Corso GIT

LEZIONE 1: Introduzione & Basi di git

Alessio Serraino

serrainoalessio@gmail.com

29/11/2016

POLITECNICO OPEN
unix LABS
Come hack with us.

Version control

Git è un Version Control System (VCS) decentralizzato

Cos'è un sistema di controllo versione?

Version control

Git è un Version Control System (VCS) decentralizzato

Cos'è un sistema di controllo versione?

- Un software che serve a tracciare lo sviluppo di un progetto
- Ciò offre numerose comodità in fase di sviluppo

Version Control

Perché un sistema di controllo versione? Non è sufficiente la funzione UNDO?

- Risposta breve: No

Version Control

Perché un sistema di controllo versione? Non è sufficiente la funzione UNDO?

- Risposta breve: No
- L'undo NON permette di annullare il penultimo cambiamento mantenendo l'ultimo
- In genere chiudendo l'editor si perde la history, oppure la history non è facilmente trasferibile fra più computer
- L'undo salva ogni minima modifica, senza marcare in alcun modo i momenti più importanti
- La funzione undo non aggiunge metadati (messaggi del programmatore, chi e quando ha apportato la modifica...)
- Git può mostrare tutte le modifiche apportate fra diversi momenti di sviluppo

Git basics

L'idea di fondo è salvare lo stato di ogni file del progetto in determinati momenti più importanti, come se si facesse una foto del progetto in un certo istante.
Tali salvataggi sono detti *commit*

Un'altra idea chiave di git è il branch. Un branch è un ramo sul quale si lavora sviluppando un unica macro-feature di un progetto
Dividere un progetto in branch è comodo per mantenerlo ordinato, ed è indispensabile per coordinare un team di sviluppo
Per un po' ci occuperemo di progetti su un singolo branch

Git basics

L'idea di fondo è salvare lo stato di ogni file del progetto in determinati momenti più importanti, come se si facesse una foto del progetto in un certo istante.
Tali salvataggi sono detti *commit*

Un'altra idea chiave di git è il branch. Un branch è un ramo sul quale si lavora sviluppando un unica macro-feature di un progetto
Dividere un progetto in branch è comodo per mantenerlo ordinato, ed è indispensabile per coordinare un team di sviluppo
Per un po' ci occuperemo di progetti su un singolo branch

Git offre numerose altre possibilità, che saranno discusse nelle lezioni successive, in genere accessibili ciascuna tramite il proprio comando, sempre nel formato

```
git [comando] [--opzioni]
```

Git init

Come facciamo in modo che il nostro progetto venga tracciato da git?

Git init

Come facciamo in modo che il nostro progetto venga tracciato da git?

Entriamo nella directory da tracciare, e digitiamo il comando

```
git init
```

Git init

Il comando **git init** creerà la cartella nascosta .git nella directory che vogliamo tracciare.

Questa cartella contiente tutti i files di git (database del progetto, config, etc.), e può essere tranquillamente ignorata.
Meno viene toccata e meglio è.

Si può eliminare .git e tutto il suo contenuto per annullare un git init (e rimuovere ogni informazione relativa al tracciamento)

Stato dei files

Dal momento dell'inizializzazione della directory git ha cominciato a tracciare i files al suo interno.
Ogni file può ricadere in 4 stati diversi:

- Untracked - non tracciato, non è un vero e proprio stato
- Modified (o Modificato) - il file è stato modificato dall'ultimo salvataggio nel database
- Staged (o in Stage) - il file è stato contrassegnato per essere salvato nel database
- Committed (o Committato) - una copia del file è salvata al sicuro nel database Git

Stato dei files

Dal momento dell'inizializzazione della directory git ha cominciato a tracciare i files al suo interno.
Ogni file può ricadere in 4 stati diversi:

- Untracked - non tracciato, non è un vero e proprio stato
- Modified (o Modificato) - il file è stato modificato dall'ultimo salvataggio nel database
- Staged (o in Stage) - il file è stato contrassegnato per essere salvato nel database
- Committed (o Committato) - una copia del file è salvata al sicuro nel database Git

Il perché dello stato Staged sarà più chiaro a breve

Stato dei files

Dal momento dell'inizializzazione della directory git ha cominciato a tracciare i files al suo interno.
Ogni file può ricadere in 4 stati diversi:

- Untracked - non tracciato, non è un vero e proprio stato
- Modified (o Modificato) - il file è stato modificato dall'ultimo salvataggio nel database
- Staged (o in Stage) - il file è stato contrassegnato per essere salvato nel database
- Committed (o Committato) - una copia del file è salvata al sicuro nel database Git

Il perché dello stato Staged sarà più chiaro a breve
Per vedere lo stato dei files in qualsiasi momento si può usare il comando **git status**

Git status

```
$git status
Untracked files:
  (use "git add <file>..." to include in what will be c
    pippo.c
    main.c
nothing added to commit but untracked files present (us
$git add pippo.c
$git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    new file: pippo.c
Untracked files:
  (use "git add <file>..." to include in what will be c
    main.c
```

Stato dei files - Untracked

Untracked significa che di tale file non saranno tracciate le modifiche

Per tracciare un file non tracciato basta usare il comando **git add [nomefile]**, che lo aggiunge alla staging area

A volte può essere comodo non tracciare alcuni files, per esempio i binari di build

In tal caso si ricorre al file **.gitignore**

Stato dei files - .gitignore

Deve essere sempre piazzato nella root della directory da tracciare

Ogni linea contiene il nome di un file o un pattern da ignorare

I files che terminano per / sono directory, in tal caso sarà ignorata tutta la directory

Come in bash si può usare il carattere * per matchare ogni stringa. Per esempio *.out matcherà tutti i files con estensione .out

Sebbene .gitignore indichi quali files non tracciare il file .gitignore va tracciato, quindi:

```
$git add .gitignore
```

Stato dei files - .gitignore

- Esempio di gitignore

```
# Io sono un commento

# Tutti i files nella cartella build
build/

# Tutti i files .out, .a, .o
*.out
*.a
*.o

# il file do_not_track_me.txt
do_not_track_me.txt
```

Stato dei files - Staged

È lo stato di un file nella staging area.

La staging area è un'area temporanea prima della commit.

Ogni commit prenderà in considerazione solo files inseriti nella staging area.

Per aggiungere ogni file alla staging area (quindi contrassegnarlo per la prossima commit) si usa il comando **git add [nomefile]**

La versione **git add -A** aggiungerà alla staging area tutti i files (anche quelli non tracciati) che non compaiono nel gitignore

Stato dei files - Modified, Commited

Un file committed è un file che è rimasto *immodificato* dall'ultima commit.

Immediatamente dopo un commit tutto il contenuto della staging area passa allo stato Commited

Un file Modified è un file che non è stato inserito nella staging area, e che è stato modificato dall'ultima commit.

Se un file viene inserito nella staging area, poi modificato, le modifiche NON entreranno nella staging area. Tale file apparirà staged nella versione in cui è stato aggiunto alla staging area, e modified nella versione non aggiunta alla staging area.

Git commit

Il comando **git commit** dice a git di memorizzare e congelare lo stato del progetto nel suo database interno. Una commit è lo stato del progetto una volta che viene salvato.

Git commit

Il comando **git commit** dice a git di memorizzare e congelare lo stato del progetto nel suo database interno. Una commit è lo stato del progetto una volta che viene salvato.

- I commit devono essere quanto più piccoli e frequenti possibile, ed includere solo modifiche fra loro coerenti

Git commit

Il comando **git commit** dice a git di memorizzare e congelare lo stato del progetto nel suo database interno. Una commit è lo stato del progetto una volta che viene salvato.

- I commit devono essere quanto più piccoli e frequenti possibile, ed includere solo modifiche fra loro coerenti
- Ogni commit deve essere almeno vagamente funzionante. Il codice committato dovrebbe come minimo compilarsi
  - Se devo condividere il codice è utile che tutti possano scaricare il progetto e compilarlo
  - Se mi servisse necessariamente una vecchia versione, è meglio se ho sempre un progetto che come minimo si compila

Git commit

Il comando **git commit** dice a git di memorizzare e congelare lo stato del progetto nel suo database interno. Una commit è lo stato del progetto una volta che viene salvato.

- I commit devono essere quanto più piccoli e frequenti possibile, ed includere solo modifiche fra loro coerenti
- Ogni commit deve essere almeno vagamente funzionante. Il codice committato dovrebbe come minimo compilarsi
  - Se devo condividere il codice è utile che tutti possano scaricare il progetto e compilarlo
  - Se mi servisse necessariamente una vecchia versione, è meglio se ho sempre un progetto che come minimo si compila
- I vecchi commit non devono mai essere eliminati

Git commit

Il comando **git commit** dice a git di memorizzare e congelare lo stato del progetto nel suo database interno. Una commit è lo stato del progetto una volta che viene salvato.

- I commit devono essere quanto più piccoli e frequenti possibile, ed includere solo modifiche fra loro coerenti
- Ogni commit deve essere almeno vagamente funzionante. Il codice committato dovrebbe come minimo compilarsi
  - Se devo condividere il codice è utile che tutti possano scaricare il progetto e compilarlo
  - Se mi servisse necessariamente una vecchia versione, è meglio se ho sempre un progetto che come minimo si compila
- I vecchi commit non devono mai essere eliminati

Esiste sempre un particolare puntatore a commit, chiamato **HEAD**, esso punta sempre all'ultima commit

Git commit - message

Ogni commit ha un suo messaggio di commit, che dovrebbe specificare cosa è stato modificato

Ogni commit dovrebbe avere un suo messaggio

Per inserire un messaggio insieme alla commit si può usare l'opzione -m, seguita dal testo

```
git commit -m "messaggio di commit"
```

Il messaggio di commit è sostanzialmente obbligatorio, se non viene specificata l'opzione -m sarà aperto l'editor di default per inserire un messaggio. Salvate ed uscite per concludere il processo

Git log

Il comando

```
git log
```

mostra la storia di un progetto. La configurazione di default mostra ogni commit in ordine cronologico, esistono modi per ottenere più o meno informazioni e formattarle in modo diverso, non ce ne occuperemo in questo corso

Di ogni commit viene mostrato il messaggio, l'autore, una data ed un hash che identifica univocamente il commit.

```
$git log
commit 78b78694f40fac6533971c137402b9d9073f0848
Author: Alessio Serraino <serrainoalessio@gmail.com>
Date: Fri Nov 25 18:34:11 2016 +0100
    aggiunto main.c
```

Git diff

- Il comando **git diff** mostra tutti i cambiamenti fra due momenti del progetto

- Chiamato senza argomenti mostra le differenze fra lo stato attuale e la staging area

- Chiamato con un argomento aggiuntivo **git diff [commit]** mostra le differenze fra lo stato corrente ed un certo particolare commit, eventualmente **HEAD** per riferirsi l'ultimo commit

- Chiamato con l'opzione --**cached** oppure --**staged git diff --cached**
  mostra le differenze fra la staging area e l'ultimo commit

Git diff - formato

Il formato è circa lo stesso del comando unix **diff**
Il diff compara le linee di codice, non i singoli caratteri
Le impostazioni di default mostrano il codice colorato nel modo seguente:

- Ogni linea rossa che comincia con un - è una linea tolta
- Ogni linea verde che comincia con un + è una linea aggiunta

Git diff - formato

```
@@ -1,26 +1,23 @@
 #include <stdio.h>
-#include <stdlib.h>
+#include <math.h>
 int is_prime(int n) {
     int i;
-    for (i = 2; i < n; i++)
+    for (i = 2; i <= sqrt(n); i++)
         if (n % i == 0)
             return 0;
     return 1;
 }
-int funzione_inutile(void) {
-    // Estrae 1Mb di dati casuali e li scarica in /dev/null
-    system("dd if=/dev/urandom of=/dev/null count=256 bs=4K");
-}
```

Git branch

Cos'è un branch?

Git branch

Cos'è un branch?

Un branch è un particolare ramo di svuluppo del progetto. Essi consentono di mantenere ordinato, pulito, e soprattutto *modulare* il codice ed i relativi cambiamenti. Ogni branch ha un nome.

- In genere in ogni branch si sviluppano insieme funzioni strettamente correlate o librerie interne di un software
- È bene creare un branch per ogni isnsieme di funzioni strettamente correlate o librerie interne di un software
- È necessario eseguire un merging dei vari branch prima di rilasciare il software

Git branch

Cos'è un branch?

Un branch è un particolare ramo di svuluppo del progetto. Essi consentono di mantenere ordinato, pulito, e soprattutto *modulare* il codice ed i relativi cambiamenti. Ogni branch ha un nome.

- In genere in ogni branch si sviluppano insieme funzioni strettamente correlate o librerie interne di un software
- È bene creare un branch per ogni isnsieme di funzioni strettamente correlate o librerie interne di un software
- È necessario eseguire un merging dei vari branch prima di rilasciare il software

Al momento dell'inizializzazione di git in una cartella viene sempre, e da subito, creato il branch *master*.

Git branch - in pratica

Come si crea un branch?

Git branch - in pratica

Come si crea un branch?

Con il comando

```
git branch [nomebranch]
```

Git branch in realtà non crea nulla di nuovo, semplicemente inizializza un'altro puntatore a **HEAD**
Eseguendo un commit subito dopo aver eseguito git branch però sembra non succedere nulla. Infatti il branch è stato creato, ma il branch su cui committiamo è ancora master.
Dopo un commit il nuovo branch punterà ancora al penultimo commit, master all'ultimo.

Git branch - in pratica

Come si crea un branch?

Con il comando

```
git branch [nomebranch]
```

Git branch in realtà non crea nulla di nuovo, semplicemente inizializza un'altro puntatore a **HEAD**
Eseguendo un commit subito dopo aver eseguito git branch però sembra non succedere nulla. Infatti il branch è stato creato, ma il branch su cui committiamo è ancora master.
Dopo un commit il nuovo branch punterà ancora al penultimo commit, master all'ultimo.
Questo perché il nuovo branch non è il branch attivo, esso è ancora quello vecchio. Possiamo ovviamente attivare il nuovo branch.

Git branch - in pratica

Come si passa da un branch ad un altro?

Git branch - in pratica

Come si passa da un branch ad un altro?

- Con il comando

  `git checkout [nomebranch]`

Dopo aver eseguito questo comando ogni modifica apportata avrà come radice il branch scelto

Git branch - in pratica

Come si passa da un branch ad un altro?

- Con il comando

  `git checkout [nomebranch]`

Dopo aver eseguito questo comando ogni modifica apportata avrà come radice il branch scelto

- Esiste anche la possibilità di creare e switchare il branch insieme, con il comando

  `git checkout -b [nuovobranch]`

Git merge

Come si fondono insieme due branch?

Git merge

Come si fondono insieme due branch?

Con il comando

```
git merge [nomebranch]
```

Questo comando mergerà il branch indicato nel branch corrente, ovvero apporterà al branch corrente tutte le modifiche eseguite nel branch da unire

Git merge

Come si fondono insieme due branch?

Con il comando

```
git merge [nomebranch]
```

Questo comando mergerà il branch indicato nel branch corrente, ovvero apporterà al branch corrente tutte le modifiche eseguite nel branch da unire
ATTENZIONE: Questa operazione potrebbe mettere in luce dei conflitti, in tal caso git aggiunge una sintassi particolare nei punti dei files interessati. Il conflict deve essere risolto manualmente.

Git merge

Come si fondono insieme due branch?

Con il comando

```
git merge [nomebranch]
```

Questo comando mergerà il branch indicato nel branch corrente, ovvero apporterà al branch corrente tutte le modifiche eseguite nel branch da unire
ATTENZIONE: Questa operazione potrebbe mettere in luce dei conflitti, in tal caso git aggiunge una sintassi particolare nei punti dei files interessati. Il conflict deve essere risolto manualmente.

È possibile eliminare il branch dopo averlo mergiato?

Git merge

Come si fondono insieme due branch?

Con il comando

```
git merge [nomebranch]
```

Questo comando mergerà il branch indicato nel branch corrente, ovvero apporterà al branch corrente tutte le modifiche eseguite nel branch da unire
ATTENZIONE: Questa operazione potrebbe mettere in luce dei conflitti, in tal caso git aggiunge una sintassi particolare nei punti dei files interessati. Il conflict deve essere risolto manualmente.

È possibile eliminare il branch dopo averlo mergiato?

- Sì, è sicuro eliminare il branch, con il comando

  ```
  git branch -d [nomebranch]
  ```

Git merge - conflicts

In caso di conflitto git mostrerà un messaggio di errore ed inserirà una sintassi particolare nei punti dei files interessati. A questo punto il conflict va risolto manualmente.
Finchè i conflict non vengono risolti git mostrerà messaggi di errore se si cerca di commitare. Git status ci informerà dei files che contengono conflicts.

```
<<<<<<< HEAD
printf("Hello world!\n"); // Questo codice viene dal master
=======
printf("The cake is a lie\n"); // Questa è una modifica nel branch
>>>>>>> newBranch
```

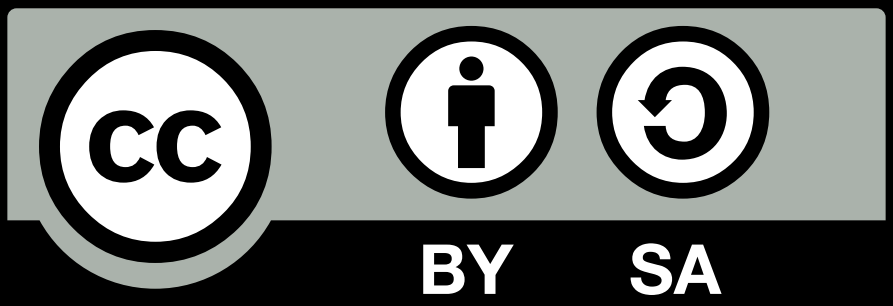
In tal caso si deve scegliere una delle modifiche, salvare il file, ed aggiungerlo alla staging area. Quando tutti i conflict sono risolti committare.

Fonti

https://git-scm.com/book/en/v2
https://git-scm.com/docs/
http://learngitbranching.js.org/
I ragazzi del POuL
Molto poco dalla mia esperienza

Fine

Grazie per l'attenzione!



`http://www.poul.org`