**5.1.1. Creando un repositorio de Git**

Para crear un repositorio es tan sencillo como, dentro del directorio que queremos gestionar con git, hacer:

$ git init

Git contestará:

Initialized empty Git repository in .git/

Esto quiere decir que se ha creado el directorio .git/ donde Git guardará toda la información de control. Esto es otra diferencia con los sistemas de control habituales, como Subversion, que nos dejaban un directorio de control en todos los subdirectorios de nuestro proyecto (el típico .svn/). Lo cual es de agradecer porque “ensucia” menos nuestro trabajo.

Si os dais cuanta como Git es distribuido el repositorio por ahora sólo lo tenemos en nuestro local, y por lo tanto, no ha hecho falta tener conexión de red. De esta forma también podemos usar Git para versionar nuestro trabajo de forma muy sencilla, aunque no pensemos compartilo con nadie más.

**5.1.2. Clonando un repositorio de Git**

Lo normal cuando trabajamos con sistemas de control de versiones suele ser “descargarnos” un proyecto de un repositorio existente (normalmente los repositorios no los creamos nosotros como en el punto anterior, sino que nos conectamos a ellos).

Para esto es tan fácil como hacer cosas como:

git clone git://git.kernel.org/pub/scm/git/git.git

git clone http://www.kernel.org/pub/scm/git/git.git

Fijaos como podemos usar un protocolo propio de git (git://) que es más óptimo, o podemos hacerlo por HTTP normal (http://). Otra de las ventajas de git es que podemos hacer el “trasiego” de información de muchísimas maneras, incluso a través de un emil o un pendrive.

**5.2. Trabajo habitual (trabajando en local yo solo) con Git**

Modificaremos fichero, bien porque cambiamos su contenido o bien porque los añadimos nuevos. En ambos casos tendremos que hacer:

$ git add file1 file2 file3

Es importante destacar que el comando add hay que lanzarlo incluso si el fichero ya estaba bajo el “control” de repositorio y sólo hemos cambiado su contenido. Esto es bastante diferente a como suelen trabajar el resto de sistemas de control de versiones. Podríamos decir que el add no significa “añadir al repositorio”, sino que significa “añadir al *index* para el próximo commit”. Es decir, con el add le estamos indicando que es lo que queremos que versione la próxima vez que hagamos commit.

Podemos ver que es lo que tenemos pendiente de hacer commit con:

$ git diff --cached

La opción --cached indica que queremos ver los cambios con respecto a lo que ya tenemos añadido al *index* y que se tendrá en cuenta en el proximo commit. Si no ponemos esta opción nos indicará todas las coas que potencialmente podríamos añadir al*index* con un add.

Otra forma de ver los cambios pendientes es con:

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# modified: file1

# modified: file2

# modified: file3

#

Cuando ya no queramos hacer más cambios y queramos hacer el commit (la versión en el repositorio), haremos:

$ git commit

Para automatizar el comando add, y no tener que hacerlo de cada fichero que hemos modificado, podemos ejecutar:

$ git commit -a

Pero ojo, que sólo se hará el add automáticamente de los ficheros modificados que ya existían en el repositorio; de los ficheros nuevos siempre tendremos que hacer el add.

**5.3. Trabajo distribuido (trabajando en equipo) con Git**

El trabajo distribuido implica que estamos trabajando con más personas, donde cada una tiene su propio repositorio, con sus propios branches, ...

En este entorno empezaremos trabajando con un clone de forma que tengamos nuestra propia copia en local.

A partir de aquí editaremos ficheros y haremos commit tantas veces como queramos. Simplemente hay que tener en cuenta que todos estos commits los estamos haciendo sobre nuestro repositorio local.

La cuestión es como sincronizamos estos cambios que tenemos en nuestro local con el repositorio original del cual hicimos el clone. Pues bien, tenemos dos herramientas básicas:

* git pull - “Tira” de un repositorio los cambios hacia nuestro repositorio. Nos estamos bajando las actualizaciones.
* git push - “Empuja” nuestros cambios hacia otro repositorio. Es como si estuviéramos subiendo hacia ese repositorio los cambios que hemos hecho en local.

Vamos a ver algunos modelos de trabajo. Pero no debéis quedaros sólo con esto porque Git, como el resto de sistemas de control de versiones distribuidos, permite un enorme abanico de nuevos modelos de trabajo, al romper “las cadenas” de tener un único repositorio central.

**5.3.1. Trabajando sólo con git pull**

Imaginemos que tenemos a María y a Carlos. Carlos puede empezar a trabajar a partir del código de María con:

$ git clone ssh://mariahost/home/maria/project

Ahora Carlos hace varios cambios en su repositorio, cuando cree el que el código está listo puede avisar a María para que esta haga un pull del código de Carlos. De esta forma María ejecutará algo como:

$ git pull ssh://carloshost/home/carlos/project

Lo que está haciendo María es “bajarse” el código de Carlos y luego hacer un merge en su propio repositorio. Evidentemente si hay conflictos, tendrá que ser María la que los resuelva.

Si ahora María ha hecho más cambios sobre el código, y Carlos quisiera actualizarse, podría hacer:

$ git pull ssh://mariahost/home/maria/project

De forma que Carlos se estaría bajando el código del repositorio de María y estaría haciendo un merge con su propio repositorio. Vemos así como el pull puede funcionar en ambos sentidos.

Esta forma de trabajar puede estar bien para equipos pequeños y con mucha comunicación. El problema que tiene es que el que hace el pull es el encargado de resolver los conflictos, así que si queremos tener una “copia maestra” del repositorio, el que mantenga esta copia tendrá que resolver los conflictos de todos los participantes.

**5.3.2. Trabajando con repositorios públicos de Git (git pull y git push)**

En este punto vamos a ver otra forma de trabajar que puede ser más conveniente si queremos mantener esa “copia maestra”.

Todo empieza igual que siempre, haciendo un clone. Lo único es que está vez lo podríamos hacer de la “copia maestra”.

$ git clone ssh://mainhost/var/autentia/project

Ahora, igual que antes, iremos haciendo modificaciones y commits sobre nuestro repositorio local. Llegará un momento en el cual querremos “subir” nuestros cambios. Para ello podemos hacer:

$ git push ssh://mainhost/var/autentia/project master

Con esto estamos subiendo todos los cambios de nuestro repositorio local al repositorio remoto. “master” indica que los cambios se subirán al branch “master” que es el branch por defecto.

Podría pasar que no pudiéramos hacer el push directamente porque el código del repositorio remoto a cambiado. En este caso somos nosotros mismos los que nos tendremos que encargar de resolver los conflictos, normalmente haciendo primero un pullpara actualizarnos, resolviendo los conflictos en local, y luego volviendo ha hacer un push para volver a subir los cambios.