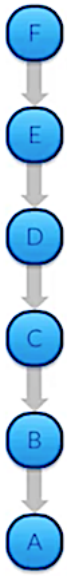
GIT

diff y patch

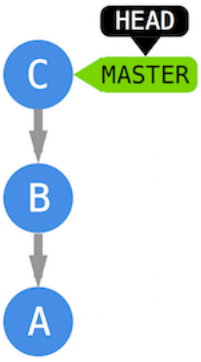
El comando **diff** sirve para encontrar diferencias entre archivos. El resultado es una serie de instrucciones que sirven para convertir un archivo en otro.

El comando **patch** actualiza la información basándose en la información de un *pachFile*, archivo de instrucciones oculto generado con **diff**.

En GIT, con **diff** guardamos las diferencias para convertir la una versión del proyecto en la inmediatamente anterior. Esto se hará ejecutando los *patchFiles*. Por tanto, no se guardan copias enteras del proyecto sino que se crean archivos enlazados virtualmente.



GIT gestiona nuestro **Working copy** con enlaces simbólicos o accesos directos. Es un sistema rápido porque tiene referencias a archivos, en lugar de archivos. Por esta razón tampoco se crean ni se destruyen archivos.



¿Para que sirve GIT?

- Para controlar la evolución del proyecto.
- Para poder trabajar en proyectos de modo colaborativo.
- Para desarrollar funcionalidades en paralelo.
- Para la estructuración y mantenimiento de versiones.

Las tres zonas de GIT:

1. Working Copy:

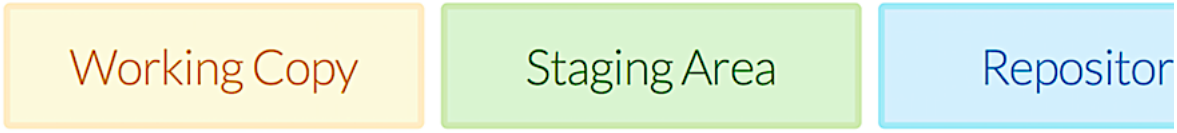
La carpeta donde trabajamos. Es donde se encuentran nuestros los archivos.

2. Staging Area:

Aquí pondremos los archivos a traquear. Para guardar versiones.

3. Repository:

Donde se almacena toda la información de los cambios.



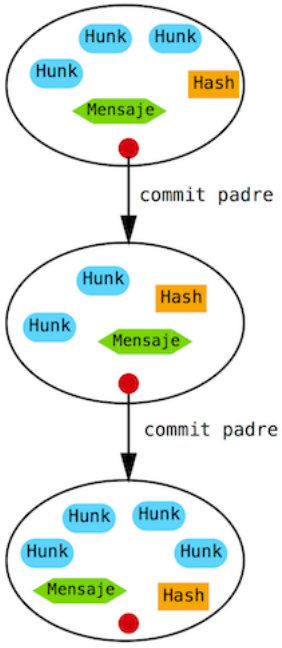
- Nosotros solo veremos el Working copy. Para ver las otras zonas necesitaremos los comandos de Git (las gafas de Git).

Los commits

Sirven para pasar archivos del **Staging Area** al **Repository**.

Un commit es un paquete que contiene:

- Uno o mas hunks.
- Un mensaje que describe los cambios.
- Un identificador del commit.
- Un enlace con su commit padre.



GIT y la consola de comandos

pwd —> Para saber donde nos encontramos

clear —> Limpia la pantalla

ls —> Muestra el contenido de un directorio

ls -l ~/ —> Muestra lo que hay en el *Home* de usuario

ls -a —> Muestra el contenido de un directorio con los archivos ocultos

mkdir *directorio* —> Se crea un directorio

cd *directorio* —> Cambia de directorio de trabajo

cd . . —> Cambia al directorio anterior

rm *archivo* —> Borra archivo

rm *directorio* —> Borra directorio

nano *archivo* —> Se abre un nuevo archivo en el editor “nano”

git —version —> Información sobre la versión de git

git init —> Para crear un repositorio en un directorio

rm -rf .git —> Para que un directorio borre su repositorio

git status —> Para ver el working copy y el staging area

git add *file* > Mueve un archivo del working copy al staging area

git commit —> Creamos un commit

git log —> Para ver los commits

git branch —> Para ver las ramas

git branch *rama* —> Crear una nueva rama

git branch -m *nombreActual nuevoNombre* —> Cambiar el nombre de una rama.

git branch -D —> Elimina rama. Desaparece el puntero pero no se pierde información (no se borran clos commits).

git checkout *rama* —> Situar el HEAD en esa rama

git checkout *identificador* —> Situar el HEAD en un commit

git merge *rama* —> Fusionar dos ramas

git reflog —> Para ver el historial de los commits

git diff HEAD —> Compara el Working copy con el Repositorio.

git diff —> Compara el Working copy con el Staged

git diff --staged —> Compara el Staged con el Repositorio.

git checkout -- *archivo* —> Si no quiero guardar los cambios que he hecho en el archivo para dejarlo como estabe en el repositorio

git reset HEAD~1 —> Deshace el último commit pero mantiene lo que había en el Working copy

git reset --hard HEAD~1 —> Deshace el commit y lo que hay en el Working copy

git reset HEAD —> Saca un archivo del Stagin area

git tag *nombre* —> Etiqueta para identificar un commit

git tag -d —> Borrar una etiqueta

git tag —> Listar los tags

git show version *nombre*—> Puedo ver en que commit está cada tag

git reset --hard *identificador* —> Lleva el puntero master y HEAD a donde nos estamos moviendo y así modifica el Working copy

Descargas

Git: http://www.git-scm.com

SourceTree: https://www.sourcetreeapp.com

MacDown: http://macdown.uranusjr.com