Jorge Andrés Pardo Rea

Introducción a Git, comandos Básicos, Trabajando con repositorios en GitHub, branches, merge y conflictos

¿Qué es Git?

Git es el sistema de control de versiones más utilizado actualmente. Es un proyecto de código abierto creado en 2005 por Linus Torvalds, el creador de Linux. Muchos proyectos de software, tanto comerciales como de código abierto, utilizan Git para gestionar versiones. Los desarrolladores con experiencia en Git son altamente valorados y Git es compatible con una amplia variedad de sistemas operativos y entornos de desarrollo.

Características de Git

- Git almacena la información como un conjunto de archivos.
- No existen cambios, corrupción en archivos o cualquier alteración sin que Git lo sepa.
- Casi todo en Git es local. Es difícil que se necesiten recursos o información externos, basta con los recursos locales con los que cuenta.
- Git cuenta con 3 estados en los que podemos localizar nuestros archivos: **Staged, Modified y Committed.**

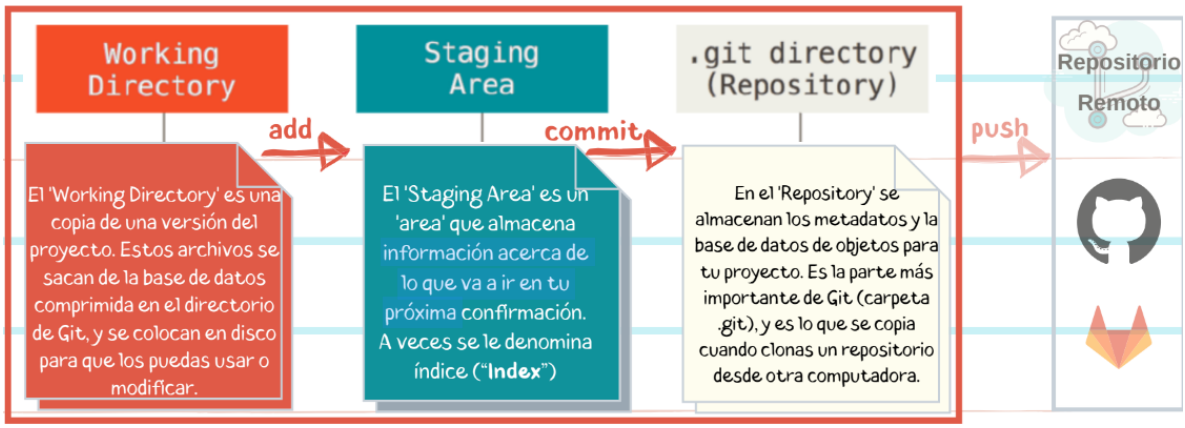
Comandos básicos de git en la terminal

git init <directorio>	Crea un repositorio Git vacío en el directorio especificado. Puedes ejecutar sin argumentos para inicializar el directorio actual como repositorio git.
git clone <repositorio>	Clona el repositorio ubicado en <repositorio> en la computadora local. El repositorio original puede estar ubicado en el sistema de archivos local o en una máquina remota a través de HTTP o SSH.
git config user.name <name>	Define el nombre del autor que se utilizará para todos los commits del repositorio actual.
git add <directorio>	Coloca todos los cambios en el <directorio> para la siguiente confirmación. Puedes sustituir <directorio> por <archivo> para cambiar un archivo específico.
git commit -m "<mensaje>"	Confirma los cambios en estado "staged", utiliza <mensaje> como mensaje de confirmación.
git status	Lista los archivos "staged", "unstaged" y "untracked".
git log	Muestra el historial de commits.

git diff	Muestra los cambios que no han pasados a estado "staged" comparando el índex y el directorio de trabajo.

Ciclo básico de trabajo en Git

Un proyecto en Git tiene 3 'secciones' principales:



El flujo de trabajo básico suele ser de esta forma:

1. Modificas una serie de archivos en tu directorio de trabajo.
2. Preparas los archivos, añadiéndolos a tu área de preparación (staging).
3. Confirmas los cambios (commit), lo que toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera permanente en tu directorio de Git.

¿Qué es Github?

GitHub es una plataforma de desarrollo colaborativo basada en el sistema de control de versiones Git. Fue fundada en 2008 y se ha convertido en una de las principales plataformas para alojar y gestionar proyectos de software. GitHub proporciona una interfaz web fácil de usar y herramientas que facilitan la colaboración entre desarrolladores.

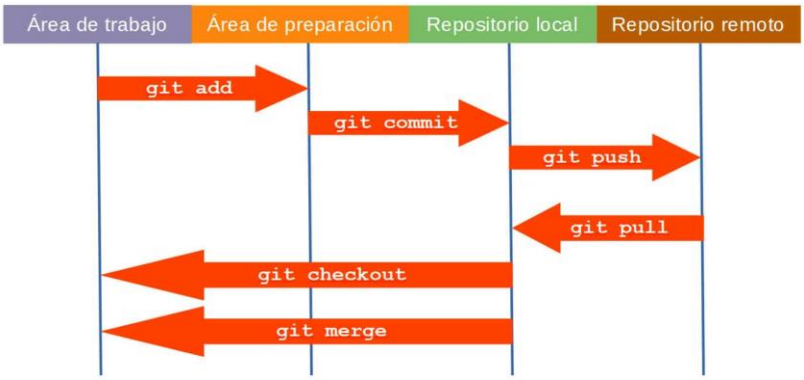
Características de Github

- GitHub permite que alojemos proyectos en repositorios de forma gratuita y publica, pero tiene una forma de pago para privados.
- Puedes compartir tus proyectos de una forma mucho más fácil.

- Te permite colaborar para mejorar los proyectos de otros y a otros mejorar o aportar a los tuyos.
- Ayuda reducir significativamente los errores humanos, a tener un mejor mantenimiento de distintos entornos y a detectar fallos de una forma más rápida y eficiente.
- Es la opción perfecta para poder trabajar en equipo en un mismo proyecto.
- Ofrece todas las ventajas del sistema de control de versiones, Git, pero también tiene otras herramientas que ayudan a tener un mejor control de nuestros proyectos.

Trabajando con repositorios en Github

Los repositorios remotos son versiones de tu proyecto que están hospedadas en Internet o en cualquier otra red. Para poder colaborar con otras personas implica gestionar estos repositorios remotos enviando y trayendo datos de ellos cada vez que necesites compartir tu trabajo.



Comandos importantes

- git fetch

Lo usamos para traer actualizaciones del servidor remoto y guardarlas en nuestro repositorio local. Traemos los cambios que no tenemos, pero no lo combina automáticamente con tu trabajo ni modifica el trabajo que llevas hecho.

- git merge

Este comando se ejecuta para combinar los últimos cambios traídos del servidor remoto (con git fetch), y nuestro directorio de trabajo.

- git pull

Este comando se emplea para extraer y descargar contenido desde un repositorio remoto y actualizar al instante el proyecto local para reflejar ese contenido. Básicamente, git fetch y git merge al mismo tiempo.

- git push

Luego de hacer git add y git commit debemos ejecutar este comando para mandar los cambios de nuestro proyecto local, al servidor remoto.

Introducción a las ramas o branches de git

¿Qué es un Branch (rama)?

Una rama es un nombre que se da a un commit, a partir del cual se empieza a trabajar de manera independiente y con el que se van a enlazar nuevos commits (de esa misma rama). Una rama es simplemente un **apuntador móvil** apuntando a una de esas confirmaciones.

Comandos importantes:

- git branch <nombre rama> Comando para crear una nueva rama, se crea desde el lugar en que te encuentras.
- git checkout <nombre rama> Comando para movernos de una rama a otra, con git status nos indica y confirma que nos movimos a la nueva rama.

Tipos de ramas

- Rama Master o main

Cuando creamos un repositorio (con git init), se genera por defecto una rama que se llama master. Cualquier commit que pongamos en esta rama debe estar preparado para subir a producción.

- Rama Develop

Rama en la que está el código que conformará la siguiente versión planificada del proyecto.

- Ramas Release

Ramas que se generan a partir de la rama develop y se incorporan a esta o a master. Se utilizan para preparar el siguiente código en producción, se hacen los últimos ajustes y se corrigen los últimos bugs antes de pasar el código a producción incorporándolo a la rama master.

- Topic Branches o Feature

Ramas que se generan a partir de la rama develop y se incorporan siempre a esta. Se utilizan para desarrollar nuevas características de la aplicación.

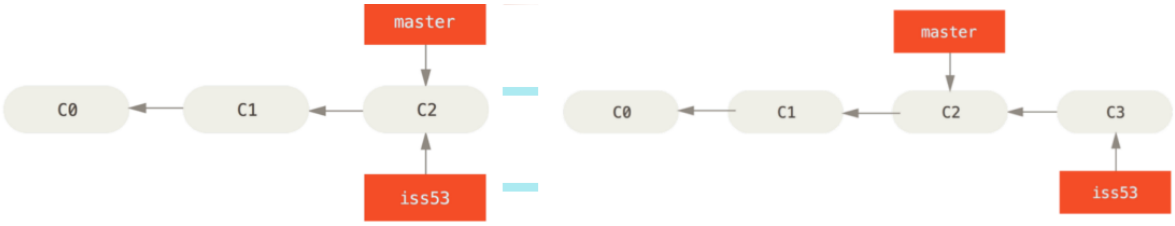
- Ramas Hotfix

Ramas que se generan a partir de la rama master y se incorporan siempre a esta o develop. Se utilizan para para corregir errores y bugs en el código en producción. Funcionan de forma parecida a las ramas Releases, siendo la principal diferencia que los hotfixes no se planifican.

Fusión de ramas con git merge

Procedimientos básicos de ramificación y fusión

1) Para crear una nueva rama y saltar a ella, en un solo paso, puedes utilizar el comando git checkout -b "iss53".
2) En la nueva rama "iss53", realizamos cambios que son independientes de la rama "master", no olvidar ejecutar los comandos add y commit (git commit -am "mensaje") para confirmarlos.



3) Se puede continuar con el trabajo en la rama "master", usando git checkout master. Estando en la rama master se realizan cambios y también se confirman (git commit -am "mensaje")
   Nota: Al hacer checkout, los archivos vuelven al estado tal como se encontraban en la rama.
4) Luego de confirmar que todo está correcto (git status o git log) se necesita fusionar ambas ramas. Con el comando: “git merge <nombre rama>”.
5) En caso de que ya no se necesite una rama y no se usará más, es importante borrarla. Usando el comando: “git merge -d <nombre rama>”.

Solución de conflictos al hacer un merge

En algunas ocasiones, los procesos de fusión no suelen ser fluidos. Si hay modificaciones dispares en una misma porción de un mismo archivo en las dos ramas distintas que pretendes fusionar. Git no será capaz de fusionarlas directamente. Por lo que pausará el proceso, esperando a que tú resuelvas el

conflicto, eligiendo manualmente o con ayuda de un editor de código el contenido que deseas conservar.

Una vez que se resuelve el conflicto se agregan los cambios con un commit. Es recomendable escribir en el mensaje los detalles sobre como has resuelto la fusión.