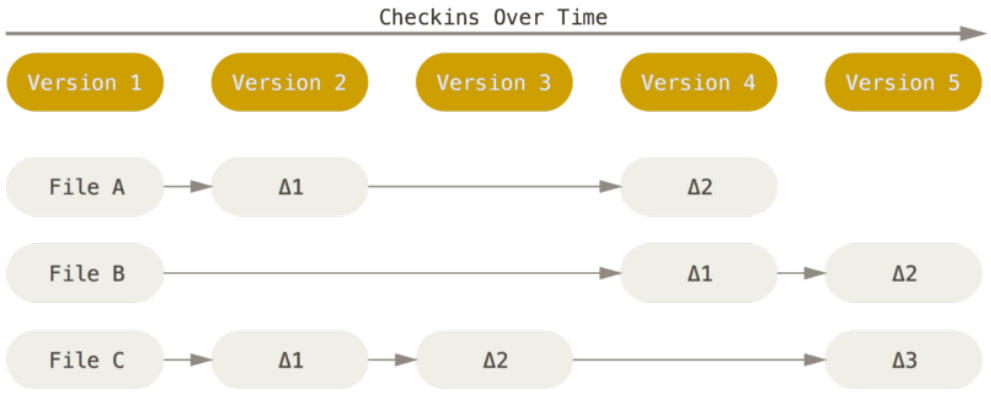
I tre stati Per la comprensione del funzionamento di Git è fondamentale conoscere i tre stati in cui un file all'interno di un repository può trovarsi:

- *Committed*: il file è salvato permanentemente nel database locale;
- *Modified*: sono state effettuate delle modifiche al file
- *Staged*

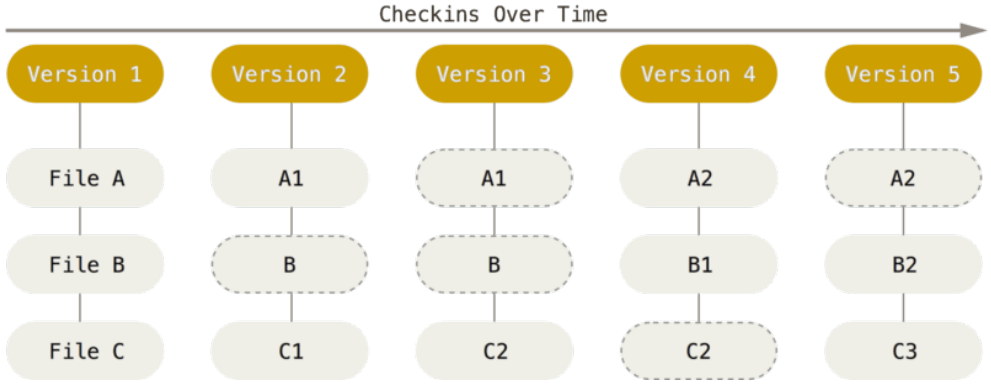
Checksumming Un'importante funzionalità di Git, parte integrante della sua filosofia, è quella di verificare qualsiasi file o directory, prima di archiviarlo, tramite una checksum che è poi utilizzata per riferirvisi. Ciò significa che è impossibile cambiarne il contenuto senza che Git ne sia a conoscenza, il che permette di rilevare qualsiasi perdita o corruzione di dati in transito. Il meccanismo impiegato per generare tale checksum è la funzione crittografica SHA-1, il cui output, basato sul contenuto di un file o sulla struttura interna di una cartella, è un numero esadecimale di 40 caratteri.

3 Il database di Git

Git può essere definito come *filesystem orientato ai contenuti*. Questo significa che il nucleo di Git è un database chiave-valore: all'inserimento di un oggetto in un repository, Git resti-



(a)



(b)

Figura 2: Confronto il modo in cui vengono salvati i dati in un VCS delta-based (a) e quanto avviene in Git (b).

tuirà la chiave con la quale esso è univocamente identificato. Per meglio rendersi conto del funzionamento di tale database, è opportuno inizializzare un repository

Riferimenti bibliografici

[1] Scott Chacon, Ben Straub. *Git Pro*, seconda edizione, 2014.

[2] `openhub.net`