Conceptos de Git

Bruno Martinez Enriquez

1. Introducción a Git.

(a) ¿Qué es Git?

> Git es un sistema de control de versiones distribuido (DCVS abreviado por sus siglas en inglés) creado en 2005.
>
> El sistema almacena archivos y registra los cambios hechos a éstos en una base de datos (a la cual se le conoce como "repositorio remoto") alojado en un servidor donde una o varias personas tienen acceso para editar los archivos que contiene. Al conocer qué cambio se hizo y quién lo hizo, se tiene un mejor control del proyecto alojado en el repositorio y, ya que los cambios se guardan, podemos revertir alguno que afecte la integridad del proyecto.
>
> Además, tiene la ventaja de que todo aquel que tenga acceso al repositorio, puede tener una copia local de éste; es decir, que trabaja con su propia copia sin necesidad de tocar lo que haga alguien más hasta el momento de juntarlo todo y también implica que si algo pasa con el servidor, estas copias pueden servir incluso como copias de seguridad.

(b) ¿Por qué usar Git?

> Anteriormente, si se quería tener un control de las versiones de un proyecto, se guardaba una copia local en otra carpeta con diferente nombre, pero ésto puede llegar a confundir, además de que no hay manera de controlar los cambios que pueden haber si más de una persona trabaja en este proyecto.
>
> Por lo tanto, Git es la mejor opción cuando se trabaja en proyectos donde trabajarán varias personas y queremos tener un monitoreo de los cambios, además de que siempre se tenga una versión de los archivos que permita el correcto funcionamient del proyecto.

2. Comandos básicos en Git.

Todos los comandos en Git comienzan con `git`.

(a) `git init`.

> Crea un subdirectorio que contiene los archivos necesarios del repositorio, o bien, sienta las bases para crear un repositorio dentro de un directorio.

(b) `git clone <enlace del repositorio>`.

> Crea una copia de un repositorio de Git existente.

(c) `git add <archivo o archivos>`.

> Mueve los cambios desde el directorio de trabajo (*working area*) al área de preparación (*staging area*). En otras palabras, prepara un *snapshot* antes de hacer el *commit*.

(d) `git commit`

Guarda los cambios del área de preparación (*staging area*) al repositorio local. Contiene un mensaje que describe los cambios hechos. Todo lo que se guarde utilizando este comando estará listo para subirse al repositorio remoto.
De manera general, va acompañado de un mensaje que sirve para indicar qué cambio se hizo.

(e) `git push`

Permite mover los cambios hechos con el `commit` del repositorio local al repositorio remoto.

(f) `git revert`

Revierte los cambios hechos en un `commit` anterior, pero en vez de borrar el registro de este `commit`, revierte sus cambios con un nuevo `commit` que comprende el contenido que queremos recuperar, manteniendo el registro del `commit` que revirtió.

(g) `git fetch`

Descarga una rama desde un repositorio remoto junto con sus respectivos *commits* y archivos, pero no los integra al repositorio local con el fin de verificar los cambios que tiene antes de unirlo con la rama local.

(h) `git merge`

Integra los cambios de dos ramas y las une.

(i) `git pull`

Descarga una rama desde un repositorio remoto y actualiza el repositorio local para reflejar el contenido. Es una combinación de `fetch` y `merge`.

(j) `git branch`

De manera general, es el comando que permite la manipulación y creación de las ramas (*branches*), es decir, ambientes de desarrollo aislados dentro del mismo repositorio.

(k) `git checkout`

Sirve para cambiar entre ramas o restaurar archivos de árboles de trabajo.
Si se quiere utilizar para un *commit*, entonces se utilizará de esta forma:
`git checkout id-del-commit`.
Por otro lado, para las ramas, se utiliza así: `git checkout nombre-rama` y para cambiar a una nueva rama se utiliza `-b` después de `checkout`.

(l) `git log`

Muestra todos los *commits* en el historial del repositorio.

3. Trabajando con repositorios en GitHub (Branches, Merge, Conflicts)

Los repositorios remotos son versiones del proyecto que se encuentran en un servidor. Su utilidad que más destaca es que permite la colaboración entre varias personas y poder gestionar los cambios que permitan el correcto funcionamiento del proyecto alojado en el repositorio.

Git almacena instantáneas (mejor conocido como *snapshots*), las cuales son copias de los archivos tal y como se encuentran en un momento determinado al hacer un *commit*. Éstos *snapshots* vienen acompañadas de los datos del autor y un mensaje que explica los cambios que se encuentran en este *snapshot*.

(a) Branches

> Por defecto, cualquier repositorio remoto tiene una rama principal (usualmente llamada *master* o *main*). Los *snapshots* se suben por defecto en esta rama, a menos de que decidamos hacer más.
>
> Para crear una rama nueva, basta con usar el comando `git branch` descrito anteriormente más el nombre de la nueva rama y cambiamos a ésta con `git checkout` más el nombre de la nueva rama.
>
> Crear más ramas tiene la ventaja de poder hacer cambios que pueden llegar a alterar el funcionamiento del proyecto sin el temor de romperlo. Podemos usar esta rama para integrar otras funcionalidades, perfeccionarlas y corregir cualquier error que puede llegar a salir.

(b) Merge

> Retomando el caso anterior de haber creado una rama de pruebas, una vez verificado que los cambios realizados ya no ponen el riesgo el proyecto principal, entonces podemos unir estas ramas de manera.
>
> Podemos utilizar el comando `git merge <rama>` para unir la rama en la que estamos ahora mismo con la rama que recibe el comando `merge`.

(c) Conflicts

> Sin embargo, no todo es tan fácil. Los conflictos ocurren cuando queremos unir dos ramas que contienen un mismo archivo que ambas ramas tienen modificado. Ésto es un problema ya que, por ejemplo, estás trabajando con algunos archivos en un proyecto y otra persona también hizo cambios en alguno de los archivos que tú modificaste, ¿cómo podría Git decidir con cuáles cambios quedarse?
>
> Git marca las partes de los archivos con las cuales tiene conflicto, donde te muestra los cambios que hiciste y los cambios que ya estaban hechos antes de que actualizaras tu repositorio local.
>
> Una solucion a este tipo de problemas es que el encargado del proyecto decida qué cambios son los que deben quedarse y cuáles no. Aunque de manera general se busca que se mantenga una buena comunicación a la hora de que dos personas trabajen sobre una misma rama o cuando la otra persona trabaja sobre la rama padre.