
[Escuela de código]

# Apuntes de clase:

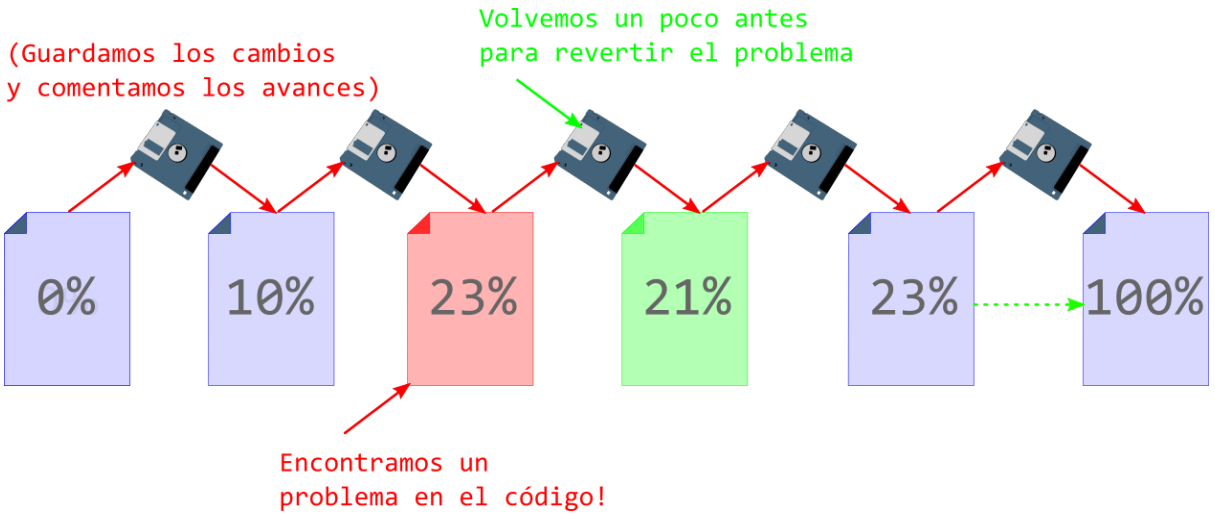
## Introducción a Git

Tabla de contenido

# GIT - Sistema de gestión de versiones

## Qué es un sistema de control de versiones?

A medida que vamos avanzando en nuestro proyecto, se hace de vital importancia contar con un sistema que nos ayude a **registrar los cambios en nuestro código**, y **la posibilidad de volver hacia atrás si hemos cometido un error**.
Eso es lo que nos brinda (en principio) un **sistema de control de versiones**, siendo el más popular **git**.

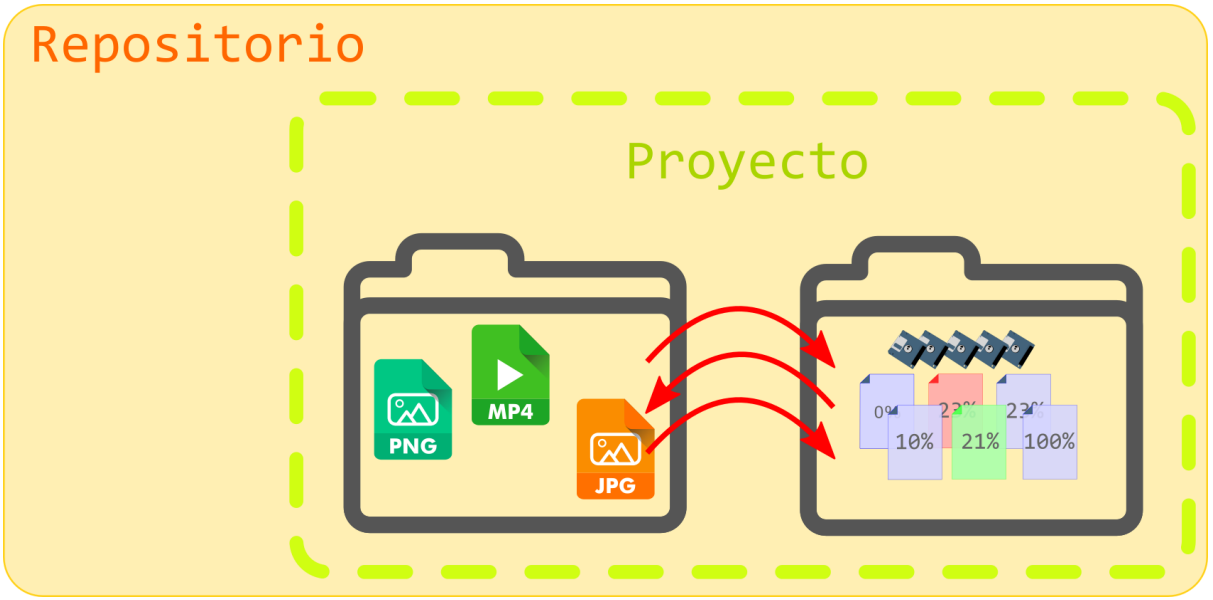



Todos aquellos sistemas basados en git, responden a los mismos comandos y formas de trabajo, como lo hace github.

De esta manera, el sistema de control de versiones mantiene todos los cambios que nosotros queramos registrar, con el comentario que nosotros adjuntamos (a esto lo llamamos "commit"). pero, qué pasa si encontramos un problema introducido al 10% de nuestro avance y vamos por el 84%? perderíamos todo nuestro avance y tendríamos que avanzar desde ese 10%? la respuesta es **no**, git nos ofrece poder **comparar** nuestro código actual con una versión anterior para poder encontrar el problema introducido y solucionarlo. Una vez solucionado, se puede hacer un commit para dejar asentado cuál fue el problema solucionado.

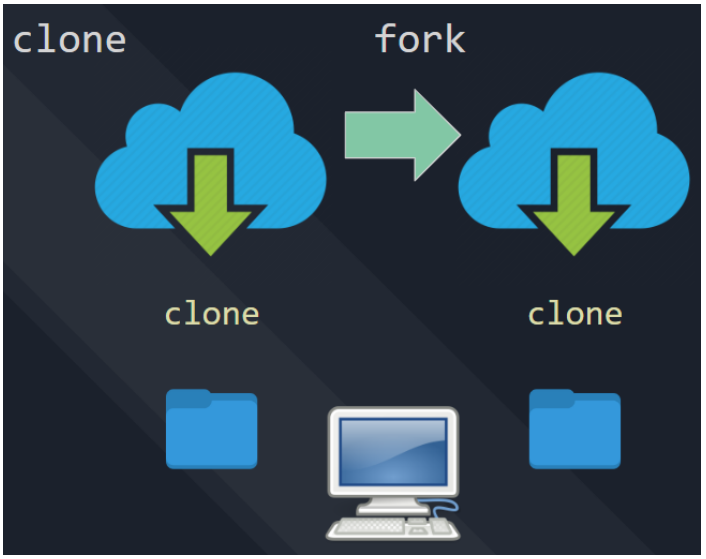
## *¿Qué es un repositorio?*

Un repositorio es un espacio virtual en donde tenemos almacenado todo el proyecto, incluyendo todas nuestras versiones de git, archivos multimedia, etc;



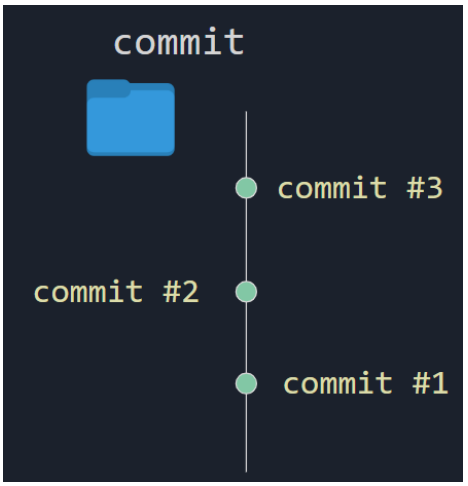

Para poder descargar un repositorio de la nube (el repositorio remoto) utilizaremos el comando de git "**clone**".

Un "**clone**" es una copia exacta de un repositorio en su equipo local. Un "**fork**" es un proyecto (repositorio) completamente diferente que se crea a partir de otro repositorio (es una copia pero cuya propiedad es de quien lo haya copiado.



## ¿Qué es un commit?

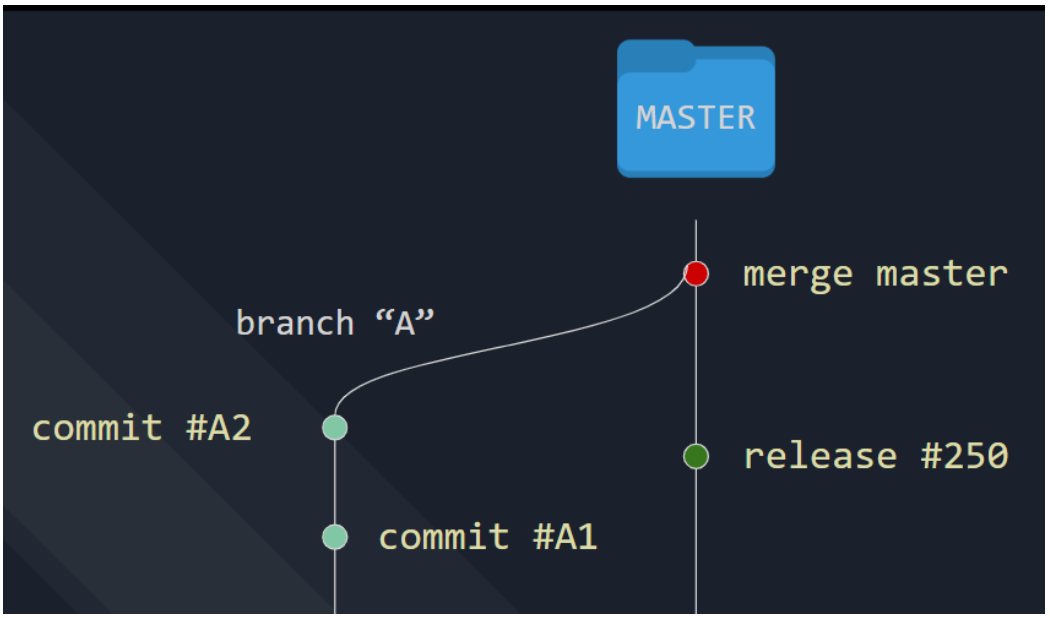
Commit es cada uno de los cambios registrados en el historial dentro de ese repositorio.




Los commits contarán la "historia" de la evolución de nuestro repositorio y software, lo cual nos permite a nosotros o a un tercero entender como el programa llegó al punto en donde se encuentra.

## ¿Qué es un branch o rama?

Los branch son bifurcaciones que toma el proyecto (las ramas). En el branch master se aconseja dejar la versión en producción (el entregable al público). Los branch se utilizan para trabajar en un ambiente aislado para no comprometer al proyecto, no comprometer el programa de producción que es entregado al cliente.



# ¿Head, Master, Origin?

Ya hablamos un poco de los repositorios y de las ramas de nuestro proyecto.

## MASTER

Es la rama principal, en donde se debe encontrar la última versión estable del proyecto.

## ORIGIN

El término "origin" lo utiliza git para referirse al repositorio remoto (en la nube). Todas las ramas que se encuentra en la nube (remoto) comenzarán con el nombre "origin", como por ejemplo:
origin-master → es la rama principal "MASTER" en el repositorio remoto

## HEAD

El término "HEAD" lo utiliza git para simbolizar en el lugar en donde nuestro repositorio local se encuentra actualmente, es la referencia de nuestro repositorio local. Si nos encontramos trabajando en la rama "branch" **desarrollo** el HEAD apuntará a este branch.

Para que nuestro "head" apunte a la rama "desarrollo" usamos el comando:

git checkout desarrollo

Para que nuestro "head" vuelva a apuntar a la rama principal "master" utilizamos:

git checkout master

# Ventaja de Git sobre otros sistemas de control de versión

Git es un sistema de control de versiones distribuidos, más allá de que exista el repositorio remoto con toda la información consolidada cada usuario que participa en el proyecto (en el repositorio) tiene una copia local con la que puede trabajar. Esta copia local tiene todo el historial de cambios, por lo que el usuario puede trabajar y hacer cualquier modificación sin necesidad de conectarse a internet haciendo que el sistema sea mucho más rápido y flexible.
Al terminar el día el usuario debe sincronizar su repositorio local con el remoto en la nube conectándose a internet para subir sus cambios (git push) y que otros miembros de su equipo puedan descargarlos cuando al día siguiente lo primero que hagan sea sincronizarse y descargar el contenido (git pull).

## Buenas prácticas

Hacer commits “pequeños” y frecuentes. Se intenta hacer un commit cada vez que modificamos una funcionalidad o agregamos una nueva, pero no realizar un commit de todo lo que se desarrolló o modificó durante el día.

Esto nos permite poder utilizar los commits para evaluar cómo se implementó o modificó una funcionalidad, nos permite ir a ese commit para ver los cambios puntuales o deshacerlo sin que afecta al resto del sistema.

Deberíamos en lo posible no realizar commits de implementaciones en progreso o no finalizadas, deberíamos en ese caso dividir la tarea o implementación en secciones más chicas realizables durante el día de trabajo.

# GitHub

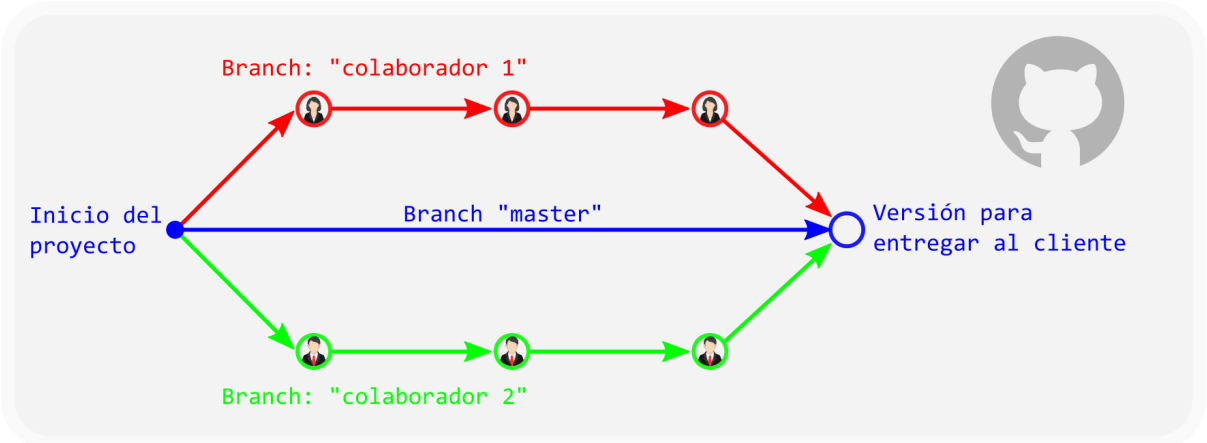
## ¿Qué es GitHub?

Github es un sitio web que nos proporciona:

- ✔ Un espacio virtual en donde alojar el contenido de nuestros proyectos (repositorios), desde el código en sus distintos lenguajes hasta los archivos multimedia necesarios.
- ✔ Un sistema de control de versiones para poder tener resguardo del código y el detalle de sus avances
- ✔ Un espacio colaborativo para el trabajo en equipo.
- ✔ Una manera de compartir nuestro código con el mundo de manera gratuita (Aunque también es posible hacer que nuestro repositorios sean privados, si así lo consideramos).
- ✔ Un lugar de donde obtener código de otros desarrolladores para avanzar más rápido con nuestro proyecto.

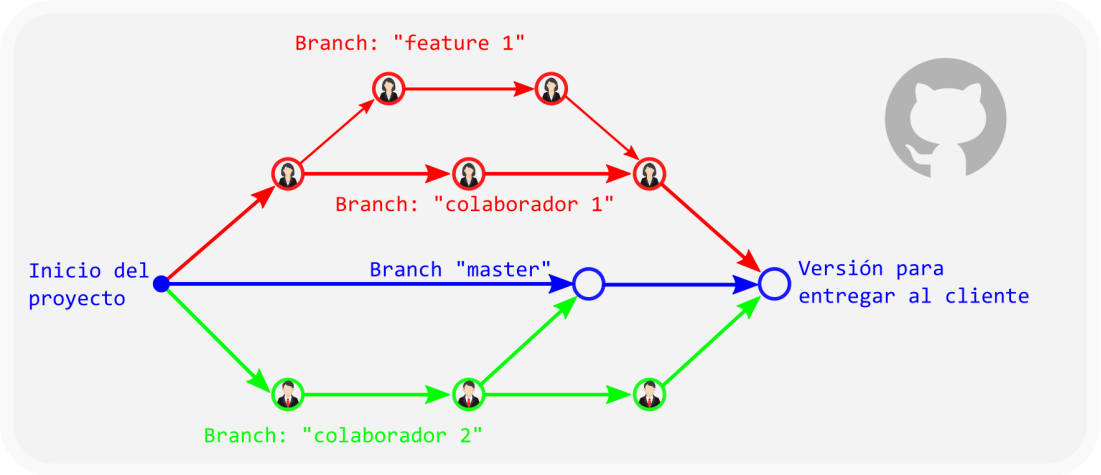
## *Trabajo en equipo*

A menudo los proyectos de software se elaboran de manera **colaborativa**, es decir, que más de una persona está involucrada en el diseño. En un equipo de trabajo deben existir tareas y roles asignados para cada miembro y el sistema git, nos ayuda a dividir el espacio de trabajo de cada uno, manteniendo el control de versiones de cada individuo y del código maestro (master)
Si representamos esto en un diagrama temporal, podemos ver como se divide el proyecto en distintas ramas, o "*branches*":

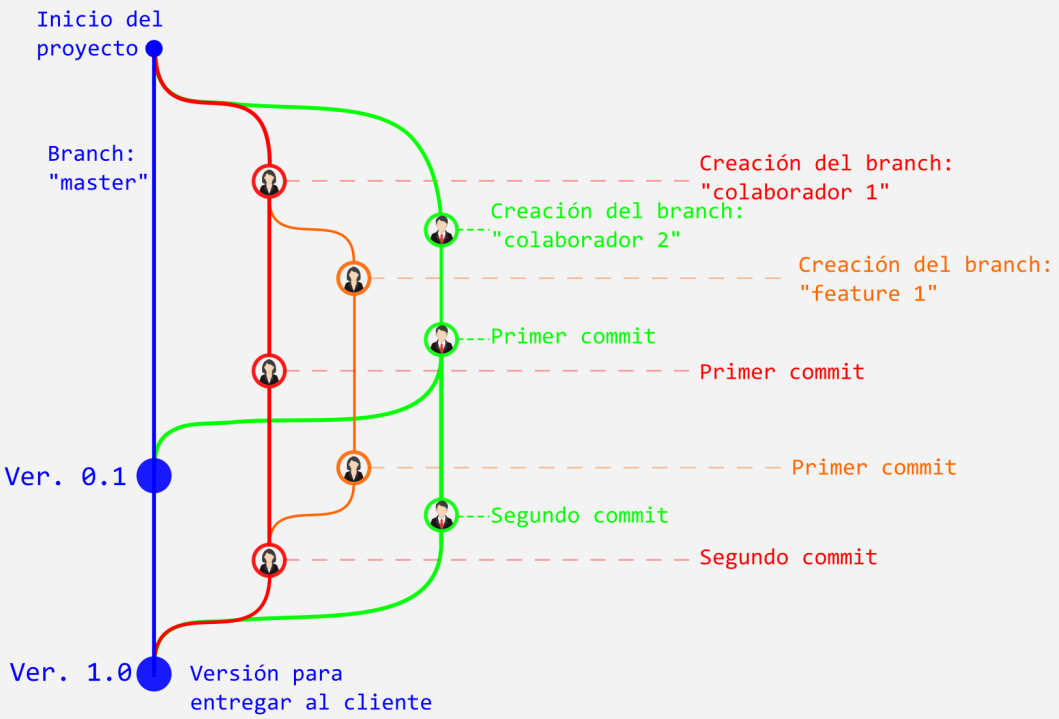


En el gráfico, cada círculo representa un commit, y se puede ver como cada colaborador va avanzando con su parte del código, y registrando estos avances. Una vez que el código está listo, el colaborador incorpora sus avances en el "master"

A su vez, cada usuario puede crear un branch para agregar un nuevo "feature" (característica) a su rama:




Los gráficos en programas como "Git Kraken" o Github Desktop, suelen presentar el árbol de la siguiente manera:

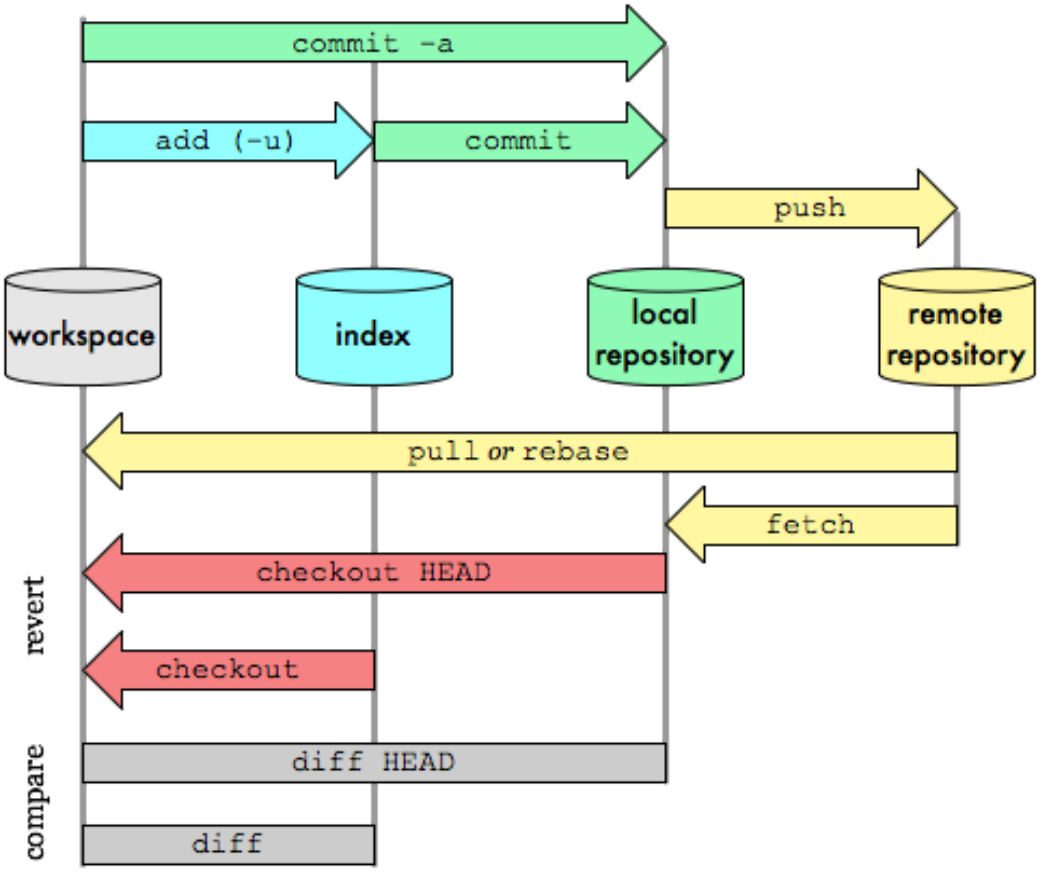



De esta manera un proyecto puede tornarse muy complejo, pero sin embargo, se puede tener completo registro del avance de cada desarrollador.
Para el avance en el contenido del curso iremos incorporando poco a poco el manejo de estas herramientas, por el momento basta con saber que existen, y cual es su propósito.

# Flujo de uso de git

Al comenzar a trabajar con git podemos encontrarnos en alguno de los siguientes escenarios:

- El proyecto recién comienza y no existe el repositorio en la nube. En este caso como necesitamos generar nuestro proyecto para luego subirlo a la nube utilizaremos "**git init**".
- Como desarrolladores nos sumamos a un proyecto existente o deseamos trabajar sobre un proyecto que existe en la nube. En esos casos como bien explicamos antes comenzaremos bajandonos una copia del proyecto con "**git clone**" para poder comenzar continuar desde el punto en donde se encuentre el proyecto.




Si quieren ver más acerca de cómo utilizar git y los comandos más utilizados recomendamos descargarse el siguiente repo:

https://github.com/InoveProyectos/charla_git

# Links de interés

- Instructivo de instalación
- Video Inove: Charla Git y Github
- Lista de comandos Git
- Manual de referencia oficial de git
- Libro de git publicado por github "pro git"