Git & GitHub

Amarok Dev

¿Qué es Git?

Git es un sistema de gestor de versiones, está desarrollado por **Linus Torvalds**. Fue pensado en la eficiencia y la contabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. **Git** nos permite:

- Guardar cambios de manera incremental.
- Contar con un historial y regresar a versiones anteriores
- Registrar cambios de otras personas.

Git mantiene 3 estados principales: **Staged, Modified y Committed.**

Importante:

Los Directorios en Git, son el lugar donde se almacenan los metadatos y los archivos para nuestros proyectos., es precisamente lo que se copia cuando clonamos de un computador a otro.

GitHub

GitHub es una plataforma, con un servicio de alojamiento que ofrece a los desarrolladores repositorios de software usando el sistema de control de versiones **Git.**

Características de GitHub:

- Permite guardar proyectos de manera gratuita y publica, aunque también tiene forma de pago para privados.
- Nos permite colaborar con otras personas para mejorar nuestros proyectos.
- Ayuda a reducir muchísimo los errores humanos, en términos del mantenimiento y detección de fallos.
- Definitivamente el trabajo en equipo.

Recomendación:

Saber un poco sobre consola del sistema y algunos de sus comandos, para poder desplazarce en carpetas, crear, modificar y eliminar directorios y archivos.

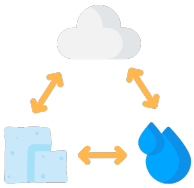
Comandos básicos:

Lo primero que haremos será crear un repositorio en Git, mediante el comando git init.

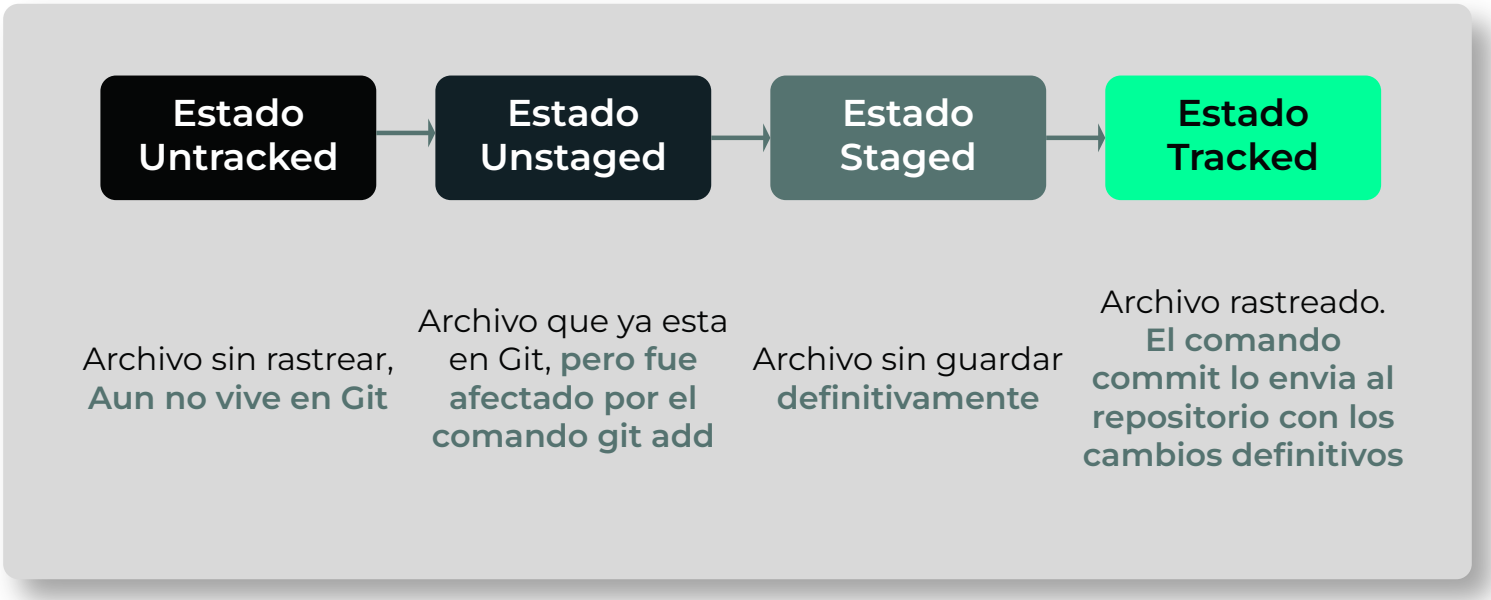
Este comando nos creará un área conocida como "**Staging**" aquí se guardará temporalmente nuestros archivos, se crear un repositorio local y una base de datos histórico del proyecto.

Estados de nuestros archivos:

Git cuando nosotros comenzamos a hacer repositorios, nuestros archivos pasan por diferentes estados



Con el comando **git status**: nos muestra en que estado se encuentran nuestros archivos.



Comandos:

git status: Ver el estado de nuestros archivos.

git add: añade el archivo a Staging.

- **git add nombre_archivo**
- **git add.** Añade todos los archivos.

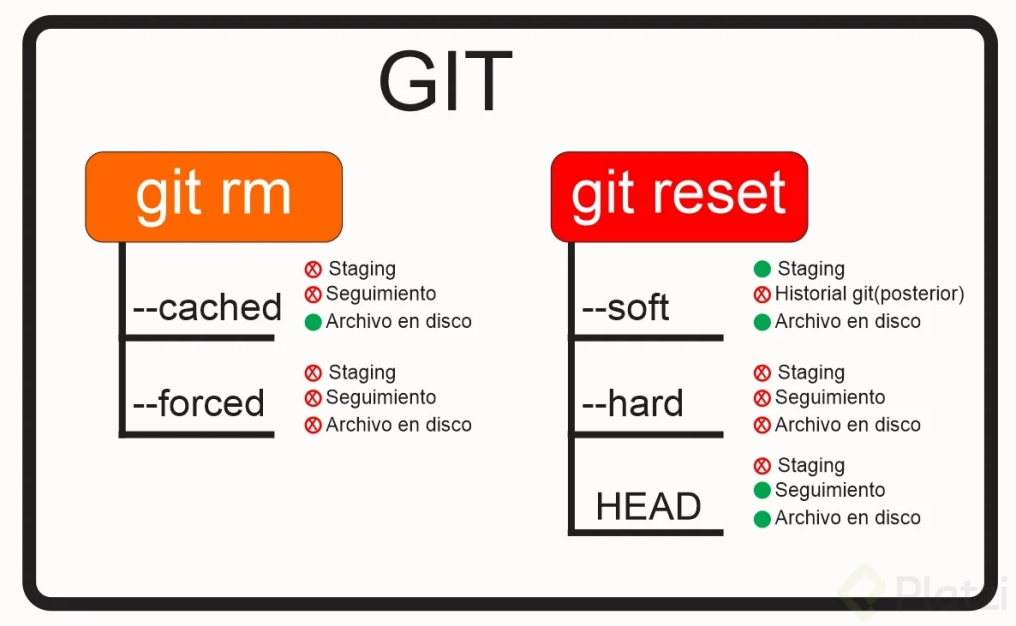
git commit: Guarda los archivos en el repositorio local.

- **git commit -m “mensaje”**

git reset head: Quita archivos de Staged o los devuelve a su estado anterior.

git rm: elimina el archivo de tu directorio de trabajo y no elimina su historial del sistema de versiones.

- **git rm --cached:** mueve los archivos que indiquemos al estado untracked
- **git rm --force:** elimina los archivos de git y del disco duro



¿Qué es Branch (Rama)?

Git nos permite crear diferentes ramas para abordar los proyectos desde diferentes problematicas o realizar experimentos sin afectar el proyecto principal.

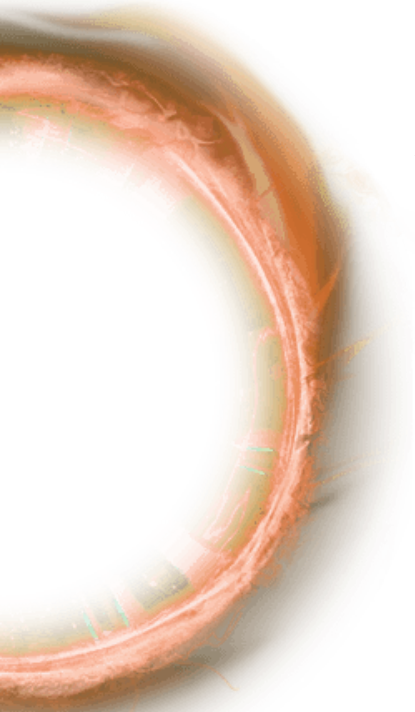
La rama principal se llama Master.

Importante:

Podemos crear todas las ramas que queramos, y podemos ir hacia adelante o atras en las versiones de cada rama, ademas se pueden conectar las diferentes ramas entre si desde cualquier version de rama.

¿Cómo crear ramas?

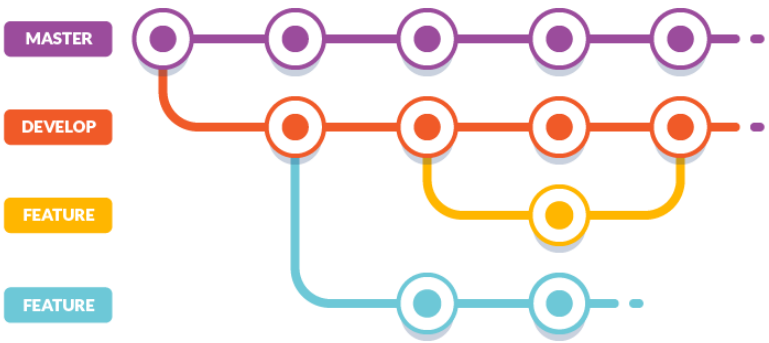
Se usa el comando **git branch nombre-rama** Esto copia el ultimo **commit** a esta rama.



Para pasar de una rama a otra, se usa el comando: **git checkout nombre-rama**

ADVERTENCIA:

No olvidemos hacer **commit** antes de migrar a otra rama, para no perder el progreso.



Para unir ramas

Primero nos ubicamos en la rama que vamos a elegir como la rama principal, por lo general es la rama **Master** y digitamos el siguiente comando: git merge **nombre-rama** Esto hace un commit a la rama principal, agregando el historial de cambios.

Importante:

Si no sabemos el nombre de la rama a la que queremos acceder, con el comando git branch, podemos ver las ramas disponibles en el proyecto.

Conectando con GitHub

Amarok Dev

Crear un repositorio en GitHub

Iniciarlmente abrir una cuenta en **GitHub** y darle a la sección de nuevo repositorio, le damos **nombre** y **descripción**. Se recomienda activar la opción de crear un archivo **Readme**.

Pasos para subir el repositorio:

- Dar click en la opción de **“Code”**
- Usar la opción de **HTTPS** y copiar la **URL**
- Volvemos al repositorio local desde **Git Bash**
- Colocamos el comando **git remote add origin**
- Pegamos la **URL** y presionamos **enter**

Importante:

Si colocamos el comando de git remote -v nos da la opción de hacer fetch (Recivir datos) y push (Enviar datos)

- Colocamos el comando **git push origin master.**
- Es probable que nos pida iniciar sesión con **GitHub.**
- Por lo general nos sale un **error**. Este error es por que en el mundo **remoto** hay cambios no registrados en el mundo **local**.
- Se soluciona con el comando **git pull origin master.**
- Ahora para unir las historias relacionadas, se usa el comando: **git pull origin master --allow-unrelated-histories**
- Esto habilita el merge para unir la rama **master remoto y local**. Le damos aceptar.
- Ahora si podemos hacer **git push origin master**.

Importante:

Para la seguridad informatica en el manejo de repositorios ubicados de manera remota, se recomienda el uso de llaves publicas y privadas, teniedo en cuenta que la información que se vatya a manejar puede ser sensible despendiendo el tipo de proyectos

Configurar las llaves en local.

Hay que estar en el Home:
ssh-keygen -t rsa -b 496 -C “Correo@mail.com”

 # -t Qué tipo de algoritmo se va a usar para crear esta llave.
 # -b Qué tan compleja es la llave desde una perspectiva matemática.
 # -c Qué correo electronico va a estar conectada a la llave, Email de GitHub.

Recomendaciones:

- Revisar que nuestras llaves se han generado con éxito en la ruta de archivo, por lo general en **C:\Users\TuUsuario\.ssh**
- Revisar que elservidor de SSH este corriendo: **eval $ssh-agent -s**

Agregar la llave que acabamos de crear

Hay que estar en el Home:
ssh-add ~/.ssh/id_rsa

Nos solicita crear contraseña.

Agregar Branches y Tags:

Los branches son la ramas de trabajo creadas anteriormente, pero cuando hacemos un commit al repositorio remono, no necesariamente se suben las diferentes ramas, del mismo modo pasa con los tags

Comandos git

Comandos básicos:

git tag -a nombreDelTag -m “mensaje” HASH-a-etiquetar

ejemplo:
git tag -a v0.1 -m “Primera version” 33bac85

Enviar tag a repositorio remoto

Se usa el comando: **git push origin --tags**
Para Eleminar un tag: **git tag -d nombreDelTag**

Despues de eliminar y enviar los cambios a GitHub, nos damoscuenta que todavia aparecen los tags, por ello se hara lo siguiente:

Hay que eliminar un tag y su referencia en el repositorio remoto, de la sigueinte manera:
git push origin :refs/tags/NombreTag

Para enviar ramas a nuestro repositorio remoto se hace así:

git push origin NombreRama