**GIT**

Una vez que generamos en nuestro equipo una carpeta de GIT, es decir, nuestro *workspace* (WS), aparecerá una carpeta de nombre “.git”. En esta carpeta se almacena toda la información de control de versiones. En ningún caso eliminar esta carpeta, porque perderíamos todo nuestro historial de versiones.

Dentro de esta carpeta podremos encontrar todos los commits del proyecto en la carpeta “objects”, en la dirección “refs/remotes/origin” cada una de las referencias o ramas del proyecto, y en “heads” aquellas ramas en las que trabajo o he trabajado en algún momento.

Ahora bien, hay que diferenciar nuestro sistema en cinco partes:

Nuestra área de trabajo es el verde, siendo el WS donde tenemos nuestros ficheros de trabajo y el Index donde subimos nuestro esquema de trabajo, es decir, un índice de los archivos que queramos tener en cuenta para el repositorio más adelante.

Por otro lado, son ambos repositorios, el local (propio) y el remoto, donde mediante commits tendremos un control de las versiones que hagamos sobre el proyecto.

Por último, el caso de Stash es donde se trabaja al margen del workspace. De esta manera siempre tendremos esta sección como escondida respecto al proyecto, pudiendo almacenar y recuperar archivos de aquí independientemente del commit en el que estemos trabajando.

**Obtener información**

Para obtener información de un repositorio global se puede hacer de tres maneras. Primero realizando un **clone**, lo que nos “descarga” una instancia de un repositorio remoto, es decir, se obtiene la información de este repositorio en el ordenador.

Con **fetch** actualizamos la carpeta .git respecto al repositorio global, pero no descarga ningún archivo para trabajar sobre él en nuestro WS.

En cambio, con **pull** equivale a hacer de golpe un fetch y modificar los archivos que ya tenía, como si hiciera consecutivamente un merge.

**Conflictos**

En la mayoría de los casos cuando haya conflictos deberemos abortar y analizarlo de manera independiente, ya que si es un archivo de notas lo va a romper (esto quiere decir que va a poner una línea en algún lugar del documento donde indicará los cambios, pero no los arreglará), y si es un archivo de bits, como será la mayor parte de los casos, como pudiera ser un archivo de Simulink, git no podrá “ver” qué ha cambiado, y hay que tener mucho cuidado porque si se nos ocurre abrir uno de los dos archivos en conflicto durante el proceso, git entenderá que el que hayamos abierto ha sido modificado y se cargará el otro.

Por este motivo es preferible hacer FETCH + MERGE que simplemente un PULL, porque con fetch vemos los conflictos y decidimos antes de hacer un merge, en cambio con pull nos saltamos el primer paso.

**Comparativa**

Con diff comparamos nuestro repositorio local con otro elemento. En este caso, siempre se utiliza el término *origin* para indicar el repositorio remoto global, en cambio los términos como *master* pueden cambiar en función de lo que nos interese.

**RESET**

Diferenciamos tres tipos de RESET:

* Soft: Modificamos el repositorio
* Normal: Modificamos el repositorio y el Index.
* Hard: Modificamos el repositorio, el Index y el WS.

RESETET

En el caso de haber hecho ya un PUSH (subida de tu repositorio al respositorio global) no hacer un RESET, ya que, si retrocedemos en nuestro repositorio y nos ponemos a trabajar de nuevo, cuando volvamos a hacer un push tendremos conflictos asegurados.

**REVERT**

Con REVERT estaríamos haciendo algo muy parecido a RESET, pero sin perder los commits B y C, es decir, genera un COMMIT nuevo que vuelve a tener las condiciones de A, por lo que revierte las acciones hechas en B y C.

REVERT

Con revert –n solo faltaría hacer el COMMIT.

Ahora bien, en relación al *reset* y al *revert*, un *reset* puede deshacer los cambios no *commiteados*, en cambio con *revert* no es posible. Con esto se quiere decir:

Si estamos trabajando en (3), realizamos varios cambios y no los *commiteamos* podemos volver al (3) inicial con un reset.

**CHECKOUT**

Con *checkout* nos movemos entre distintos *commits* que pueden estar en la misma, o en otra, rama.

Ejemplo 1: Pasamos de C a E y si hacemos algún cambio, modificamos E.

Ejemplo 2: Pasamos de C a B, y si queremos hacer algún cambio tenemos que crear una rama nueva que parta de B y vaya en paralelo a la rama B-C.

Ahora bien, podemos traer solo un archivo de un commit cualquiera. Para ello tenemos que utilizar la siguiente estructura:

git checkout *“ID del commit deseado”* --./miArchivoElegido.doc

git checkout especify files

**STASH**

Por último, como hemos comentado anteriormente, si queremos guardar un archivo sin que entre de momento en el control de versiones, utilizamos STASH. Para almacenar archivos ahí tendremos que usar:

git stash save

En cambio, para recuperarlos al WS (ojo, no los commitea con este paso) podemos utilizar la siguiente línea:

git stash apply

**GITK**

Con gitk te permite ver mediante una interfaz cómo se encuentra tu repositorio local de una manera más visual.