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

--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”

git reset -> Quita ficheros del “staging area” (y muchas otras cosas que se verán más adelante)

git checkout -- <file> -> Deshace los cambios sobre ese fichero en el área de trabajo (Working directory)

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 -> Comprueba el estado con respecto a “remote” (creo que también descarga los cambios)

git merge

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

**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**

Voy por <https://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes>