Apuntes Fundamentos

Javier Plaza Rosique

Índice

1. Introducción

En este documento, se podrán encontrar los conocimientos básicos sobre las siguientes herramientas:

- Git, y como server online del mismo GitHub.
- Linux.
- Python.
- Docker.
- SQL.

El objetivo final para mi es llegar a comprender los fundamentos de las herramientas anteriormente mencionadas. Aunque este documento lo publico por si hay alguna persona a la que le pueda llegar a ayudar.

Los conocimientos con los que ha sido redactado este documento son los adquiridos durante el módulo de Fundamentos del Master en Big Data que me encuentro en la actualidad cursando.

2. Git.

Git es un sistema de control de versiones. Sirve para guardar el historial de cambios de los archivos de un proyecto, y también sirve para poder coordinar el trabajo entre varias personas sin que unos se pisen a otros.

2.1. Bases de Git.

Para entender el funcionamiento de Git, antes hay que tener varios conceptos claros. Los conceptos son los siguientes:

- Repositorio.
- Flujo de trabajo con Git: commit, update, push, pull.

Un **repositorio** de Git es una carpeta (como cualquier carpeta del ordenador), en donde se encuentra un subdirectorio (como un archivo) oculto llamado .git, en el cual se guardan los datos de las versiones de los cambios producidos dentro del repositorio. Al poder llegar a tener el subdirectorio descargado en el ordenador, no es necesario tener conexion a internet, pues todos los cambios se encuentran el subdirectorio, por lo que podremos realizar cualquier acción. El repositorio se puede encontrar en local (una capeta del ordenador) o en remoto. Los repositorios en remoto son copias de los repositorios alojados en servidores externos o en la nube, esto permite realizar el trabajo en equipo. Uno de los servicios de repositorios en remoto más extendido es GitHub, pero también se pueden encontrar otros como GitLab.

Commit es la acción que realizamos cada vez que realicemos cambios en el proyecto, y queramos guardarlos. En otras palabras un commit es una instantanea del proyecto a la hora de hacerlo. Dichos cambios se guardan en el repositorio local, es decir, en el ordenador. Las características que todo commit tiene son las siguientes:

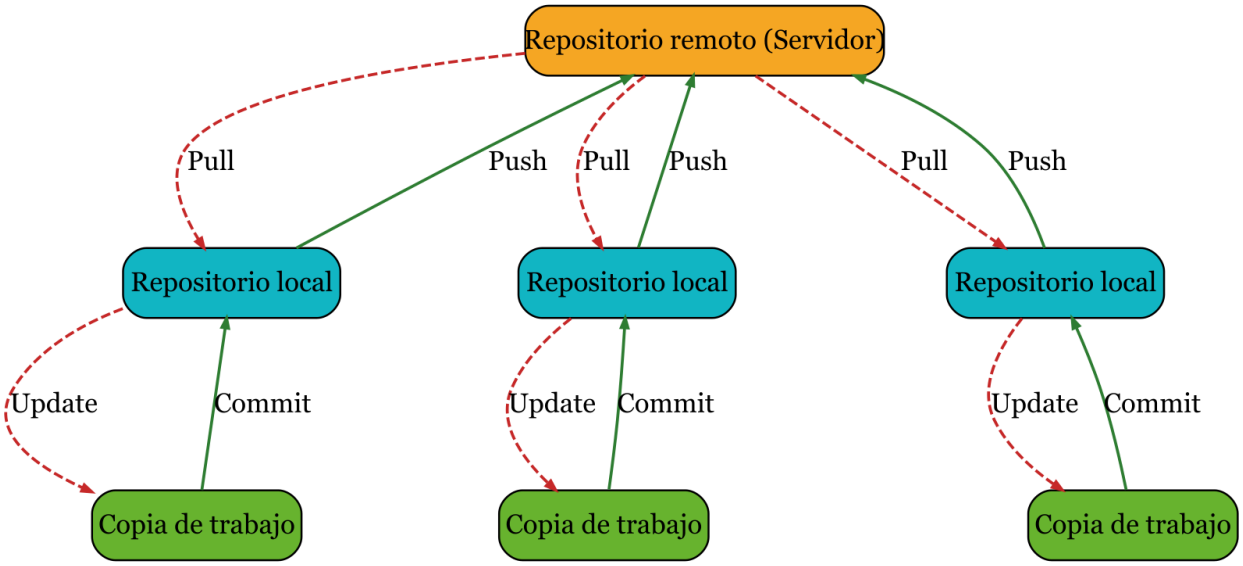
- Título.
- Descripción.
- Hash. Un hash es un código único, el cual es único por cada commit.
- Hora.

Update, en español actualizar, es llevar al espacio de trabajo la información existente en el repositorio local.

La acción **Push** envia los commits que se encuentran en el repositorio local (los nuevos, no los que ya están en remoto) al repositorio remoto. Con esta acción pueden surgir conflictos, es decir, cuando se intenta escribir por ejemplo en una línea ene la qeu ya había información. La solución del conflicto no la realiza ni Git, ni GitHub, dará error. Será la misma persona que realize el push u otra persona la que decide que hacer con el conflicto.

Pull es el update entre repositorios, es decir, que lleva al repositorio local la información recogida en el repositorio remoto. Al igual que en el push, también pueden surgir conflictos. En este caso los conflictos son más complicados que aparezcan, pues se supone que si alguien va a trabajar sobre un proyecto ya empezado, antes se bajará el proyecto al ordenador.

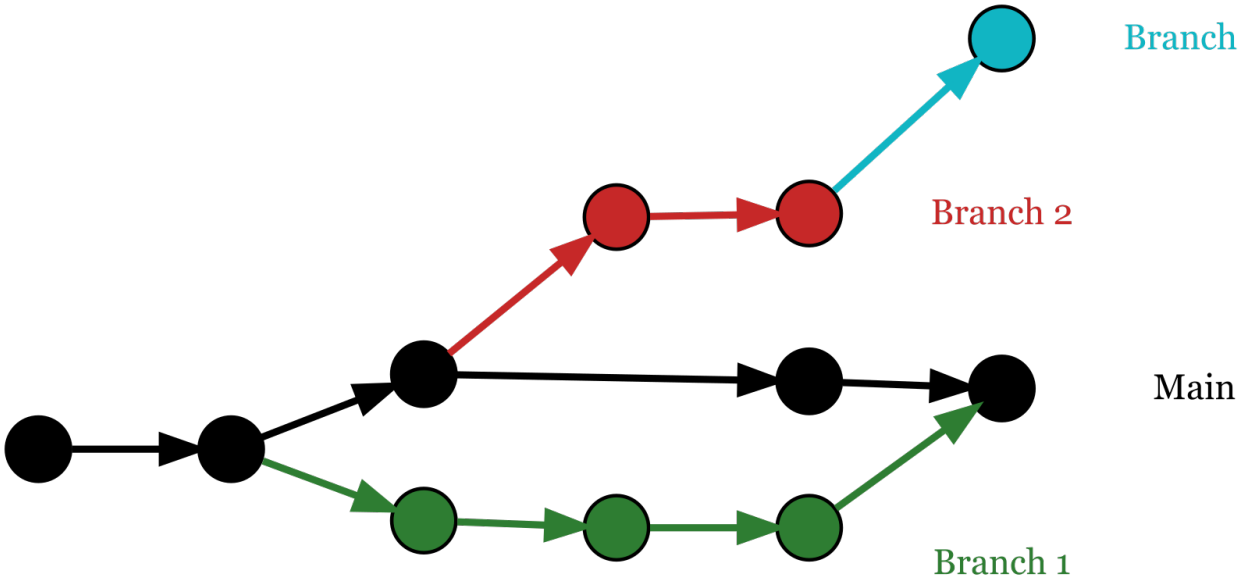
A continuación se encuentra una imagen, que refleja el flujo de trabajo con Git, con lo explicado anteriormente.



Además de lo explicado anteriormente, Git también permite realizar ramificaciones. Una rama es una línea de desarrollo independiente, que permite realizar cambios y probar nuevas cosas sin afectar a la rama principal. También permite trabajar en conjunto con un equipo, teniendo cada uno su rama. Las ramas pueden nacer de la rama principal, que por convención se llama "main" (anteriormente "master"), pero también pueden nacer de otras ramas. Cuando se termina el trabajo en una rama, se une (merge) a la rama de la que salió, pasando a la misma todos los commits realizados en la rama. Al realizar la unión entre ramas pueden ocurrir conflictos si se trabaja en las dos ramas implicadas.

Importante. Es recomendable nunca trabajar en la rama principal.

A continuación se puede observar una imagen donde se ve visualmente el funcionamiento de las ramas en Git.



Dentro de un repositorio de Git, se pueden crear una serie de archivos con funcionalidades especificas. Dichos archivos son el archivo llamado ".gitignore" y el archivo llamado ".gitkeep". Las funciones de cada archivo son las siguientes:

- **.gitkeep**. El archivo permite crear una carpera dentro del repositorio de git, sin la necesidad de tener que crear un archivo con contenido. Se emplea cuando se quiere crear la estructura del repositorio (carpetas), y aun no se quiere empezar a trabajar en el.
- **.gitignore**. El archivo sirve para excluir algunos archivos, por el motivo que sea. Al excluirlos, no los elimina, pero si que no los ratrea ni incluye en los commits. Los motivos para excluir algunos archivos pueden ser variados, pero los más comunes son por las vulnerabilidades que se pueden causar (contraseñas o similar) o por el tamaño de los archivos.

Las acciones que permite hacer Git como los commits, los push y los demás, se pueden hacer mediante interfaces simples, como puede ser el ejemplo de GitHub, el cual trabaja a partir de GitHub Desktop (luego se verá). Pero para poder trabajar sin una interfaz u en una interfaz que no "tengamos controlada", también se puede trabajar por comandos desde el bash.

De manera sencilla bash es un interprete de comandos del sistema, que nos permite realizar acciones como crear capetas, movernos por el ordenador (todo esto se verá en la parte de Linux), pero también permite tabajar con varias herramientas entre las cuales está Git. Según el sistema operativo del ordenador se puede usar el bash de manera diferente. Desde MacOS y Linux, viene por defecto en la terminal, y en Windows se puede emplear desde GitBash, el cual se instala junto a Git.

2.2. Comandos Git.

Los comandos más importantes para usar Git son los siguientes:

Creación del repositorio.

Si se quiere crear el repositorio en el directorio actual, es decir, si la carpeta en la que estamos situados queremos convertirla en un repositorio.

```
git init
```

En el caso de que queramos crear un repositorio de cero, se puede usar lo siguiente. Esto crea el repositorio (carpeta) con el nombre que se escriba en "". Para que se entienda, si estamos en una carpeta del ordenador, crea una carpeta dentro de ella con el nombre introducido, siendo esta el repositorio.

```
git init <directorio>
```

Conexión del repositorio local con el repositorio remoto.

En el caso de conectar un repositorio local con un repositorio remoto, se puede hacer en ambas vías. Es decir que se puede conectar un repositorio local con contenido a un repositorio remoto vacío (creado) o se puede conectar un repositorio remoto al ordenador creando así un repositorio local con el contenido existente en el repositorio remoto.

Para conectar un repo local con uno remoto, antes tenemos que crear un repositorio remoto. Dicho repo se quedará vacío, ya que va a tener los archivos del repo local al conectarlos. Para hacer esto se escribe el siguiente comando en el bash:

```
git remote add origin <URL del repositorio remoto>

# origin es el nombre por el que git identifica al repo remoto, se usa ese nombre
# por convención
```

En el caso contrario se utilizará el siguiente comando, y esto se realizará unicamente la primera vez que "bajemos" el repo.

```
git clone <URL del repositorio remoto>
```