Departamento de Ingeniería Informática y Matemáticas

Guía Rápida de Git

Sistema de Control de Versiones

Grado de Ingeniería Informática

Universitat Rovira i Virgili

Autor: Santiago Romaní (santiago.romani@urv.cat)

Revisión: Pere Millán (pere.millan@urv.cat)

Última actualización: 30-01-2015

Índice de contenido

1 Instalación y configuración de Git

Para instalar el sistema Git en nuestro ordenador (Windows, Linux o MacOSX) hay que descargarse el programa de instalación disponible en <http://git-scm.com/>, seleccionando el sistema operativo correspondiente.

Para introducir comandos Git hay que abrir una sesión de terminal; en entornos Linux o MacOSX, hay que abrir una *shell*; en entornos Windows se puede abrir la consola desde el menú Inicio, opción Ejecutar → comando "cmd", pero nosotros recomendamos el uso del terminal específico del entorno Git, que se llama "Git Bash".

Antes de utilizar el sistema Git en un ordenador concreto, hay que configurar nuestras credenciales de usuario:

```
$ git config --global user.name "Santiago Romani"
$ git config --global user.email santiago.romani@urv.cat
```

Evidentemente, cada persona tiene que escribir su propio nombre y su dirección de correo electrónico.

NOTA: en nuestro ordenador particular, sólo tendremos que realizar esta configuración una única vez; sin embargo, en los ordenadores del laboratorio habrá que configurar el usuario en cada sesión de laboratorio, puesto que el contenido del disco se reinicia cada vez que entramos.

2 Instalación y configuración de OpenSSH

Para poder trabajar en equipo, todos los miembros de un grupo de prácticas tienen que poder publicar su trabajo en un repositorio Git remoto. Para las prácticas de algunas asignaturas (Computadores, ESO, …), los repositorios remotos están ubicados en un servidor gestionado por el Departamento de Ingeniería Informática y Matemáticas (DEIM), de nombre **git.lab.deim**.

Sin embargo, dicho servidor sólo es accesible desde la *intranet* de la URV, es decir, para los ordenadores conectados a la red interna de la universidad. En caso de tener que acceder desde fuera de la universidad, hay que montar un túnel vía protocolo SSH.

En la URL <http://deim.urv.cat/~ajuda.deim/docencia/documents/acces_intranet.php> se explican los pasos para montar cualquier túnel de acceso a los diversos servidores del departamento.

De todos modos, a continuación detallaremos los pasos a seguir para realizar específicamente un túnel con el servidor git.lab.deim:

1. Tener instalado correctamente el paquete OpenSSH (para Windows, se puede descargar desde <sorceforge.net/projects/sshwindows>).

2. Si el primer paso se ha completado con éxito, en el directorio HOME del usuario del ordenador tiene que existir una carpeta con el nombre ".ssh"; en sistemas Windows el directorio HOME suele estar en "C:\Documents and Settings\<Usuario>", donde <Usuario> es el identificador del usuario actual del ordenador. Si no existe la carpeta, se puede crear manualmente.

3. Dentro de esta carpeta hay que editar o crear un fichero de tipo texto con el nombre "config", con el siguiente contenido:

```
host deim1
        hostname portal1-deim.urv.cat
        user <ID usuario>
        localforward 8004 git:22

host tunelgit
        hostname localhost
        user <ID usuario>
        port 8004
        hostkeyalias tunelgit
```

donde <ID usuario> se tiene que sustituir por el identificador de usuario utilizado para identificarse en el acceso a las aplicaciones de la universidad (correo electrónico, *Moodle*, etc.), que normalmente es el NIF (**8 números seguidos de guión y la letra en mayúsculas**).

NOTA: el nombre del fichero "config" no tiene que llevar ninguna extensión tipo ".txt" u otras, para que sea reconocido por el sistema OpenSSH.

4. Desde un terminal, crear una conexión con el servidor **portal1-deim.urv.cat**, que permite acceder a los servidores de la intranet a través de túneles:

```
$ ssh deim1
```

Esta acción pedirá confirmación de aceptación de la "RSA key fingerprint" del servidor **deim1**, en caso de que sea la primera vez que accedemos, y después nuestro *password* de usuario.

A partir de este momento, este terminal ya no lo podemos utilizar para introducir comandos, pero **es imprescindible no cerrarlo**, de lo contrario perderíamos la conexión a través del túnel. Por lo tanto, tendremos que abrir un segundo terminal para continuar ejecutando comandos Git o de sistema.

NOTA: es posible que al ejecutar `ssh deim1` en una consola clásica de Windows (Inicio: Ejecutar → "cmd") el comando no reconozca el identificador de servidor deim1; esto es debido a que el Windows tiene su propio gestor del protocolo SSH, que no utiliza el fichero "config". En cambio, desde un terminal "Git Bash" sí que se invoca a OpenSSH para establecer la comunicación y sí que se utiliza el fichero "config".

3 Clonar el repositorio remoto

Para empezar a trabajar con nuestro proyecto, lo primero que hay que hacer es una copia o clon del repositorio remoto sobre un repositorio local. Esto se consigue ejecutando el siguiente comando desde el directorio donde queremos ubicar el repositorio local:

```
$ git clone <ID usuario>@git.lab.deim:<ID repo>
```

donde <ID usuario> se tiene que sustituir por nuestro identificador (el NIF) e <ID repo> se tiene que sustituir por el identificador del repositorio, por ejemplo, "C_grup_A" o "ESO_grup_F".

En el caso de que tengamos que acceder a través del túnel (desde un ordenador propio), el comando será más sencillo, aunque primero habrá que establecer el túnel con el servidor:

```
$ git clone tunelgit:<ID repo>
```

En este caso no hace falta especificar el identificador de usuario porqué lo hemos introducido en la configuración del túnel (fichero ".ssh/config", "host tunelgit").

Puede que el sistema requiera confirmación de aceptación del "RSA key fingerprint" del servidor git.lab.deim, si es la primera vez que accedemos, además de nuestro *password* de usuario.

Si el comando se ejecuta con éxito, podremos entrar en el directorio (carpeta) que se habrá creado dentro del directorio donde hemos ejecutado el comando:

```
$ cd <ID repo>
```

NOTA 1: en sistemas Windows, se recomienda mover o clonar el directorio que contiene el repositorio en la raíz del disco "C:\", o en un subdirectorio con un nombre simple (por ejemplo "C:\Computadores"), con el fin de evitar conflictos con las rutas de los ficheros que puedan contener espacios en blanco como "C:\Documents and Settings", u otros signos de puntuación (acentos, caracteres regionales, etc.).

NOTA 2: el comando `git clone` **sólo se tiene que realizar una única vez** en nuestro ordenador personal (portátil o de sobremesa), ya que nuestro repositorio local se sincronizará automáticamente con el repositorio remoto al usar los comandos `git push` y `git pull`; sin embargo, en los ordenadores de los laboratorios de la universidad hay que utilizar el comando `git clone` al inicio de cada sesión de laboratorio, puesto que el disco duro se reinicia cada vez que arrancamos el ordenador.

NOTA 3: si trabajamos desde un ordenador portátil propio, **el clonado inicial del repositorio se debe realizar a través del túnel**, aunque tengamos acceso directo al servidor cuando el portátil está conectado a la intranet de la Escuela; de otro modo, cuando lleguemos a casa no podremos utilizar el túnel (aunque esté bien configurado), porque la referencia del repositorio remoto registrada será la del acceso directo al servidor a través de la intranet (se puede consultar invocando el comando `git remote -v`).

4 Secuencia habitual de trabajo con Git

El flujo de trabajo habitual con un sistema de control de versiones Git es, más o menos, como sigue, suponiendo que ya se ha realizado la configuración del usuario y el clonado del repositorio remoto en un repositorio local:

1. Actualización del repositorio local:

```
$ git pull origin <ID rama>
```

donde <ID rama> es el identificador de la rama donde estamos trabajando. Este paso **no** es obligatorio realizarlo al principio de cada sesión, aunque es necesario si sabemos que algún compañero puede haber modificado la rama con la que vamos a trabajar.

2. Modificación de ficheros:

Con el entorno de desarrollo elegido, editaremos los ficheros fuente, compilaremos, ejecutaremos y depuraremos el código del proyecto. Cada vez que obtengamos una versión mejorada del proyecto, será recomendable crear un nuevo *commit*.

Antes de realizar el *commit* es recomendable consultar el estado actual de los ficheros:

```
$ git status
```

3. Realización de commits:

```
$ git commit -a -m "<ID rama>: mensaje descriptivo del commit"
```

El mensaje descriptivo no tiene que ser muy largo (hasta 50 caracteres), pero tiene que servir para indicar cual es la modificación introducida en el *commit*, por ejemplo, "primera versión de la función X" o "arreglado problema Y". Además, en el caso de que el proyecto se desarrolle con diversas ramas, se recomienda añadir el identificador de la rama <ID rama> al principio del mensaje, con el fin de poder realizar más eficazmente el seguimiento del trabajo aportado en cada rama, ya que Git no registra automáticamente esta información.

4. Actualización del repositorio remoto:

```
$ git push origin <ID rama>
```

donde <ID rama> es el identificador de la rama donde estamos trabajando. Este paso es necesario para que los compañeros tengan acceso a los últimos cambios que hayamos realizado. Se puede hacer después de cada *commit*, pero no es imprescindible, ya que el comando `git push` es capaz de subir varios *commits* de una sola vez.

En el caso de que algún compañero hubiera realizado algún cambio en la rama que queremos actualizar, el sistema Git nos obligará a realizar primero un `git pull`, con el fin de realizar una fusión de las dos versiones (*merge*), y después ya podremos invocar de nuevo el comando `git push`.

5 Resumen de comandos Git

Este apartado tiene el propósito de servir como una guía de referencia de los comandos más frecuentes, aunque el documento "**Manual práctico de Git**" los explica en más detalle y con ejemplos de uso.

5.1 Comando de clonación

`git clone <user>@<server>:<repo>` `git clone <tunelserver>:<repo>`	Clona un repositorio de nombre <repo> ubicado en un servidor de nombre <server> al cual tengamos acceso como usuario <user>, o mediante el túnel adecuado <tunelserver>.

5.2 Comandos de información básica

`git status`	Saber el estado actual del repositorio local.
`git log`	Ver la secuencia de *commits* de la rama actual.

5.3 Comandos para añadir o quitar ficheros y directorios

`git add -- <fichero>` `git add .`	Añade un nuevo fichero o directorio al repositorio local; la opción de '.' permite añadir todos los ficheros modificados o nuevos.
`git rm -- <fichero>`	Elimina un fichero o directorio del repositorio local.

5.4 Comando para crear una versión o commit

`git commit -a -m "Mensaje"`	Añade las modificaciones realizadas desde el último *commit* y crea una nueva versión del proyecto.

5.5 Comandos para sincronizar los repositorios

`git push origin <rama>`	Publica los últimos *commits* del repositorio local en el repositorio remoto, en la rama identificada por <rama>.
`git pull origin <rama>`	Recibe las últimas actualizaciones del repositorio remoto en el repositorio local, en la rama identificada por <rama>.
`git fetch`	Descarga el estado actual del repositorio remoto, lo cual permite observar su evolución (con 'gitk', por ejemplo) antes de hacer un 'git pull'.

5.6 Comandos para gestionar ramas

`git checkout <id_rama>`	Cambia la rama actual a <id_rama>, actualizando el *commit* correspondiente de la nueva rama.
`git checkout <id_commit>`	Cambia el *commit* actual a <id_commit>, que se puede especificar con los 7 primeros dígitos de su código identificador SHA1 (visibles en el 'gitk')
`git branch -a`	Muestra todas las ramas, locales y remotas.
`git branch <nombre_rama>` `git checkout -b <nombre_rama>`	Crea una nueva rama desde el *commit* actual (los dos comandos son sinónimos).
`git branch -D <id_rama>`	Borra un identificador de rama, pero sin borrar los *commits* referenciados por dicha rama.
`git merge --no-ff <id_commit>` `git mergetool`	Fusiona el contenido de la cabecera actual con el contenido de otro *commit* <id_commit> (si hay conflictos, se puede usar *mergetool*).

5.7 Comandos para deshacer cambios

ATENCIÓN: sólo se pueden deshacer cambios NO publicados en el servidor.

`git checkout -- <fichero>` `git checkout <id_rama> -- <fichero>`	Recupera el contenido de un fichero o directorio. Recupera contenido desde una rama concreta.
`git merge --abort`	Deshace el último *merge*.
`git commit --amend -m "Mensaje"`	Sustituye el último *commit* por el estado actual de los ficheros, y con un nuevo mensaje descriptivo.
`git reset --hard <id_commit>`	Fija el *commit* especificado <id_commit> como última versión de la rama actual. Si no se especifica *commit*, restaura todos los ficheros.
`git branch -f <id_rama> <id_commit>`	Fuerza el puntero de la rama especificada a que apunte al *commit* especificado.

5.8 Comandos para invocar interfaces gráficas

`gitk --all --date-order &`	Visualiza los *commits* gráficamente, así como las diferencias introducidas en cada *commit*.
`git gui`	Permite invocar los comandos básicos de Git en un entorno gráfico.

5.9 Comandos de ayuda

`git help`	Lista los comandos de Git más habituales.
`git <command> --help`	Ofrece información detallada de cada comando.