GIT - CORSO INTENSIVO

V. 0.4

Indice

Introduzione

Perché scegliere GIT?

TODO

Installare GIT

Ubuntu

Teoria

Repository Locali e Remoti

Git permette di gestire *contemporaneamente* più **repository** per lo stesso progetto. Esistono due tipi di repository:

- **Repository Locale:** È unico e rappresenta la copia del progetto sul PC dell'utente.
- **Repository Remoto:** Rappresenta un repository situato in remoto, solitamente un servizio di hosting (ad esempio GitHub). Possono essere uno o più. Ad esempio possiamo avere un nostro repo remoto in cui condividiamo il nostro sviluppo (origin) e uno o più repo remoti che contengono i rami remoti dei nostri "collaboratori" (per approfondire vedi *Pratica*)

Rami

Git permette, come molti altri CVS, di gestire in parallelo più **rami** per ogni repository. Nel mio repo locale posso, per esempio, tenere un ramo principale (solitamente chiamato `master`) che mantiene una versione *funzionante* del programma e sperimentare modifiche anche radicali su un ramo `nuova_feature`. Le modifiche che apporto nel ramo `nuova_feature` non verranno visualizzate quando mi trovo nel ramo `master`. In pratica è come se potessi saltare in modo tremendamente veloce e facile in due stati diversi del progetto.

Anche i repository remoti possono avere più rami. Solitamente i rami vengono indicati con:

`nome_repository/nome_ramo`

In generale però il repository locale non ha un nome quindi la prima parte viene saltata. Possiamo quindi avere cose come master, `origin/master`, `upstream/dev` e via dicendo.

Commit Selettivo

Git permette di fare il commit solo su alcuni file alla volta. Se ho modificato due file `A.cpp` e `B.cpp` posso

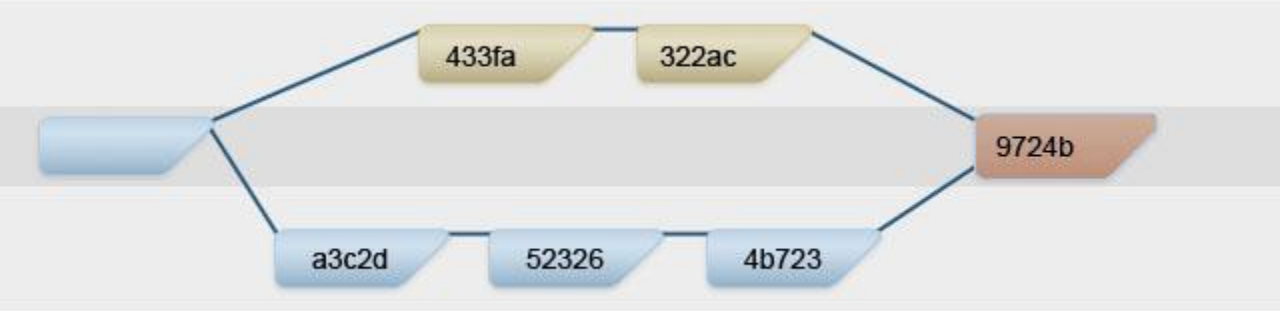
eseguire prima un commit solo su `A.cpp` e poi solo su `B.cpp` Questo è molto comodo per suddividere *logicamente* i vari commit. Per esempio nel primo commit potrei aver risolto un problema di networking mentre nel secondo aver aggiunto una nuova funzionalità a `B.cpp` Dividendo le due modifiche in due commit le modifiche appariranno più chiare.

Questo meccanismo si chiama **stage**. Prima di fare un commit si devono infatti aggiungere i file allo stage. (il parametro `-a` aggiunge automaticamente tutti i file modificati allo stage corrente).

Merge e Rebase

Se abbiamo, come buona norma, effettuato le nostre modifiche su un ramo di sviluppo ci troviamo nella condizione di ricombinare questo ramo con il ramo principale. Ci sono due approcci: il **merge** e il **rebase**.

MERGE



REBASE

Come si può vedere i comportamenti sono sostanzialmente diversi:

- Il **merge** tiene i due rami separati e li unisce creando una nuova revisione che unisce e mescola i due rami. Da notare che il merge non elimina il ramo che viene mescolato, in questo modo possiamo continuare ad usarlo per sviluppare nuove features.
- Il **rebase** riapplica tutto il ramo in coda all'ultima revisione del ramo principale.

Il rebase crea quindi una storia più lineare ed è meno suscettibile ai "merge-conflict". Tuttavia se usato male crea un enormità di revisioni duplicate quindi sarebbe meglio attenersi a questa regola:

MAI FARE IL REBASE DI RAMI PRESI DA REMOTO MA SOLAMENTE DI RAMI LOCALI

La scelta fra rebase e merge è tutta vostra, usate quella che vi sembra più comoda.

Pratica

Configurazioni Base

Prima di utilizzare Git bisogna attuare alcune configurazioni basilari:
http://davideaversa.it/slashcode/2010/06/git-configurazioni-utili/

TODO: Trasferire ed ampliare la descrizione dal blog a qui.

Creare un Repository Vuoto

Per creare un repository vuoto si usa il comando:

```
gitinit
```

Questo inizializza un repository vuoto nella cartella in cui viene dato il comando.

CASO D'USO
Voglio creare un nuovo repository nella cartella di un mio progetto. Entro nella cartella e digito il comando sopra indicato.

Aggiungere File allo Stage di un commit

Per aggiungere file allo stage corrente si usa il comando:

```
gitadd <nomefile>
gitadd <percorso>
```

Il primo comando aggiunge un singolo file, mentre il secondo aggiunge tutti i file all'interno del percorso specificato (ricorsivamente sulle cartelle).

CASO D'USO
1) Ho un repository vuoto. Voglio aggiungere tutti i file del progetto. Vado nella cartella e digito

```
gitadd .
```

2) Voglio preparare per un commit il file main.cpp. Vado nella cartella e digito

```
gitadd main.cpp
```

Fare un Commit

Per fare un commit basta usare il comando:

```
gitcommit
```

```
gitcommit-a
gitcommit-m "messaggio"
```

Il primo comando inizia il commit dei file attualmente presenti nello stage. Il secondo comando aggiunge tutti i file modificati allo stage ed effettua il commit. Il terzo comando effettua il commit con il messaggio "messaggio". Comodo per commit molto semplici.

CASO D'USO
1) Ho appena modificato i file `network.cpp` e `interface.cpp` Voglio fare due commit separati poiché la prima modifica riguarda la rete mentre la seconda sono variazioni all'interfaccia carica.
La prima cosa che faccio è aggiungere la prima modifica allo stage, poi faccio il commit:

```
gitadd network.cpp
gitcommit-m "Protocollodi retemigliorato"
```

Poi faccio lo stesso con la seconda modifica:

```
gitadd interface.cpp
gitcommit-m "FIX:Erroregraficorisolto."
```

2) Sempre con le stesse due modifiche voglio velocemente fare il commit di tutti i cambiamenti che ho effettuato.

```
git-am "FIX:Correttialcunierrori"
```

Rimuovere, Spostare e Rinominare file

Per rimuovere, spostare e rinominare un file all'interno del progetto è **obbligatorio** utilizzare i comandi interni di git. Questo garantisce che i file rimangano sempre **correttamente** indicizzati all'interno del repository locale di git. I comandi sono del tutto simili a quelli classici di bash ma vanno preceduti dal comando git.

```
gitrm <nome_file>
gitmv <sorgente><destinazione>
```

CASO D'USO
Voglio spostare il file main.cpp nella cartella src.

```
gitmv main.cppsrc/
```

Visualizzare e Creare un nuovo ramo

Per visualizzare tutti i rami locali basta digitare:

```
gitbranch
```

Per creare un nuovo ramo:

```
git branch nuovo_ramo
```

CASO D'USO
Voglio creare un nuovo ramo in cui provare a sviluppare la feature "scrnshot". Vado nella cartella e digito

```
git branch screenshot
```

Attivare un ramo

Per saltare da un ramo ad un altro si usa il comando:

```
git checkout nome_ramo
```

CASO D'USO
Dopo aver creato il ramo screenshot voglio attivarlo e cominciare a sviluppare la feature. Entro nella cartella e digito:

```
git checkout screenshot
```

Eliminare un ramo

Dopo aver usato un ramo e dopo aver fatto il merge nel ramo principale possiamo decidere di eliminarlo dal nostro repository.

```
git branch -d <nome_ramo_locale>
git branch -d -r <nome_ramo_remoto>
```

Si possono eliminare sia rami locali che rami remoti. Eliminando un ramo remoto, però, **viene eliminato solo il riferimento ad esso nel repo locale e NON il repo remoto**. Questo significa che al prossimo fetch e/o pull tali rami verranno ricreati.

Aggiungere un repository remoto

E' possibile collegarsi a repository remoti con il comando:

```
git remote add <nome_repo_remoto> <url>
```

CASO D'USO
1) Faccio parte di un team di sviluppatori e voglio aggiungere al mio repo locale e al mio repo remoto il repository principale (**upstream**) da cui posso sempre aggiornare il mio "master" con gli aggiornamenti inviati da tutti gli sviluppatori.

```
git remote add upstream <url>
```

2) Sono a capo di un progetto software con più sviluppatori. Voglio aggiungere i repository remoti dei miei collaboratori dai quali posso aggiornare il ramo principale del progetto.

```
git remote add primo_sviluppatore <url_primo_sviluppatore>
git remote add secondo_sviluppatore <url_secondo_sviluppatore>
```

Scaricare un repository remoto e aggiungerlo ai miei

Per recuperare i dati da un repo remoto appena aggiunto o per recuperarne gli aggiornamenti **senza unirlo ad un ramo locale** posso usare il comando:

```
git fetch <nome_remoto>
```

CASO D'USO

Dopo aver aggiunto i rami dei miei collaboratori voglio recuperare i dati per poter valutare le loro modifiche e selezionare cosa va aggiunto.

```
git fetch primo_sviluppatore
git fetch secondo_sviluppatore
```

Aggiornare un repository remoto

Bast asemplicemente dare il comando:

```
git push <repo_remoto> <ramo_da_inviare>
```

CASO D'USO

Voglio aggiornare il ramo origin/master con il mio master locale.

```
git push origin master
```

Aggiornare un ramo locale con un ramo remoto

Per recuperare i dati da un repo remoto appena aggiunto o per recuperarne gli aggiornamenti **ed unirlo al ramo locale corrente** posso usare il comando:

```
git pull <repo_remoto> <ramo_remoto>
```

CASO D'USO

Voglio recuperare le modifiche del ramo upstream/master per aggiornare il mio ramo "master" locale con le ultime modifiche fatte in upstream:

```
git checkout master (mi assicuro di stare in master)
git pull upstream master
```

Ripristinare un file da un commit precedente

```
git checkout $(git rev-list -n 1 HEAD -- "$file")^-- "$file"
```

git rev-list restituisce l'ultimo commit in cui è stato modificato il file $file. git checkout ripristina la versione di quel commit specifico.

Commenti e feedback

Se vi è piaciuta, se vi è stata utile, se pensate che manchi qualcosa di importante, se pensate che ci sia qualcosa di impreciso... **non mancate di commentare e contattarmi.** Mi potete trovare sempre su **http://slashcode.davideaversa.it.**

Prontuario

`git init`	Inizializza un repo vuoto.
`git add <nome_file/percorso>`	Aggiunge un file o un percorso allo stage corrente.
`git commit`	Effettua il commit delle modifiche in stage.
`git branch`	Visualizza, crea o rimuove un ramo.
`git checkout <ramo>`	Passa al ramo <ramo>.
`git remote`	Gestisce i repo remoti (aggiunge, elimina, rinomina, cambia url, etc...)
`git fetch`	Scarica oggetti da un repo remoto. Di solito si usa per scaricare tutti i rami da un repo remoto.
`git push`	Carica un ramo locale in un repo remoto (se il repo non esiste viene creato).
`git pull`	Scarica un ramo da un repo remoto e lo unisce al ramo corrente.
`git merge`	Fa il merge fra due rami locali.
`git rebase`	Fa il rebase fra due rami locali.
`git cherry-pick`	Unisce al ramo corrente **un commit specifico** preso da un altro ramo.