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”

**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.

Glossario

**Branch**

Ramo di sviluppo

**Commit**

Il nome di una revisione particolare

**Merge**

Fusione

**Revision**

Revisione

Comandi di base a riga di comando

Di seguito potete trovare una mini guida con i comandi principali di git, che vi permetterà di iniziare ad esplorare le possibilà di questo potente strumento.

**Creare il repository per un proprio progetto**

$ gitinit

Eseguendo gitinit dentro alla cartella principale di un progetto, verrà creato il repository locale. A differenza di svn che crea cartelle nasconde dentro a ciascuna cartella del vostro progetto, git crea una sola cartella .git nella radice del progetto. Questo è un vantaggio perchè mantiene più "pulite" le vostre cartelle.

**Scaricare (clonare) un repository già esistente**

$ git clone git://url/nome-progetto.git

**Aggiungere file da inserire nella prossima versione**

$ gitadd

Ci sono varie modalità per aggiungere i file.

* gitadd . per aggiungere tutti i file, comprese le sotto cartelle
* gitaddnomeCartella per aggiungere una singola cartella
* gitadd percorso/nomeFile per aggiungere un singolo file

**Ignorare file e/o cartelle**

Spesso all'interno di un singolo progetto ci sono cartelle e file che non devono mai essere messe nel repository, si pensi ad esempio alla cartella con la cache, oppure a quella dei log, ecc. Lasciarle visibili, oltre ad essere fastidioso fornisce una possibilità in più per compiere degli errori ed inserirli erroneamente in qualche versione. Proprio per questo scopo, esiste la possibilità di non fare comparire tali file e cartelle nell'elenco del nostro progetto. Basta creare nella radice del progetto un file chiamato .gitignore. All'interno è possibile elencare una serie di pattern con i file da escludere. Di seguito riporto alcuni esempi:

#Il simbolo di cancelletto inizia un commento e i caratteri successivi vengono ignorati

\*.tmp # Ignora i file con estensione .tmp

cache/ # Ignora i file della cartella cache

log.txt # Ignora il file log.txt

**Visualizzare lo stato**

$ git status

Visualizza lo stato dei file, dividendoli in 3 gruppi. Modifiche non preparate, Modifiche preparate, (saranno nella nuova revisione)

**Changes to be committed**

Mostra i file modificati (rispetto alla commit precedente), che sono stati aggiunti tramite gitadd e che quindi sono pronti per il commit

**Changedbutnotupdated**

Mostra i file modificati (rispetto alla commit precedente), che non sono stati aggiunti tramite gitadd e che quindi non verranno inseriti nel prossimo commit

**Untrackedfiles**

Mostra i file che non sono mai stati inseriti nel controllo versione di git

**Vedere i dettagli delle modifiche effettuate sui file**

$ gitdiff

Con git status vediamo i file che sono stati modificati, mentre per entrare nei dettagli delle righe modificate è necessario utilizzare gitdiff

* gitdiff Mostra le righe che sono state cambiate nei file che non sono ancora stati preparati per la commit, confrontati con la copia presente nell'ultima commit
* gitdiff –-staged Mostra le righe che sono state cambiate nei file che sono stati preparati per la commit, confrontati con la copia presente nell'ultima commit

**Abbandonare le modifiche fatte ad un file nella cartella di lavoro**

$ git checkout -- nomeFile

Lanciando il comando git checkout -- nomeFile il file nomeFile presente nella cartella di lavoro viene cancellato e sovrascritto con quello presente nell'ultima versione. Invece, usando

$ git checkout -f

otterremo lo stesso effetto, su tutti i file modificati dall'ultimo commit.

**Inviare le modifiche**

$ gitcommit

Una volta che sono state effettuate le modifiche e che sono stati aggiunti i file da inserire nel prossimo blocco di modifiche tramite il comando gitadd, è possibile inviare effettivamente le modifiche utilizzando:

$ gitcommit -m "Descrizione delle modifiche"

-m è l'opzione per inserire il messaggio (che per altro è obbligatorio, se non viene inserito come argomento comparirà un editor perché tu possa scrivere più comodamente)

**Visualizzare lo storico dei commit**

$ git log

git log --color mostra lo storico dei commit. Tramite qusto comando è possibile generare anche delle patch. Di questo comando esiste anche una versione grafica; si lancia con gitk e permette anche di vedere le modifiche apportate ai singoli file.

**Creare un branch**

$ git checkout

Lanciando git checkout -b nomeDelBranch si crea un nuovo brach chiamo nomeDelBranch. I successivi comandi (status, commit) vengono riferiti a questo branch

**Tornare al ramo principale**

$ git checkout master

Per tornare al ramo principale, chiamato master.

**Sincronizzarsi con le ultime modifiche**

$ git pull

Eventuali modifiche effettuate da altre persone possono essere scaricate con git pull

**Fondere un ramo secondario con il principale**

$ git merge

Per fondere un ramo secondario nomeDelBranch con il principale (master), portarsi su master se non ci si è già, con:

$ git checkout master

Per effettuare la fusione:

$ git merge nomeDelBranch

**Aggiornare il repository remoto**

$ gitpush

**Generare la Propria Chiave Pubblica SSH**

Come detto precedentemente, molti server Git usano l'autenticazione con la chiave pubblica SSH. Per poter avere una chiave pubblica, ogni utente del tuo sistema deve generarne una se già non la possiede. Questo processo è simile per tutti i sistemi operativi. Primo, devi controllare di non avere già una chiave. Di base, le chiavi SSH degli utenti sono salvate nella directory ~/.ssh. Puoi facilmente controllare spostandoti nella directory e controllandone il contenuto:

$ cd ~/.ssh

$ ls

authorized\_keys2 id\_dsaknown\_hosts

config id\_dsa.pub

Devi cercare una coppia di chiavi dal nome simile a qualcosa e qualcosa.pub, dove quel qualcosa in genere è id\_dsa o id\_rsa. Il file .pub è la tua chiave pubblica e l'altro file è la chiave privata. Se non hai questi file (o non hai una directory .ssh), puoi crearle avviando un programma chiamato ssh-keygen, che è fornito assieme al pacchetto SSH sui sistemi Linux/Mac ed è fornito dal pacchetto MSysGit su Windows:

$ ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/Users/schacon/.ssh/id\_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /Users/schacon/.ssh/id\_rsa.

Your public key has been saved in /Users/schacon/.ssh/id\_rsa.pub.

The key fingerprint is:

43:c5:5b:5f:b1:f1:50:43:ad:20:a6:92:6a:1f:9a:3a schacon@agadorlaptop.local

Prima chiede la conferma dove vuoi salvare la chiave (.ssh/id\_rsa) e poi chiede due volte la passphrase, che puoi lasciare vuota se non vuoi inserire una password quando usi la chiave.

Ora, ogni utente che ha fatto questo deve inviare la propria chiave pubblica a te o a chi amministra il server Git (supponiamo che tu stia usando un server SSH impostato in modo da richiedere le chiavi pubbliche). Tutto quello che devono fare è copiare il contenuto del file .pub ed inviarlo via e-mail. La chiave pubblica è qualcosa di simile a questo:

$ cat ~/.ssh/id\_rsa.pub

ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAklOUpkDHrfHY17SbrmTIpNLTGK9Tjom/BWDSU

GPl+nafzlHDTYW7hdI4yZ5ew18JH4JW9jbhUFrviQzM7xlELEVf4h9lFX5QVkbPppSwg0cda3

Pbv7kOdJ/MTyBlWXFCR+HAo3FXRitBqxiX1nKhXpHAZsMciLq8V6RjsNAQwdsdMFvSlVK/7XA

t3FaoJoAsncM1Q9x5+3V0Ww68/eIFmb1zuUFljQJKprrX88XypNDvjYNby6vw/Pb0rwert/En

mZ+AW4OZPnTPI89ZPmVMLuayrD2cE86Z/il8b+gw3r3+1nKatmIkjn2so1d01QraTlMqVSsbx

NrRFi9wrf+M7Q== schacon@agadorlaptop.local

**Accesso tramite RSA GitHub**

Creare la nostra SSH-KEY tramite un comando da terminale(myemail@mail.com=email di registrazione a GitHub):

misdur@getsugatenshou:~$ ssh-keygen -t rsa -C "myemail@mail.com" -f ~/.ssh/git\_rsa

i seguenti dati sono opzionali

-t tipo del file

-C comment se presente inserisce come nome finale del keygen“[myemail@mail.com](mailto:myemail@mail.com)”

f ~/.ssh/git\_rsaridireziona la chiave nel file git\_rsa

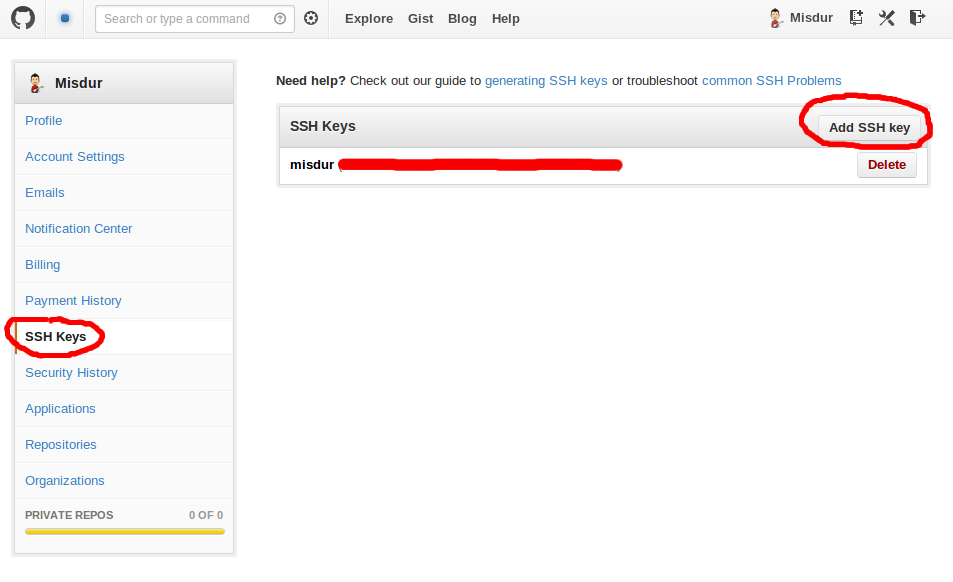
Seguiamo le istruzioni e il gioco e` fatto.

La chiave verra` creata nel file di sistema ~/.ssh/git\_rsa.pub

Ora dobbiamo copiare la nostra public key dal file in questione:

misdur@getsugatenshou:~$ cat ~/.ssh/git\_rsa.pub

Copiamo per intero l’output del cat e andiamo nelle nostre impostazioni su [GitHub](https://github.com/settings/ssh) e aggiungiamo la nostra key

[](http://misdur.altervista.org/wp-content/uploads/2013/05/Schermata-del-2013-05-02-141959.png)

Opzionale

Come gia` detto Git utilizza il protocollo ssh per connettersi a GitHub e di default utilizza laprivate key nel file ~/.ssh/id\_rsa; la nostra key si trova in un altro file e bisogna che git “vada a pescarsi” la nostra chiave che e` nel file ~/.ssh/git\_rsa. Per fare cio`ci sono due modi:

1. Quindi creiamo il file~/.ssh/config e aggiungiamo al suo interno: (obbligatorio in caso di chiave multipla)

Host github.com  
User git  
IdentityFile ~/.ssh/git\_rsa

Ora controlliamo se git riesce a comunicare con server remoto:

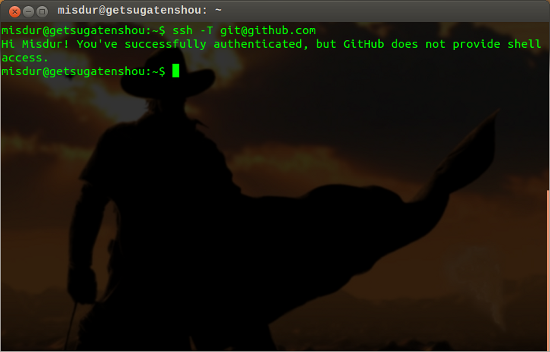
misdur@getsugatenshou:~$ ssh -T git@github.com

1. Se non si e creato il file config eseguire il seguente comanda per verificare la connessione

misdur@getsugatenshou:~$ ssh -T [git@github.com](mailto:git@github.com) -i ~.ssh/git\_rsa

oppure

ssh -i ̴.ssh/git\_rsa -T git@github.com

[](http://misdur.altervista.org/wp-content/uploads/2013/05/Schermata-del-2013-05-02-135809.png)

Fatto cio` non ci resta che clonarci il repository e iniziare a lavorare in locale sul nostro progetto.

misdur@getsugatenshou:~$ git clone git@github.com:username/myrepo.git

**Accesso tramite RSA Server Privato**

Per clonare un repositoryGit via SSH, puoi specificare un URL ssh:// come questo:

$ git clone ssh://git@hostserver:report.git

O non specificare proprio il protocollo — Git utilizza SSH non lo specifichi:

git clone git@hostserver:report.git

Riferimenti

<http://git-scm.com/book/it/I-comandi-interni-di-Git>

<http://digitalworlds.altervista.org/2011010567/Notizie/Git/git-comandi.html>

<http://marklodato.github.io/visual-git-guide/index-it.html>

<http://rogerdudler.github.io/git-guide/index.it.html>

<http://git-scm.com/docs>

<http://git-scm.com/book/it>

<http://blog.mrwebmaster.it/2012/03/13/5-comandi-utili-per-lutilizzo-di-git.html>

<http://criluzone.altervista.org/appunti-git/riferimento-dei-comandi-principali.html>

<http://www.netd.it/sviluppo/guida-git-controllo-di-versione-per-repository-software>

<http://it.gitready.com/beginner/2009/01/10/stashing-your-changes.html>

<https://github.com/progit/progit/tree/master/it>

<https://www.exratione.com/2011/03/setting-up-an-access-controlled-git-repository-server-with-gitolite/>

<https://www.digitalocean.com/community/articles/how-to-use-gitolite-to-control-access-to-a-git-server-on-an-ubuntu-12-04-vps>