**GITHUB**

**TERMINOLOGIA Y FUNCIONALIDAD**

**Repository (Git)**

Para iniciar el control de versiones sobre código que tenga alojado en cualquier localización, tengo que crear previamente un *Repository*. El *Repository*  lo iniciaré en la carpeta raíz que contiene los archivos que voy a gestionar. Al iniciar un repositorio se genera una carpeta en la raíz  **.git** que contendrá toda la información, actualizaciones y demás eventos que vayan sucediendo dutante la vida del repositorio.

**Remote Repository (Git)**

Un *Remote Repository* es lo mismo que uno local, pero como la misma palabra indica, se aloja en remoto. Estos repositorios serán normalmente los alojados en Github (O en cualquier otro sistema que use git de manera remota). Serán los repositorios que descargaremos (*Pull*)  y clonaremos (*Clone*) para trabajar de manera local. Posteriormente subiremos a ellos (*Push*) los cambios que hayamos realizado de manera local.

**Commit (Git)**

Un *Commit*  es un conjunto de cambios que se hayan realizado sobre el código de una *Branch* de un *Repository*. Cuando hago un *Commit* se crea una nueva version del codigo fuente de la *Branch* de mi *Repository.* Por tanto, en mis primeros pasos con git, las primeras cosas que haré seran modificar archivos de código y generar *Commits* que versionen mi *Branch* en un momento dado. Son como fotos en un instante del desarrollo. Añado una funcionalidad ... hago *Commit*,  resuelvo un incidencia en el código ... hago *Commit*.  A cada *Commit* le añadiré una descripción del conjunto de cambios realizados.

**Cherry Pick Commit (Git)**

Este término se refiere a la habilidad de git para repetir un *Commit* ya realizado con anterioridad.

**Revert Commit (Git)**

Un *Commit* no es posible eliminarlo. Si quiero deshacer un *Commit,* es necesario realizar un *Revert Commit,* que hará un nuevo *Commit* quitando los cambios realizados por el primer *Commit*.

**Stage (Git)**

Antes de hacer un *Commit* de una serie de modificaciones, he de poner los archivos modificados en *Stage*.  Es como una fase intermedia.

**Stash (Git)**

Al pincipio lo confundía con  *Stage ...* a veces, algunas de las acciones que me permite hacer git requieren que no existan modificaciones en el directorio de trabajo. Si no quiero perder estas modificaciones puedo hacer un *Stash* de los ficheros modificados*.* Una vez completada la acción que requería el directorio de trabajo sin modificaciones , puedo quitar los ficheros del *Stash* y devolverlos al directorio de trabajo. Una situación típica es cuando estoy trabajando en algo, me llega una incidencia que he de resolver de inmediato para posteriormente seguir trabajando sobre lo que estaba. Sin duda querré poner mis cambios en *Stash*. Puedo acumular varios *Stash,* y posteriormente deshacerlos.

**Branch (Git)**

La estrella de git. Las *Branch* se usan para separar grupos de *Commits.* Es muy común crear y eliminar *Branch* con frecuencia y al mismo tiempo mantener otras *Branch* en el tiempo. Son al fin y al cabo, punteros a un *Commit* concreto.

Por ejemplo. Si estoy desarrollando un Sitio Web, un uso muy común es mantener un *Branch* **master** y otro *Branch* **develop,** para distinguir el código estable de la aplicación del código que está en fase de desarrollo o que necesita ser testeado en más profundidad. Estos serían *Branch* de larga duración. Al mismo tiempo los desarrolladores que mantienen o crean nuevas funcionalidades en la aplicación crearán otros *Branches* para resolver una incidencia en concreto, para desarrollar una nueva funcionalidad o simplemente para probar ideas que se vayan ocurriendo. Cuando acabo de trabajar con el *Branch* podré fusionar los cambios (*Merge*) o descartarlos (Borrar el *Branch)*.

**Tag (Git)**

Un *Tag* se usa para marcar un número de revisión concreta. Si quiero modificar un *Tag* primero he de eliminarlo y después crearlo otra vez. Normalmente se usará para marcar la version de mi codigo (1.0.0, 1.01, 1.1.0, 2.0.0 …etc).  El *Tag es* una foto de mi codigo en un momento del tiempo, pero no es lo mismo que una *Branch*, es simplemente una etiqueta con la que marcar una versión en un momento del tiempo.

**Fork (Github)**

Este es una de las palabras que más se escuchan cuando se habla de Github. Hay multitud de páginas de desarrolladores en las que podemos ver que nos piden que haganos un *Fork* en uno de sus proyectos ("Fork me in Github") ... pero qué es hacer un *Fork* realmente. Cuando pulso el boton de *Fork* en un proyecto de Github lo que estoy haciendo es generar un repositorio entre mis repositorios personales que clona el repositorio de origen sobre el que he hecho el *Fork*. A partir de este clon que he generado ya puedo libremente hacer un *Clone* del mismo y trabajar sobre el como lo haria con cualquier repositorio propio.

**Clone (Git)**

Cuando hago un *Clone* de un *Repository* estoy descargando el código de ese *Repository* a una localización indicada para trabajar con él, normalmente de manera local. Como comentábamos anteriormente, para poder hacer esto con un repositorio de otra persona en Github, primero he de hacer un *Fork* para generar un *Repository* personal propio. Una vez dado este paso, haré un *Clone* localmente para comenzar a trabajar con el código.

**Push (Git)**

Hacer *Push* quiere decir que voy a integrar los cambios que haya realizado en un repositorio local con un repositorio remoto. Antes de hacer *Push* debo hacer un *Commit* o bien *Stash* de las modificaciones que haya hecho en mi repositorio local. Esto actualiza el repositorio remoto con el estado del repositorio local.

**Pull (Git)**

Este comando es el contrario de *Push* y lo que hace es  descargar los cambios que se hayan realizado en el  repositorio remoto sobre un repositorio local. Antes de ejecutar esta accion, las modificaciones del repositorio local han de estar *Commit* o *Stash.*

**Merge (Git)**

Un *Merge* fusiona los contenidos del *Branch* de origen con el *Branch* en el que estoy situado. Esto provoca que se añadan en mi *Branch* actual los ultimos *Commits* del *Branch* de origen, pero no elimina mis cambios actuales a no ser que haya modificado los mismos ficheros que en el origen o que haya ficheros borrados en un lado que no lo están en el otro. En estos dos últimos caso se provocaria un *Merge Conflict* que tendriamos que resolver manualmente quedándonos con nuestro archivo o bien con el de origen.

**Rebase (Git)**

*Rebase* es un comando muy similar a *Merge.* De hecho sirve para lo mismo. La diferencia está en que *Rebase* trata de mantener la historia de un *Branch* de manera lineal, mientras que *Merge* genera ramificaciones en dicha historia

**Patch (Git)**

A veces no podemos hacer directamente *Push* sobre un repositorio remoto porque podemos no tener privilegios suficientes. Para solucionar esto se puede crear un *Parch.* Un *Patch* es un conjunto de cambios, es decir, un *Commit (o varios),* que se puedenadjuntar de manera sencilla y enviar por ejemplo en un email. Posteriormente, el receptor de este *Patch,* puede aplicarlo el mismo sobre el repositorio y así hacer un *Merge* sobre el mismo.

**Watch (Github)**

Cuando pulsamos este botón que podemos encontrar en cualquier repositorio de Github, a partir de ese momento, somos un watcher más de ese proyecto y recibiremos información acerca de cambios, actualizaciones, etc. Cuando hago un *Fork* a un proyecto, paso automáticamente a ser un watcher del mismo.

**Pull Request (Github)**

En algún momento podemos querer contribuir a algún proyecto en concreto que nos interese en Github. Para poder hacerlo hemos de crear un Pull Request que informe al usuario dueño del proyecto de nuestra intención. No es más que una petición para hacer un *Pull*.

**Follow (Github)**

Esto en Github tiene el mismo significado que en cualquier otra red social. Puedo convertirme en seguidor de un usuario de Github y así ver y que se me envíen notificaciones del trabajo que está realizando.