**DEFINICION**

Git es un sistema de control de versiones, esto significa que nos permite almacenar archivos conforme va pasando el tiempo, permitiéndonos volver a estos archivos (que son versiones anteriores de los que usamos en el presente) en cualquier momento. Esto nos permite volver a un punto anterior de un código cuya funcionalidad queremos recuperar, probar nuevas ideas sin causar problemas en el código principal y colaborar con más programadores de manera más sencilla.

Por otro lado Github es un servicio online que hostea nuestros proyectos y nos permite colaborar de manera sencilla con otros programadores que pueden descargar nuestros trabajos, editarlos y proponerlos a votación para añadir las modificaciones al código principal.

**DESCARGA**

Para descargar git iremos a la página <https://git-scm.com/downloads> y pulsaremos en la opción correspondiente a nuestro sistema operativo.

**REPOSITORIOS**

Un repositorio es un contenedor para almacenar un proyecto cuyo desarrollo queremos seguir.

Podemos tener tantos repositorios como queramos para tantos proyectos como queramos.

**COMMITS**

Son puntos de guardado de nuestros proyectos.

Para realizarlos nuestros archivos pasarán por una serie de estados:

1. Al realizar cambios en el archivo este se encontrará en una fase de “modificación”.

2. Deberemos entonces guardar estos archivos en el área donde se encuentran los archivos modificados listos para guardar en el “historial de commits”.

3. Realizamos el commit y estos archivos modificados pasan a formar parte del susodicho historial.

**INICIAR REPOSITORIO**

Una vez sabemos esto, empecemos a ver cómo funciona git en la práctica.

Para comenzar accedemos al archivo en el que queramos empezar nuestro nuevo repositorio (todo esto en la consola, que es desde donde trabajaremos con git) y escribimos **git init**.

**ALMACENAR LOS FICHEROS MOFICIADOS EN LA ZONA DE PREPARACIÓN PARA COMMIT**

Antes de realizar un commit y añadir un fichero al historial de commits hay que “prepararlo” por decirlo de algún modo. Para realizar esta acción una vez estemos en un fichero que actúe como repositorio utilizaremos el comando **git status** que nos mostrará por pantalla que archivos del repositorio han sido modificados. Para añadir los archivos que veamos mostrados utilizaremos el comando **git add + ‘filename’**. Tras esto si volvemos a escribir **git status** veremos que se sigue mostrando el archivo pero de una forma diferente, de esta forma sabremos que ha sido añadido correctamente a la zona de preparación. Si deseamos deshacer esto, al habernos equivocado de fichero, usaremos el comando **git rm --cached + ‘filename’** lo cual retornará el archivo a la fase de ‘modificado’. En caso de que queramos añadir o eliminar varios ficheros de esta zona utilizaremos **git add .** y **git rm --cached -r .** respectivamente.

**REALIZAR LOS COMMITS**

Para realizar los commits de aquellos archivos que ya están preparados para ello utilizamos el comando **git commit -m + “”** que nos permite no solo añadir este fichero con cambios al historial de commits sino que además nos permite añadir un mensaje para que al revisar los commits realizados sepamos para que era cada uno.

**REVISAR LOS COMMITS**

Revisar los commits realizados es tan fácil como escribir **git log** que nos los mostrará con el mensaje con el que los guardamos. Para ver estos commits de una forma más resumida podemos utilizar **git log --oneline** lo que nos permitirá verlos de una forma mucho más clara.

**ROLLBACKS (VOLVER ATRÁS)**

En git, tenemos diferentes formas de volver atrás en nuestro historial de commits dependiendo de si pretendemos revisar como era el código en ese momento, ,eliminar uno de los commits o deshacer todos los commits hasta cierto punto, ahora veremos estos tres aspectos:

1. Para revisar como era el código en un momento anterior mediante la id del commit en cuestión escribimos **git checkout + ‘id’** , para regresar al código actual usaremos **git checkout + master (la rama que usamos)**, hablaremos de ramas más tarde.

2. Para “eliminar” uno commit lo que haremos será escribir el comando **git revert + ‘id’**, pero esto no eliminará realmente el commit sino que creará un nuevo commit en el que ese commit no tiene ningún efecto, por lo que podríamos volver al commit en cuestión porque sigue en el historial.

3. Para realmente eliminar todos los commits que queramos desde el punto actual hasta uno en concreto el comando a usar será **git reset + ‘id’**, hay que tener mucho cuidado con este comando ya que va a eliminar los commits, pudiendo causar problemas si no es una acción deseada. En caso de que tengamos 4 commits y usemos git reset + commit1, en el historial de commits solo veriamos ese primer commit pero en nuestro código actual seguiria estando el código que teníamos antes de usar el git reset. Para que este código vuelva en el tiempo también, no dejando rastro de los commits que queríamos eliminar se usará **git reset + ‘id’ + --hard**, lo cual como hemos dicho no solo eliminará los commits del historial sino que nuestro código volverá al punto del commit deseado.

**BRANCHES (RAMAS)**

En git, una de las cosas más importantes que tenemos que conocer es el funcionamiento de las ramas. Hasta ahora hemos trabajado siempre en la rama principal o master, pero esto no es lo correcto. La rama principal es la parte que se supone estable dentro de un proyecto, por lo que a la hora de realizar cambios, pruebas o comprobaciones de la que no estamos 100% seguros debemos recurrir a una zona diferente para estas pruebas. Para ello creamos una rama nueva a la que podemos llamar por ejemplo “implementación de x” en la que podemos realizar commits y probar todo lo que queramos sin afectar a la rama principal.

**CREAR NUEVA RAMA Y MOVERNOS ENTRE ELLAS**

Para poder crear una nueva rama utilizaremos el comando **git branch + ‘branchname’**. Esto creará la rama pero seguiremos estando en este caso en la rama master. Para comprobar cuantas ramas hay y que nombre tienen podemos usar el comando **git branch -a**. Una vez hemos creado la rama querremos movernos a ella para empezar a trabajar, para ello utilizaremos **git chechout + ‘branchname’**, lo que nos mostrará el mensaje “switched to branch ‘branchname’”, significando que estamos en la nueva rama. En el caso de que queramos eliminar una rama que hemos creado utilizaremos **git branch -D + ‘branchname’** si la rama aún no está fusionada con la principal, o **git branch -d + ‘branchname’** en caso de que lo esté, veremos que es fusionar una rama más adelante.

**FUSIONAR RAMAS**

Una vez hemos comprobado que nuestra nueva implementación o código que queremos añadir a la rama principal funciona correctamente tenemos que fusionar ambas ramas, para ello nos movemos a la rama master (donde queremos que aparezca el contenido de la rama secundaria) y usamos el comando **git merge + ‘branchname’**.

**CONFLICTOS**