Git para el Trabajo Práctico

> Presentación del video: **git para el TP de Operativos**

¿Qué es y para qué nos sirve Git?

Git es un sistema de control de versionado que, trabajando con otras personas sobre el mismo código, nos permite mantener un historial sobre los cambios realizados diciéndonos quién, cuándo y qué se realizó sobre un proyecto en común.

Local vs Remoto

Un ***repositorio*** podría considerarse un lugar virtual en el que se van a guardar todos los commits asociados a un proyecto.

Un ***commit*** es un conjunto de cambios en tu proyecto. Un commit está asociado a un momento dado del mismo, a una *versión*. Git nos permite controlar los cambios y, de ser necesario, retornar a una versión anterior (o un commit anterior).

Con Git, trabajaremos con un repositorio *local* y un repositorio *remoto*. Pero, ¿qué significa esto?

Cuando hablamos de repositorio remoto, nos referimos al proyecto y sus versiones que se encuentran hospedados en internet como, por ejemplo, en **GitHub**.

Cuando hablamos de repositorio a nivel local, nos referimos al proyecto hospedado dentro de nuestra máquina.

Siempre que hagamos un cambio en nuestro proyecto, el mismo será *local*. Esto significa que, salvo que los subamos al repositorio remoto, nuestros compañeros no podrán verlo ni accederlo.

Por otro lado, cuando nuestros compañeros hagan sus propios cambios y los suban al repositorio remoto, nuestro proyecto no verá esos cambios localmente de forma automática. Es por eso, que tendremos que encargarnos de mantenerlo actualizado.

¿Cómo lo uso?

Obviando algunas otras funcionalidades, para gestionar un proyecto con Git vas a tener que seguir los siguientes pasos:

- **Clonar** un repositorio remoto.
- Realizar los cambios que necesites localmente y luego **commitearlos**.
- **Subir** los commits al repositorio remoto.
- **Descargar** los cambios de tus compañeros a tu repositorio local.

Comandos útiles de Git

A continuación, vas a poder leer para qué sirven algunos de los comandos más utilizados de Git. Recordá que podés ver el resto por tu cuenta corriendo `$ git --help` en tu terminal. Si quisieras entender alguno más en detalle, podés hacerlo corriendo `$ git help <nombre del comando>` .

git config

Si es la primera vez que usamos Git, va a ser necesario configurar nuestro nombre y el email con el que estaremos trabajando utilizando:

sh

```
git config --global user.name "NOMBRE"
git config --global user.email "EMAIL"
```

Si quisieras que la configuración de nombre y mail haga efecto únicamente sobre el repositorio actual, podés omitir el flag `--global` .

git clone

Nos permite "clonar" un repositorio remoto, lo que significa que nos deja descargar la última versión del mismo y copiarlo dentro de nuestra máquina. Uso:

sh

```
git clone <url del repositorio>
```

git branch

Este comando nos lista todas las ramas que existen en el repositorio y nos señala con un asterisco (*) aquella sobre la cual estamos parados (sobre la que se efectuarán los cambios que hagamos).

Para utilizarlo simplemente hacemos `git branch` .

Si en vez de llamarlo de esa manera, hacemos `git branch <nombre de la rama>` , estaremos creando una nueva rama con el nombre establecido.

Si hacemos `git branch -d <nombre de la rama>` , estaremos borrando la rama con el nombre señalado.

git checkout

Se utiliza para cambiar la rama (branch) sobre la que se está trabajando actualmente. Uso:

sh

```
git checkout <nombre de la rama>
```

Si quisiera crear una nueva rama, puedo utilizar el comando con el flag `-b` :

```sh
git checkout -b <nombre de la nueva rama>
```

git status

Utilizamos este comando cuando queremos ver qué archivos han sido modificados y están listos para ser añadidos (git add) o commiteados (git commit).

Lo corremos simplemente haciendo `git status` .

git log

Corriendo `git log` podemos ver el historial de commits hechos hasta el momento.

git add

Este comando se utiliza cuando queremos guardar los cambios realizados dentro del repositorio de forma local.

Para agregar un único archivo utilizamos `git add <ruta del archivo>` .

Para agregar todos los archivos que incluye el directorio sobre el que estoy parado, utilizamos:

```sh
git add .
```

git commit

Una vez que los cambios se hicieron y se añadieron con `git add` localmente, el siguiente paso es *"commitearlos"* (un anglicismo muy popular que se utiliza para denotar esta acción). Un commit guarda el estado actual de la rama del proyecto sobre el que se está haciendo en un momento dado y nos permite regresar a su versión en el futuro si fuera necesario.

Lo utilizamos llamando:

```sh
git commit -m "<mensaje explicativo sobre los cambios>"
```

git push

Cuando hacemos `$ git push` , Git se encarga de subir al repositorio remoto los cambios que hayamos *commiteado* localmente.

Si existen cambios en el repositorio remoto que aun no tenemos, el comando será rechazado y nos pedirá que realicemos primero un `git pull` .

Si el *push* es aceptado, Git intentará de *fusionar* (o mejor conocido como '***mergear***') los cambios automáticamente.

Si reconociera que hubo cambios de dos fuentes diferentes sobre un mismo archivo y línea, esto generaría lo que se conoce como ***MERGE CONFLICT*** y nos pediría que revisemos manualmente la diferencia entre ambos opciones y aceptemos la modificación que corresponda. Luego de arreglar un conflicto de mergeo, el resultado debe ser incluido con `git add` seguido de `git commit` .

git pull

Este comando nos permite traer todos los cambios que existen en el repositorio remoto que todavía no tengo de forma local en la rama actual.

Lo ejecutamos simplemente corriendo `$ git pull` .

git diff

El comando `diff` nos permite comparar dos fuentes de información (commits, ramas, archivos).

Lo que más nos importa es que corriendo `$ git diff` llanamente, podemos ver los cambios que aun no fueron commiteados contra el último commit realizado.

También podemos utilizar `$ git diff <un commit> <otro commit>` para ver las diferencias entre dos commits.

Podés ver todos más específicamente los usos de `git diff` **acá**.

git reset

Si todavía no subimos nuestro commit al repositorio remoto, haciendo `$ git reset <archivo>` , podemos quitar las modificaciones que fueron llevadas a cabo en el mismo del commit.

Algunos videos explicativos

Por si sos una persona más de videos que de lectura, te dejamos a continuación algunos que te pueden ser útiles para que veas la aplicación de los comandos más usados de git. ¡Esperamos que te sirvan 😃!

Introducción a GitHub, git add, git commit, git pull y git push

Git merge, merge conflict, git revert

Branches, git fetch, git checkout, git branch, git merge

Cómo llamar los comandos de git desde GitHub, pull request, merge

Material recomendado

- **Tutorial interactivo de Git Branching** (en inglés)
- **Soluciones a errores comunes al usar Git**
- **Preguntas comunes sobre Git** (en inglés)
- **Git para Diseño de Sistemas**