



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

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

Sommario

Contenuto del sommario

Indice

Introduzione	1
1 GitHub Classroom	3
1.1 GitHub	3
1.1.1 Git	3
1.1.2 GitHub Classroom	7
2 Lavori correlati	9
3 Studio ed implementazione	11
3.1 MOSS	11
Conclusioni	15
Bibliografia	17

Elenco delle figure

1.1	Salvataggio dati come cambiamenti	4
1.2	GIT salvataggio dati come snapshot	4
1.3	Esempio di un workflow a branch	6
3.1	Esempio di risultati di MOSS	14
3.2	Esempio pagina analisi sospetto plagio di MOSS	14

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. È 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).

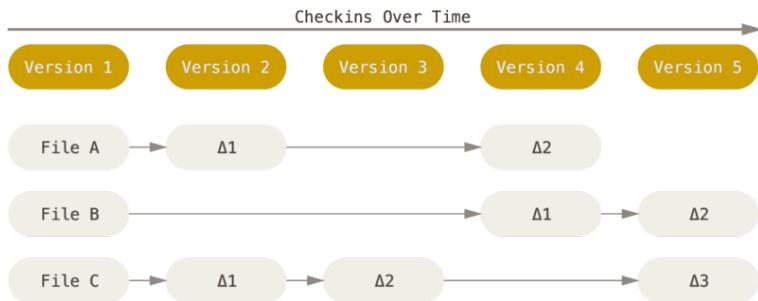


Figura 1.1: Gli altri VCS tendono ad immagazzinare i dati come cambiamenti alla versione base di ogni file.

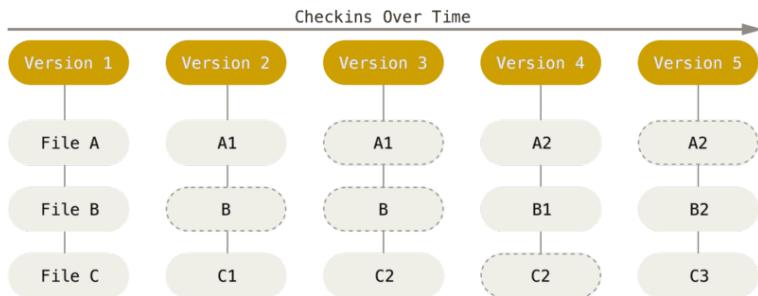


Figura 1.2: Git immagazzina i dati come snapshot del progetto nel tempo.

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 dovrà 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 dovrà 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, dovrà caricarle nel server i tuoi file con il comando `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` in figura 1.3, oppure possono essere riuniti ad un altro branch come ad esempio `Feature 1` o `Develop` in figura 1.3.

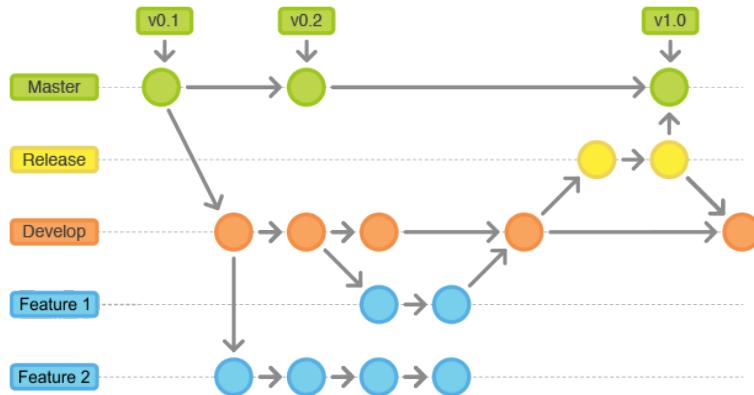


Figura 1.3: Esempio di un workflow a branch

Pull

Il comando `git pull` generalmente recupera i dati dal server da cui originalmente 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



GitHub Classroom è lo strumento messo a disposizione da GitHub per gli insegnanti. Gli step fondamentali per iniziare ad usarlo sono:

- **Nuova organizzazione** Creare un'organizzazione dal pannello personale di GitHub o se già esistente sfruttare quest'ultima.
- **Nuova Classroom** Il passo successivo è creare una nuova Classroom. Si dovrà selezionare l'organizzazione desiderata per poi inserire il nome della classe. C'è la possibilità di invitare altri insegnanti alla classe appena creata o dei membri.
- **Nuovo Compito** A questo punto si può procedere alla creazione di un nuovo *assignment* ovvero un nuovo compito. È possibile fare scegliere se assegnare compiti di gruppo o individuali. Ogni compito ha un titolo, può essere reso pubblico o privato, si può anche aggiungere una repository per dare un insieme di file di base.

2 | Lavori correlati

Parker e Hamblen [Parker and O. Hamblen, 1989] definiscono il plagio di software come "un programma che è stato prodotto da un altro con un esiguo numero di trasformazioni di routine".

Le trasformazioni che possono aver luogo possono variare da quelle più semplici (come cambiare i commenti nel codice oppure cambiare I nomi delle variabili) a quelle più complesse (rimpiazzare strutture di controllo con altre equivalenti, ad esempio sostituire un ciclo "for" con un "while").

CAPITOLO 2. LAVORI CORRELATI

3 | Studio ed implementazione

L'applicazione sviluppata consente di accedere al proprio account GitHub, selezionare l'organizzazione utilizzata per Classroom e infine di scaricare le repositories degli studenti, le quali sono organizzate a loro volta in progetti, i quali corrispondono agli assignment di GitHub Classroom.

Una volta selezionato il progetto da voler prendere in analisi è possibile selezionare le varie repositories contenenti il lavoro degli studenti per poi essere scaricate nella cartella che si desidera. Le repositories possono essere scaricate in due modalità *Salta* o *Aggiorna*. La prima consente di saltare le repository già scaricate in precedenza, mentre la seconda modalità consente di aggiornare all'ultimo commit effettuato nella repository.

Una volta scaricati i lavori che si vogliono analizzare si può eseguire il controllo del plagio. Esso è possibile tramite il tasto nella barra superiore dell'applicazione. Una volta cliccato il pulsante verrà richiesto di selezionare prima la cartella contenente le repositories da analizzare e poi la cartella contenente i files base dell'assignment. I significati di queste directory verranno analizzati in dettaglio nella sezione dedicata a MOSS ([3.1](#))

3.1 MOSS

MOSS è un sistema automatico per determinare la similarità dei sorgenti di programmi [[Aiken, 1994-1997](#)].

MOSS sta per Measure Of Software Similarity, accetta gruppi di documenti e ritorna un insieme di pagine HTML che mostrano, dove significanti, sezioni di coppie di documenti molto simili. MOSS è principalmente usato per *scovare plagi* in progetti di programmazione in informatica e altri corsi di ingegneria, sebbene comunque supporti molti altri formati di testo.

Funzionamento [[Clough, 2000](#)]

Il servizio usa una robusta selezione, cioè sceglie poche fingerprint¹ per la stessa qualità di risultati, rendendolo più efficiente e scalabile rispetto ad

¹In informatica un *algoritmo di fingerprinting* è una procedura che mappa un elemento di dati arbitrariamente grande in una stringa di bit molto più breve che identifica in modo univoco i dati di origine.

altri algoritmi.

In MOSS, l'informazione posizionale, cioè il documento e il numero di linea, è memorizzato per ciascuna fingerprint selezionata.

Il *primo passo* dell'algoritmo costruisce un indice che mappa le fingerprint alle locazioni per tutti i documenti, come l'indice invertito costruito dai motori di ricerca per mappare parole a posizioni nei documenti. Nel *secondo passo*, ciascun documento viene effettuata la fingerprint una seconda volta e le fingerprint selezionate vengono cercate nell'indice; questo ritorna la lista di tutte le fingerprint risultanti per ciascun documento.

Ora la lista delle fingerprint risultanti per un documento d possono contenere le fingerprint di molti documenti differenti d_1, d_2, \dots . Al prossimo passo, la lista delle fingerprint risultanti per ciascun documento d viene ordinata per documento e i risultati vengono raggruppati per ciascun paio di documenti $(d, d_1), (d, d_2)$.

Le coppie di documenti vengono ordinati per rank dato dalla dimensione (numero di fingerprint) e i più grandi vengono riportati all'utente, cioè i documenti con maggior numero di plagi saranno visualizzati per primi.

Notate che fino a quest'ultimo passo, non è stata richiesta nessuna coppia. Questo è molto importante, poiché non potremmo sperare di effettuare il rilevamento di copie confrontando ciascuna coppia di documenti in un grande corpus, rendendolo per questo molto efficiente.

MOSS ha varie centinaia di utenti che sperano di trovare plagi in una molitudine di dati di vari tipi. Esso richiede di specificare il tipo di documenti che andrà ad analizzare. Questo perché per ciascun formato di documenti, provvede ad eliminare le caratteristiche che non dovrebbero distinguere i vari documenti (ad esempio, si elimina gli spazi bianchi dal testo). Questo documento generato da questa scrematura è l'input da dare al motore fingerprint, il quale non sa niente riguardo i differenti tipi di documento, caratteristica che lo rende fortemente scalabile.

Il servizio

MOSS mette a disposizione un web services accessibile tramite vari script e librerie implementate negli anni dagli sviluppatori di tutto il mondo. In particolare nel progetto realizzato è stata usata la libreria java **MOJI** [Zielke], la quale permette di implementare agevolmente un client per interfacciarsi con il servizio offerto da MOSS.

Per prima cosa bisogna registrarsi al servizio inviando una e-mail ll'indirizzo `moss@moss.stanford.edu` con oggetto vuoto e con corpo dell'email come segue:

```
registeruser mail username@domain
```

dove l'ultima parte in italics è il tuo indirizzo email. Dopo pochi minuti riceverai nella mail indicata le istruzioni sul funzionamento di MOSS e cercando `$userid` si troverà il token di autenticazione per poter utilizzare il servizio.

Questa libreria permette di inviare le repositories da controllare in blocco, cioè passando come parametro una cartella strutturata come è mostrato successivamente.

```
solution_directory
|- student1
    |- main.c
    |- ...
|- student2
    |- ...
|_ student3
    |- ...
```

Oltre alle repositories da analizzare è anche possibile inviare una cartella contenente la traccia base. Quest'ultima è l'insieme dei file dati dal professore agli studenti ed è utilizzata per ridurre ulteriormente la possibilità di errori nella valutazione dei plagi e quindi aumentare l'affidabilità dei risultati.

MOSS da la possibilità di settare altri parametri come:

- **optC**: è un commento che apparirà nella pagina HTML di ritorno di MOSS.
- **optD**: indica che l'invio è strutturato a cartelle e non a file. In MOJI c'è solo la versione a Directory.
- **optM**: una porzione di codi quanto specificato. Fornisce un metodo semplice per rilevare il codice base/modello.
- **optN**: definisce quante coppie si devono mostrare nella pagina dei risultati.
- **optX**: supporto alle features sperimentali. Settare a 1 per usare il server sperimentale di MOSS.

Dopo pochi secondi che si sono inviati i documenti da analizzare verrà ritornato un URL che porta alla pagina HTML con i risultati (figura 3.1). I risultati rimarranno nei server di MOSS per circa 14 giorni dopo l'invio della richiesta. All'interno della pagina dei risultati si troveranno le coppie dei progetti incriminati con la rispettiva percentuale di codice simile e il numero righe di codice sospette. La percentuale dei risultati indica il grado di sovrapposizione tra il progetto stesso e quello con cui si è comparato.

CAPITOLO 3. STUDIO ED IMPLEMENTAZIONE

Moss Results		
Wed Jul 4 13:31:37 PDT 2018		
Options -l c -d -n 10		
ESAME PROGRAMMAZIONE 1		
[How to Read the Results Tips FAQ Contact Submission Scripts Credits]		
File 1	File 2	Lines Matched
C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_S2 (98%)	C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_S2 (97%)	1035
C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_mid (74%)	C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_elli (78%)	776
C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_ce (96%)	C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_ndo (97%)	669
C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_fa (85%)	C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_p9 (92%)	831
C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_etta (84%)	C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_etti (86%)	759
C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_etto (76%)	C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_sae (80%)	722
C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_e0 (80%)	C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_sae (75%)	729

Figura 3.1: Esempio di risultati di MOSS

Aprendo uno dei risultati verranno mostrati i sorgenti a confronto (figura 3.2) e nella parte alta della pagina HTML verranno visualizzati in vari colori le parti di codice simili.

The screenshot shows three windows side-by-side. The first window displays the results of the plagiarism check, listing various files and their similarity percentages. The second and third windows show the actual source code for 'File 1' and 'File 2' respectively, with color-coded highlights indicating matching code segments between them.

```

File 1:
C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_S2 (57%)
244-311
183-243
523-377
390-428
C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_S2 (65%)
231-301
168-329
317-369
379-415

File 2:
C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_itta
231-301
168-329
317-369
379-415
C:/Programmazione1-UniPG/prova-finale-di-programmazione-secondo-appello[REDACTED]_itta
231-301
168-329
317-369
379-415

```

Figura 3.2: Esempio pagina analisi sospetto plagio di MOSS

Un’altro aspetto da tenere in considerazione è la privacy e quindi sono state prese delle precauzioni per mantenere private le pagine dei risultati. Per prima cosa i risultati non possono essere scansionati da robots o sfogliati da persone che navigano nel Web. Inoltre il numero casuale nell’URL viene reso noto solo all’account che ha inviato la query e non vi è alcun modo per accedere ai risultati tranne che attraverso quell’URL. Per concludere, le pagine dei risultati scadono automaticamente dopo 14 giorni e il codice non viene mantenuto nel server. L’unione di queste accortezze rende molto basso il potenziale di abuso.

Conclusioni

CONCLUSIONI

Bibliografia

Alex Aiken. Moss: A system for detecting software similarity, 1994-1997.
URL <https://theory.stanford.edu/~aiken/moss/>.

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

Paul Clough. Plagiarism in natural and programming languages: an overview of current tools and technologies, 2000. URL <https://ir.shef.ac.uk/cloughie/papers/plagiarism2000.pdf>.

Alan Parker and James O. Hamblen. Computer algorithms for plagiarism detection. *Education, IEEE Transactions on*, 32:94–99, June 1989.

Björn Zielke. Moji client for the moss plagiarism detection service. URL <https://zielke.it/moji/>.