ISLT2013 Git - guida all'utilizzo

Indice

1 EGit

EGit è un plugin di Eclipse che permette di utilizzare il sistema di controllo di versione **Git** direttamente da IDE Eclipse. In questa breve guida saranno mostrati alcuni scenari d'utilizzo di EGit per lo sviluppo collaborativo.

1.1 Impostazioni iniziali

1.1.1 Impostazioni Git

Prima di iniziare il lavoro con EGit vanno impostati nome, cognome e la propria email. Questo permette di attribuire ad ogni membro del gruppo i cambiamenti apportati.

Per fare tali configurazioni bisogna selezionare *Window* → *Preferences* → *Team* → *Git* → *Configuration*, tramite pulsante *Add Entries* inserire *Key* user.name con *Value* nome e cognome, e *Key* user.email con *Value* uguale all'email dell'università (figura 1). Tutte le impostazioni saranno salvate nel file di configurazione selezionato nel campo *Location*, rimanendo memorizzate anche per altri progetti.

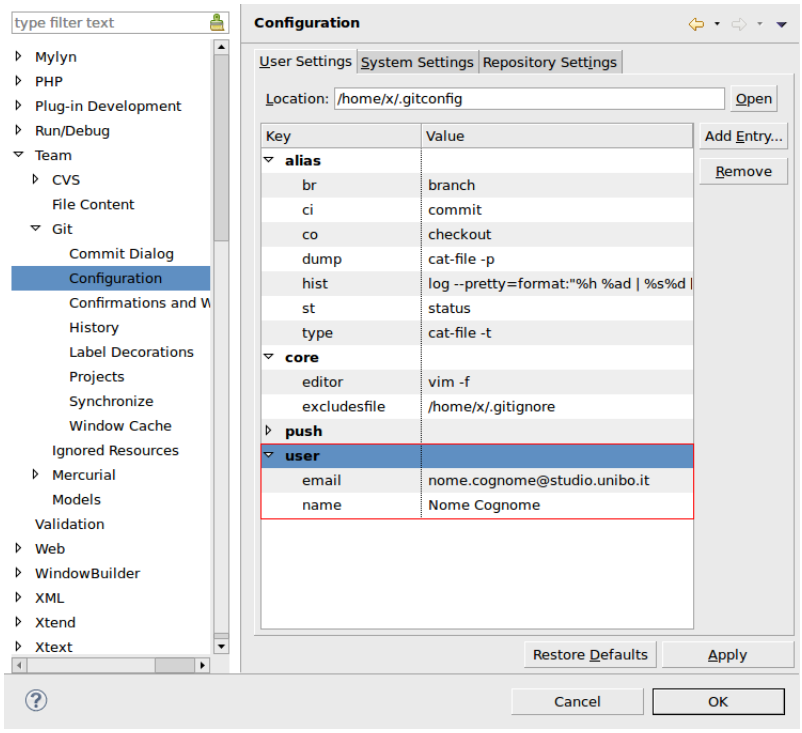

Figura 1: EGit setup

1.1.2 Generazione delle chiavi RSA

L'accesso ai repository sul server avviene tramite protocollo SSH, con metodo di autenticazione basato sulla crittografia asimmetrica. Per accedere l'utente deve prima generare la copia delle chiavi: chiave pubblica e chiave privata. La chiave pubblica deve essere trasferita sul server e la chiave privata deve essere conservata dall'utente.

E' possibile generare la copia delle chiavi tramite Eclipse IDE:

1. Aprire *Window* → *Preferences* → *General* → *Network Connections* → *SSH2*
2. Selezionare la scheda *Key management*
3. Con apposito pulsante *Generate RSA Key ...* generare la copia delle chiavi e salvarla tramite *Save Private Key ...*

1.1.3 Impostazione della chiave privata

Per configurare Eclipse con la propria chiave privata:

1. Aprire *Window* → *Preferences* → *General* → *Network Connections* → *SSH2*
2. Selezionare la scheda *General* (figura 2).
3. Tramite pulsante *Browse* selezionare la directory con la chiave privata *SSH2 home*.
4. Inserire il nome della chiave privata da utilizzare nel campo *Private keys*.

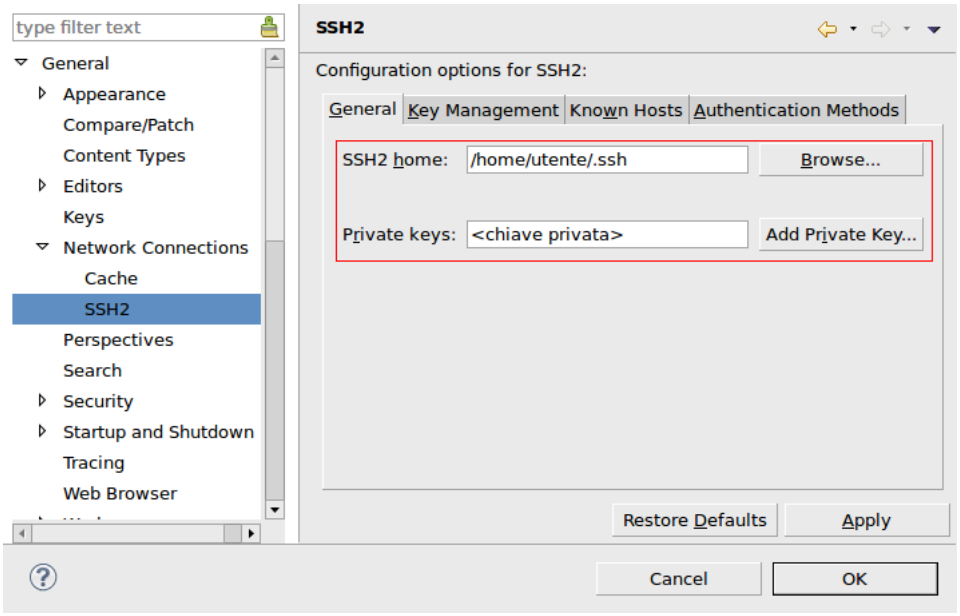

Figura 2: Chiave privata

1.2 Importazione dei contenuti dal repository remoto

Un progetto software può essere iniziato a partire dal materiale disponibile nel repository remoto sul server centrale. Quindi, prima di iniziare la progettazione, il materiale del repository remoto deve essere trasferito nel proprio spazio di lavoro (*git clone*). Per fare questa operazione con Eclipse:

1. Attivare la vista *Window* → *Show view* → *Other* → *Git* → *Git repositoryes* (figura 3)

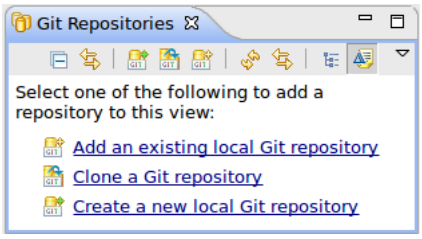

Figura 3: Repository view

2. Selezionare *File* → *Import...* → *Git* → *Projects from Git*.
3. Selezionare *URI*.
4. Inserire l'indirizzo del repository remoto (figura 4).
5. Selezionare tutti i branch di interesse.
6. Tramite il pulsante *Brows* selezionare la posizione in cui verrà creato il repository locale.
7. Selezionare *Import all existing projects after clone finishes*. E' possibile anche importare i progetti in un secondo momento, tramite menù contestuale *Working Directory* su *Repository View* (figura 5).

Alla fine di questo procedimento, nel *Package Explorer* deve essere visibile il progetto importato.

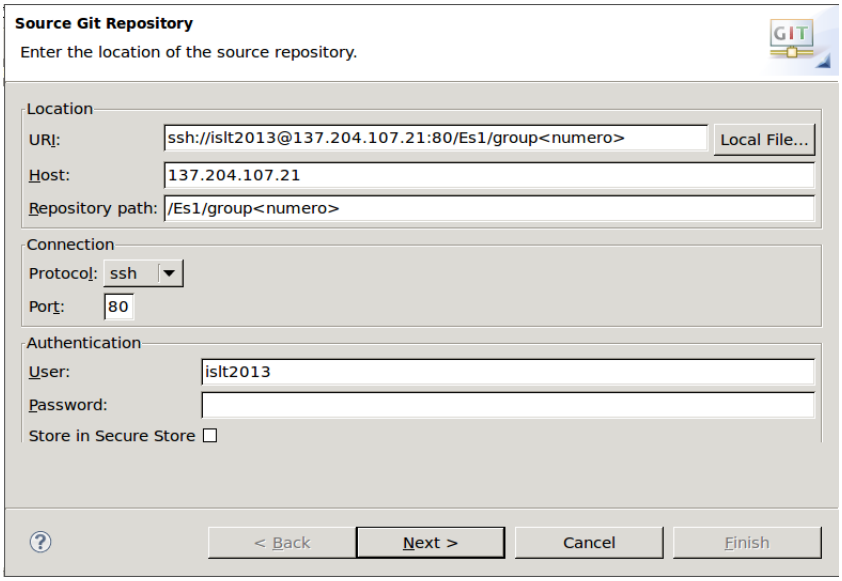

Figura 4: Remote repository address

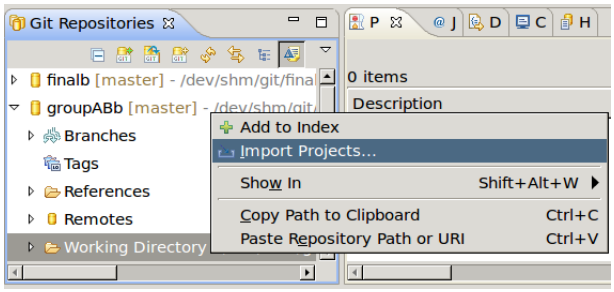

Figura 5: Importazione progetti da repository Git

1.3 Invio delle modifiche al repository remoto

1. Proporre le modifiche (aggiungendole all'Index) usando il menù contestuale dei file modificati: *Team* → *Add to Index*
2. Validare queste modifiche: *Team* → *Commit*
3. Inviare le modifiche al repository remoto: *Team* → *Push to Upstream*

Altri membri del gruppo possono ottenere le modifiche usando: *Team* → *Pull*.

E' una buona prassi eseguire sempre: *Team* → *Pull* prima di inviare i cambiamenti al repository remoto.

1.4 Condividere un nuovo progetto

1. Condividere il progetto usando il menù contestuale: *Team* → *Share Project*
2. Selezionare *Git* come *repository type*
3. Selezionare il repository in cui deve essere aggiunto il nuovo progetto (figura 6).
4. Inviare il progetto al repository remoto come descritto in 1.3.

Di seguito all'operazione *Pull*, altri membri del gruppo devono importare un nuovo progetto tramite il menù contestuale di *Working Directory* su *Repository View* (figura 5).

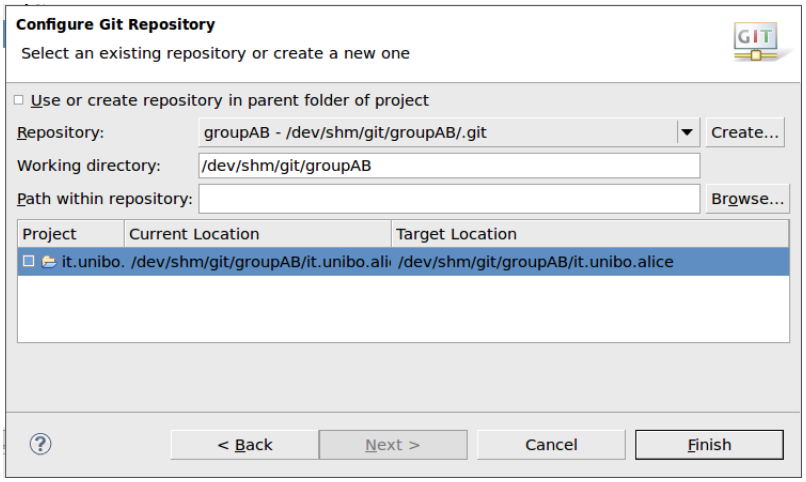

Figura 6: Share Project

2 Collaborazione

Tra i principali vantaggi del sistema di controllo di versione distribuito *Git* vi sono la facilità e la flessibilità di gestione dei progetti sviluppati in modo collaborativo. Di seguito vengono presentate alcune regole di utilizzo del repository centrale per la collaborazione.

2.1 Procedimento di lavoro per la risoluzione di un compito

1. Prima di cominciare a lavorare su un compito bisogna assicurarsi di avere a disposizione l'ultima versione dell branch principale che contiene l'ultima versione "stabile" del progetto (*pull*).
2. Per la realizzazione di un compito viene creato un branch a partire dal branch principale (*master*) del progetto. Il nome del branch deve essere creato in modo da avere un riferimento all'obiettivo da raggiungere. Questo branch può essere condiviso tramite il server centrale (*push*).
3. Ogni modifica apportata viene realizzata all'interno del branch. Per ognuna di essa, una volta completata, deve essere creato un nuovo commit (per esempio un nuovo commit dopo l'aggiunta di un nuovo plugin o funzionalità, un nuovo commit dopo il refactoring o pulizia del codice). Questo facilita le correzioni di errori e refactoring.
4. Quando tutte le funzionalità sono state testate e i commit sono stati fatti per tutte le modifiche, il branch deve essere integrato nel branch principale (*merge* o *rebase*) e caricato sul repository centrale (*push*).
5. A questo punto altri componenti del gruppo possono scaricare i nuovi aggiornamenti (*pull*).
6. I branch aggiuntivi possono essere cancellati in locale e/o in remoto.
7. Il branch principale deve essere caricato sul repository di consegna.

2.2 Raccomandazioni per i commit "puliti"

1. Ogni commit deve avere un autore - Nome Cognome, email dell'università.
2. Il testo dei commit deve contenere la descrizione breve dei cambiamenti.
3. Il commit non deve contenere i file che non corrispondono ai cambiamenti, per esempio i file che vengono creati da strumenti esterni. I file "indesiderati" possono essere aggiunti in *.gitignore*.
4. Il commit non deve aggiungere o rimuovere le righe vuote o gli spazi, ad esclusione dei casi che corrispondono alla logica di commit (refactoring, pulizia del codice).
5. Lo stile dei cambiamenti deve essere mantenuto uniforme. Per esempio se per l'allineamento del codice vengono utilizzati i tab, il codice aggiuntivo deve seguire lo stesso modello.

6. Il codice deve essere scritto in modo da minimizzare i conflitti dovuti ai eventuali modifiche.

3 Link utili

- `http://git-scm.com/book`
- `http://wiki.eclipse.org/EGit/User_Guide`
- `http://pcottle.github.io/learnGitBranching/`