Università degli Studi di Perugia

DIPARTIMENTO DI MATEMATICA ED INFORMATICA

Corso di Laurea in Informatica

Tesi di laurea

IMPLEMENTAZIONE DI UN TOOL PER IL SUPPORTO ALLA VALUTAZIONE AUTOMATICA DI PROGETTI SU GITHUB

Laureando:
Matteo Azzarelli
Matricola 244999

Relatore:
Prof. Francesco Santini

Anno Accademico 2017-2018

Un ringraziamento particolare ai miei genitori
e a Luisa che mi hanno supportato in questi anni.

Sommario

Contenuto del sommario

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

1 | GitHub Classroom

1.1 GitHub

GitHub è un servizio di hosting per il controllo delle versioni basato su *Git* (1.1.1). Esso, oltre al controllo delle versioni, fornisce anche altre funzionalità di collaborazione come il bug tracking, gestione delle attività e wiki per ogni progetto.

GitHub offre la possibilità di creare repositories sia private che pubbliche, quest'ultime spesso sono utilizzate per condividere progetti open-source, molto importante per la comunità scientifica.

Nato nel 2008 questo progetto prende subito piede, infatti in nemmeno un anno raggiunge 100.000 utenti iscritti. Dopo 10 anni dalla sua nascita arriva a 22 milioni di iscrizioni diventando uno standard e un requisito necessario per tutti i programmatori.

Un importantissima iniziativa lanciata da GitHub è il nuovo programma **GitHub Student Developer Pack** che concede gratuitamente agli studenti un insieme dei tools e servizi più popolari come ad esempio *awsEducate* ,*bitnami* o ancora *Microsoft Azure* e ovviamente *GitHub* offre repositories private illimitate. GitHub offre agli insegnanti il tool chiamato **GitHub Classroom** (1.1.2), che si propone come uno strumento per aiutare gli insegnanti ad educare le nuove generazioni di sviluppatori ad utilizzare gli strumenti richiesti dalle aziende e a padroneggiare il linguaggio necessario per immettersi nel mondo del lavoro.

1.1.1 Git

Una breve digressione sul *controllo di versione distribuito* ***Git*** [Chacon and Straub, 2018].

Esso tiene traccia dei cambiamenti dei file e coordina il lavoro su questi file tra un team di più persone. É utilizzato prevalentemente per la gestione di codice sorgente di progetti di software, ma esso può essere utilizzato per tenere traccia dei cambiamenti in qualsiasi set di files.

Pensato per essere un sistema di controllo distribuito esso è stato progettato per mantenere l'integrità dei files, di garantire un'alta velocità e anche di gestire dei flussi di lavoro non lineari.

Questo magnifico software è stato creato da *Linus Torvalds* nel 2005 insieme ad altri suoi colleghi per facilitare lo sviluppo del kernel di Linux .

Git si distingue da tutti gli altri software di versione per il modo in cui immagazzina i dati. A differenza dei precedenti VCS che salvano i dati come una lista delle modifiche (figura 1.1), Git salva i dati come delle istantanee (figura 1.2).

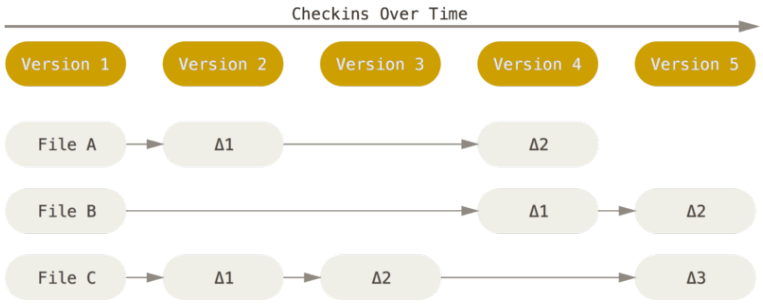

Figura 1.1: Gli altri VCS tendono ad immagazzinare i dati come cambiamenti alla versione base di ogni file.

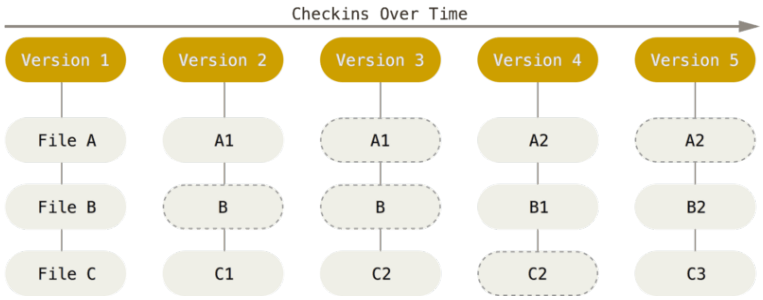

Figura 1.2: Git immagazzina i dati come snapshot del progetto nel tempo.

Questa peculiarità rende Git simile ad un mini *filesystem* con tutti i benefici di un gestore delle versioni. Questo modo di pensare i dati rende possibile la creazione di differenti Branch paralleli, in seguito verranno analizzati.

Ora esploriamo brevemente i controlli principali di git:

- Init
- Clone
- Commit
- Fretch
- Push
- Pull

Per gli altri comandi si rimanda alla guida ufficiale ProGit [Chacon and Straub, 2018].

Init

Se hai una directory di un progetto e vuoi iniziare il controllo di versione su di essa con Git dovrai utilizzare il comando:

```
$    git init
```

Questo comando crea una nuova sottodirectory chiamata .git, la quale contiene tutti i file necessari per la repository. A questo punto ancora non è tenuta traccia di alcun file. Se vuoi iniziare il controllo di versione su file già preesistenti dovrai aggiungere i file alla repository e poi fare un commit iniziale, ad esempio aggiungiamo tutti i file `.c` e un file di licenza e facciamo il commit con un messaggio:

```
$    git add *.c
$    git add LICENSE
$    git commit -m 'progetto versione iniziale'
```

A questo punto abbiamo una Git repository con i file tracciati e con un commit iniziale.

Clone

Molto spesso ci troviamo a lavorare con progetti già avviati, quindi in questo caso dovremmo utilizzare il comando `git clone <url>` per, appunto, clonare la repository desiderata. Esempio:

```
$    git clone https://github.com/MatteoAzzarelli/GitHub
ClassroomDownloader.git
```

In questo modo verrà creata una directory chiamata ClassroomDownloader, già inizializzata con la cartella .git, dove verranno scaricati tutti i file del progetto pronti per essere utilizzati.

Commit

Forse è il comando più utilizzato, in quanto permette di rendere effettive le modifiche ai file. Aggiungendo il flag `-m` abbiamo la possibilità di mettere un commento al commit, così da poter capire e far capire anche ai collaboratori quale modifiche sono state apportate. Esempio:

```
$    git commit -m 'Fix al Bug#'
```

Fetch

Per prendere i dati dal progetto remoto, ad esempio per aggiornare i file con le modifiche apportate da un collaboratore, è possibile utilizzare il seguente comando:

```
$    git fetch <remote>
```

dove <remote> è il nome del server remoto. si possono visualizzare i server remoti tramite il comando `git remote` , il server di default è chiamato `origin`.

Push

Nel momento in cui vuoi condividere con gli altri collaboratori le tue modifiche, dovrai caricarle nel server i tuoi file con il comando se `git push <remote> <branch>`. Esempio:

```
$    git push origin master
```

Il branch di default è chiamato master. Un branch è un ramo del progetto, questi spesso vengono utilizzati per aggiungere delle features al progetto, senza influire sulla versione base se la modifica non dovesse andare a buon fine. I rami possono terminare senza essere integrati nella versione base come si vede nel branch *Feature 2* dell'immagine 1.3, oppure possono essere riuniti ad un altro branch come ad esempio *Feature 1* o *Develop* dell'immagine 1.3.

Figura 1.3: Esempio di un workflow a branch

Pull

Il comando `git pull` generalmente recupera i dati dal server da cui originariamente sono stati clonati e tenta automaticamente di unirlo al codice su cui stai attualmente lavorando. Se non dovesse riuscire ad unire i file chiederà all'utente di risolvere i conflitti generati.

1.1.2 GitHub Classroom

2 | Lavori correlati

3 | Studio ed implementazione

Conclusioni

Bibliografia

S. Chacon and B. Straub. *Pro Git*. The expert's voice. Apress, 2018. URL `http://git-scm.com/book/en/v2`.