Git: sistema de control de versiones distribuido

GLOSARIO:

**commit**: confirmar, cometer. Cada commit es un “snapshot” (¿exactamente?)

**origin**: es el nombre por defecto que Git asigna al servidor desde el cual se ha clonado un repositorio, cuando se obtiene con la orden “clone”. Para utilizar otro nombre, el comando es git remote add <nombre> <url>

**blob**: viene de Binary Large Objects. En Git se utiliza para referirse a los objetos (ficheros) individuales que se guardan en el repositorio con un “commit” (con sus “checksums”). Luego, el propio commit (el snapshot) guarda un puntero a un “tree” con su “checksum” que a su vez guarda los blobs (punteros a ellos):



**HEAD:** en Git representa una cosa totalmente distinta de otros controladores de versiones (SVN o CVS); es un puntero a la rama local sobre la que estás trabajando.

--staged y --cached son sinónimos

Otros VCS (Version Control Systems) manejan el control de cambios como un conjunto de archivos y los cambios que se han realizado en ellos a lo largo del tiempo (delta-based version control).

En Git, el sistema se compone de una serie de “instantáneas” del proyecto completo, creadas cada vez que se actualiza o se guarda el proyecto; así, en local tendremos el proyecto actual con toda su historia de cambios. Se podría decir que un proyecto gestionado con Git es un mini-sistema de ficheros, más que un simple control de versiones.

La mayoría de las operaciones en Git se realizan sobre los ficheros y recursos locales, dado que cada equipo tiene el proyecto completo. Esto tiene la ventaja de que se puede trabajar “offline” sin problemas, en incluso realizar “commits”, naturalmente en tu copia local.

Todos los recursos en Git tienen asociada una suma de verificación (checksum), de manera que es imposible que haya corrupción de archivos en su transmisión por red, o por cualquier otro motivo, sin que Git avise del problema. El sistema usado para la “checksum” es SHA-1.

Cualquier operación en Git añade datos a su sistema de archivos, en lugar de borrar (ver).

Hay tres posibles estados para un fichero o recurso en Git:

- **Unmodified**: simplemente está ahí, pero sin control por parte de Git (untracked)

- **Commited**: el dato está guardado en la base de datos local (el directorio .git)

- **Modified**: el dato se ha cambiado pero no se ha hecho “commit” aún. Está en el “working tree” o “working directory”. En este caso los ficheros están “tracked” (Git realiza su seguimiento)

- **Staged**: el dato modificado se ha marcado en su versión actual para que vaya en la siguiente “snapshot”. Se almacena en el llamado “staging-area” (un fichero que contiene un índice, en realidad), que es el paso previo a hacer el “commit” (aunque se puede saltar, si fuera necesario)

**Áreas de Git**



**Posibles estados de un archivo en Git**



**COMANDOS:**

git config [opciones]:

git config --global <nombre\_variable> <valor> -> Asigna un valor

git config --global <nombre\_variable> -> Muestra el valor

git config --show-origin <nombre\_variable> -> Informa de dónde está declarada

git status [-s, --short]

git diff

git init

git clone <url> → Añade implícitamente el “origin remote” y utiliza la “master branch” (o la que haya por defecto)

git remote [-v] [add] [show <remote>] [rename <old> <new>] [remove]

git reset -> Quita ficheros del “staging area” (y muchas otras cosas que se verán más adelante). Para eliminar cambios que ya se han guardado en el repositorio local (se ha hecho “commit”), hay dos manera: git reset --hard HEAD~1 o bien con --soft (elimina y no elimina el snapshot respectivamente).

git checkout -- <file> -> Deshace los cambios sobre ese fichero en el área de trabajo (Working directory). Si se hace sobre el nombre de una rama, hace que HEAD apunte a esa rama (y sea la rama actual de trabajo). Con la opción -b (de branch) crea la rama y apunta hacia ella, ahorrándose un paso.

git add <file> -> Tiene varias funciones. Pone el fichero bajo seguimiento (en “tracking”); lo registra en el “Staging Area” y marca los conflictos como resueltos

git fetch -> Obtiene los datos de los repositorios remotos (los que no tenemos o son diferentes. Únicamente los descarga; no hace ningún merge)

git pull -> Es la combinación de un “git fetch” y un “git merge”

git merge

git commit [-a (se salta el paso de “staging”)] [--amend (sustituye el commit anterior por el que se haga con el --amend, reemplazándolo, de manera que no hay dos commits sino uno)]

git push <remote> <branch>

git log [-p, --patch (muestra la diferencia incluída en cada commit)] [-2 (o cualquier número, que indicará las entradas que se quieren ver)] [--stat (muestra información adicional)]

git tag [-l, --list <pattern-opcional>] [-a (de annotated)] <versión> [-m] <comentario> [checksum-part]

git branch <nombre\_rama> -> crea una nueva rama (ojo, no apunta el HEAD a ella automáticamente) a partir de la rama actual de trabajo. git branch –d <rama> borra la rama que se le indique. Sin argumentos, muestra las ramas que existen actualmente.

**Eliminar ficheros:**

Para eliminar ficheros hay que borrarlos del “staging area” (git rm) y después hacer un “commit”. git rm --cached <file> lo eliminará del “staging area”.

**Sintaxis .gitignore**

**Trabajar con repositorios remotos**

**Tagging**

Dos tipos de tags: “lightweight” o “annotated”. Los ligeros son simplemente una marca sobre un commit determinado (como una rama que no cambia).

Los tags anotados, en cambio, se almacenan como objetos completos en la base de datos de Git (¿?). Tienen su “checksum”, guardan el nombre de quien los ha creado y otro datos, y pueden ser firmados y verificados digitalmente.

Los tags anotados deben llevar un comentario.

Se puede crear un “tag” a partir de cualquier “commit”, no necesariamente del último.

El comando “push” no transfiere por defecto los tags que se hayan creado. Hay que especificarlo (git push origin <tagname>, o con la opción --tags para subir más de uno)

Ver lo del “checkout”, que no está claro.

**Git aliases**

En el fichero de configuración se pueden definir alias de comandos (y también combinaciones):

git config --global alias.co checkout (por ejemplo)

**BRANCHING**

En git, la rama por defecto es “master”.

Cuando Git almacena un “commit”, lo hace de esta forma: 

 Una serie de “commits” (snapshots) de una rama, cada uno enlazado a su padre (parent)

En Git, una rama es simplemente un puntero que se puede ir moviendo entre los diferentes commits:



**Merging**

**Fast-forward:**

1 – Cambios en hotfix.

2 – Al estar la rama “hotfix” por delante de “master” (y ésta no haber evolucionado independientemente ), el merge consiste simplemente en mover el puntero “master” a “hotfix”. Es lo que se llama un fast-forward.

**Basic merging:**

Git utiliza un “merge” de tres vías (three way merge)

Voy por <https://git-scm.com/book/en/v2/Git-Basics-Tagging> - para ver

<https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell>

<https://git-scm.com/book/en/v2/Git-Branching-Branching-Workflows>