***Profundización en GIT/GITHUB***

Cuando uno inicializa GIT (con el comando git init), suceden dos cosas especialmente:

1. Se crea un area en memoria RAM que se llama *“Staging”,* justamente por estar situada en la memoria RAM a esta area también se le conoce como: “area temporal”. Básicamente, es un area completamente desconectada que funciona en memoria RAM y es donde, al principio, vas a ir agregando los cambios que vayas haciendo   
(de tu carpeta), cambios que todavía no han sido guardados en una copia de seguridad (es decir, que todavía no están en repositorio local); importante esto, tan sólo se les está haciendo un seguimiento en un area *de paso*. Todo cambio registrado que quiera hacerle un seguimiento en el area temporal (staging area) debe hacerlo con el comando: **“git** **add”.** Y si, por el contrario, quiere dejar de rastrearlo o de seguirlo (al fichero o carpeta modificada en cuestión), situandolo en su punto de partida de nuevo (ya sea porque se equivocó en alguna modificación y quiere volver a manipular algunas cosas); usted puede ejecutar el siguiente comando: **“git** **rm** **--cached”.** De nuevo, su directorio o fichero en cuestión, vuelve a ser ***untracked***(deja de ser rastreado).

2. Ya lo introducimos antes, se logra la creación de un repositorio local de la carpeta actual (local repo) que recibe por nombre: **“.git”,** una carpeta oculta que emula a la carpeta original y que sirve para guardar, potencialmente, copias de seguridad **(commits)** de todos los cambios o ediciones que estemos haciendo dentro de nuestra carpeta en cuestión. Para tener en cuenta, antes de recurrir a crear una copia de seguridad de algún cambio o edición que haya hecho... debe primero hacerle un seguimiento; es decir, llevarlo a “area temporal”; y, eventualmente, luego querrá llevar esos cambios a repositorio local (ahora sí, al crearles copias de seguridad).

Es preciso decir que, depende de nosotros qué cambios almacenar de forma definitiva en nuestro disco duro por medio de copias de seguridad llamadas “commits”... en dicha carpeta oculta llamada “.git”: nuestro repositorio local. En repositorio local no habrá nada mientras no se haya decidido qué cosas guardar en copias de seguridad.

Justamente, el comando que se encarga de crear esas copias de seguridad se ejecuta así: **“git** **commit** **-m** **‘’”.** La *-m* podría ser considerada como un *elemento* (una extensión del comando “git commit”) que nos permite darle una referencia identificativa y explicita, por medio de un comentario o mensaje, al cambio que estamos almacenando en una copia de seguridad: básicamente nos recuerda qué fue lo que puntualmente modificamos de nuestra anterior versión.

Al ejecutar “git commit” sobre nuestros cambios, éstos pasan de estar seguidos/rastreados ***(tracked)*** por un “area temporal” para llegar a un “repositorio local”. Todo “commit” generado recibe un número de identificación, se llama: **identificador** (un nombre interno de la base de datos de Git para cada cambio), y básicamente con él es que podemos hacer todo tipo de manipulación sobre los “commits”. Al último “commit” o copia de seguridad, nuestra última versión, se le llama también: **“Head”.**

Entonces, tenemos que todas las copias de seguridad que hagamos se almacenarán en repositorio local (y sólo eso). El repositorio local recibe un nombre por defecto de: **“Master”.** *Master* realmente es una rama, la rama principal de nuestro repositorio. Eventualmente necesitaría crear varias cuando esté trabajando un proyecto grande o cuando esté trabajando un proyecto en equipo.

Por ejemplo, con **Git Branch** es posible crear diferentes ramas de desarrollo que pueden converger en el mismo repositorio. Podríamos hablar de: una rama *estable,* una rama *experimental* ***“development”*** *(por ejemplo, para probar nuevo código, fuera de lo convencional que te pide o exige el proyecto, que podría ayudarte a encontrar “por accidente” algo mejor que lo convencional)* y una rama *inestable*, en las industrias a un tipo de ellas se les conoce como:***“bugfixing”*** *o* ***“hotfix”*,** que sirven para arreglar un bug de la versión actual de nuestra rama principal, sin estropear en absoluto todo el trabajo que ya se lleva hecho en ella (para cuidar el trabajo que ya tenemos hecho en código precisamente); entonces, todos los errores a los que nos expongamos, posiblemente, en el intento de resolución de nuestro bug... que se queden y se den, con toda tranquilidad, en la rama experimental. Cuando ya por fin hayamos podido encontrar, en dicha rama experimental, la resolución de problema a nuestro bug, luego de desordenarlo todo sin mosquearnos por eso (no hay nada de qué preocuparse, para eso se creo esta rama); bueno, ahí sí podrá unir la rama experimental, su última versión, a la versión última de la rama *master.* Ya entre poco dejaremos claro cuál es el comando qué se encarga de unir las ramas.

Caso aparte, si está trabajando en equipo, cada integrante del mismo puede trabajar su parte en ramas diferentes; y, al final, cuando cada integrante ya tenga la edición definitiva de su parte (de cada sub-rama derivada de la rama master); entonces, ahí puede unir todas las sub-ramas a la rama principal *master,* logrando así que todos los avances definitivos de cada compañero de trabajo estén unidos a un proyecto final. En general, esta unión entre los avances de las sub-ramas y la rama *master* se hace por medio de un comando que se llama: **“git** **merge”.**

tip: Si por ejemplo me gustaría ver la totalidad de los cambios que he hecho sobre una carpeta, o fichero en especifico, en función de Git; podríamos ejecutar el siguiente comando: **“git** **log”.** Hay podríamos observar, por ejemplo, cuántos “commits” o copias de seguridadad le he creado a una carpeta o a un fichero en especifico.