# 9-¿Por qué usarlo?

Sistema de control de versiones🡪Git (Creado por linux). Son cambios muy pequeños, para q tener otro archivo? (Nos ayuda a solo guardar los cambios q se hayan hecho, NO TODO EL ARCHIVO)

* Archivo de texto plano 🡪Ejemplo .txt (No negrita).

**Git init**🡪Empezar en tu carpeta un repositiro(Carpeta donde se guardaran los cambios)

**Git add texto.txt**🡪La base de datos de sistema de control de versiones sepa q esta el archivo texto.txt.

**Git commit -m**”version 1” 🡪envia los ultimos cambios a la base de datos del sistema de control de versiones (para controlar los cambios q se le hayan hecho).

**Git status** 🡪estado en la base de datos del sistema de control de versiones

**Git show**🡪Todos los cambios historicos hechos(Toda esa información)

**Git log** 🡪toda la historia

**Git push** 🡪Enviar hacia otro repositorio lo q estas hciendo

**Recomendación🡪** Archivos de texto plano(MAS EFICIENTE EN GIT) que los archivos de texto binario.

**GITHUB🡪** Es un servicio de alojamiento que ofrece a los desarrolladores repositorio de softare usando el sistema de contro de versiones de Git. Analogo a una red social.

# Editores de codigo para trabajar con Git y archivos de texto

Archivo binario(archivos binario atraves de un IDE, estructura interna del word es binaria, esta estructurada en 0,1, word lo entiende y lo representa) vs archivo plano().

**LO QUE VAMOS A PODER MODIFICAR Y GUARDAR MODIFICAR ES TEXTO PLANO**.

# Introducción a Terminal y Línea de comandos

* Pwd🡪 en q carpeta estas actualmente
* Ls🡪Listar los archivos q estan allí en esa ruta
* Ls-al🡪Lo mismo pero trae los ocultos
* Ctrl+l🡪Limpiar
* Cd .. 🡪Regresa a la carpeta anterior
* Cd + **Inicial de nombre+**tab
* Touch nombre.txt 🡪Crear archivos
* Cat nombre.txt🡪Que tiene adentro
* Rm nombre.word 🡪 Remover.
* History 🡪Todos los comandos q he hecho.

# Cómo crear un repositorio de Git y hacer tu primer commit

//**git rm --cached**🡪esta en memoria ram, no en la bd de la sistema de cotrol de versines. El archviio esta en ese mundo stage (no en el rpeositorio aun).

**Git show** 🡪cambios q han existido sobre un archivo

**Git init** 🡪Inicializar repositorio donde se guardará en la base de datos del sistema de control de versiones.

**Git status**🡪Para ver el estado actual del archivo(si hay cambios pendiente o que)

**Git add** 🡪Añadimos el archivo q queremos. Aca estaría en **staging (**aun no se ha subido al repositorio) esta en ram.

**Git commit -m**”Comentario” 🡪Guardamos los cambios echos al repositorio

**Git log** 🡪vemos todos los cambios q se han hecho con su codigo diferenciador para posteriomente compararlos.

**Git diff** **id1 id2**🡪Muestra las diferencias de codigo entre 2 versiones distitnas. Esos **id1 id2** es el nombre interno de los cambios realizados con el commit en la base de datos del sistema de control de verisones. Tambien no es necesario poner los id1 id2, con solo po0ner git diff (diferencias entre directorio de trabajo actual y staging)

# ¿Qué es staging\_ Repositorios en git

1. codigo.js esta en un archivo en el computador

2. **git init** 🡪se crea un area en memoria ram q se llama **staging** donde al ppio se van a ir aggndo los cambios (espacio temporal). Tambien se crea el repositorio (q no es mas q es la carpeta .git)🡪donde estaran todo slos cambios al final.

3. **git add** codigo.js 🡪archivo pasa a **staging (**esperando a ser mandado al repositorio). Untracked🡪git add🡪tracked(Aca hace parte de **staging, donde los cambios ya quedan tracked**).

Git esta revisando constantemente ¿hay cambios? Hay cambios?-->una vez se da en git add, esas **modificaciones** van a **staging**.

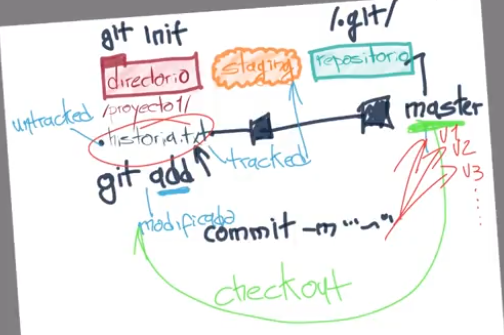
**Staging🡪**area de preparacion, temporal, entes de envias los cambios al repositorio q es donde se guardan.

4. **git commit -m**”comentario” 🡪archivo se va a repositorio (nombre por defecto q se llama **master,** donde estará todo los cambios PASAN cambios en staging a el repositorio. **Ab32131232** idnicadores del cambio que se hizieron. Cada q hago modificacion, es una nueva version de cambio del archivo. Esos numeros rarios es el nombre interno en la base de datos de git de ese cambio.

## ¿ traerte un cambio q esta en el repositorio pero no en tu carpeta?

Ejemplo, un compañero hizo un cmbio al archivo pero tu no lo tienes porq trabajaron sobre el mismo archvio.-->te vas a la rama, y para traerla a tu carpeta **checkout** (traerte los cambios hacia tu carpeta).

¿Qué pasa cuando quieres trabajar con otras ramas? Ejemplo, queremos lanzar una version experimental q no queremos mandar al repositorio principal master sino a una rama q se llama desarrollo y despues unirla a la rama master. Si quiero crear una rama de header, rama footer, rama contenido central y despues unirlo. Ramas🡪Romper en pedazos en diferentes lineas de tiempo el codigo y luego unirlas al final.



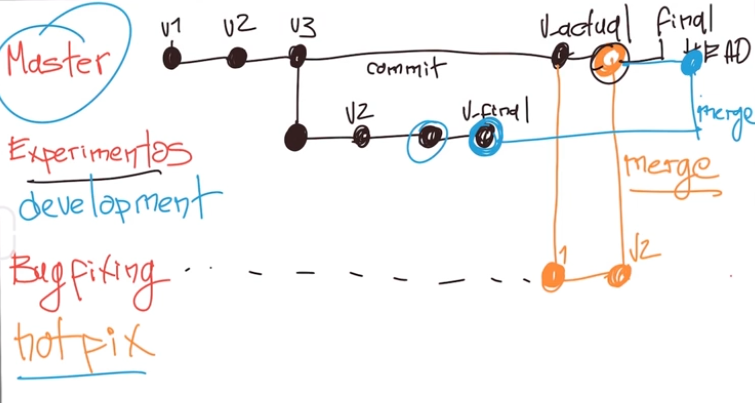
# Que es branch o ramas?

Por defecto, estas en la rama **master** en donde se guardan las versiones de los archivos. Pero supogamos que quiero trabajar en experimentos del codigo, para eso creamos una rama **developments (Experimento) y copias la v3 y sobre ella trabajas**  y ahí sigo con los commits que yo quiera. Esa rama es completamente diferente.

Luego salió un bug, necesito arreglaro, creo una rama especial **hotfix (**arreglo en caliente), hago mis cambias. Luego, conecto los cambios de na rama con otra mediante **merge** y termino ya con una version final.

Luego, tengo los experimientos hechos. Uno la **v-final** q se fusione con la **vfinal** en el repositorio **master**.

CUIDADO🡪Merge rompe otros archivos y esto se llama conflictos



# Cómo usar GIT reset y checkout

* **Volver al pasado de los archivos**
* Existen disntios tipos de **reset** (hard q devuelve y elimina TODO cambio de versiones anteriores)
* Git scheckout 🡪Para ver como era antes el archivo en una versión q hayamos elegido.
* Git reset **id1 -hard --soft(volvemos a esa version pero lo q tengamos en staging se queda, si le haz dado git add. Eso sigue estando ahí)**

$ git reset b8e76da401741fa6db5e90e19a30ceed2fac28b3 **–hard**

$ git diff

$ git checkout b8e76da401741fa6db5e90e19a30ceed2fac28b3 **codigo.js ->De esa versión SOLO traemos a la version del id(nombre raro) ese archivo, los demás los dejamos intactos.**

$ git checkout **master codigo.js (la version de ultimo, no el q tragimos de una version anterior, mirar justo atrás de esta linea)**

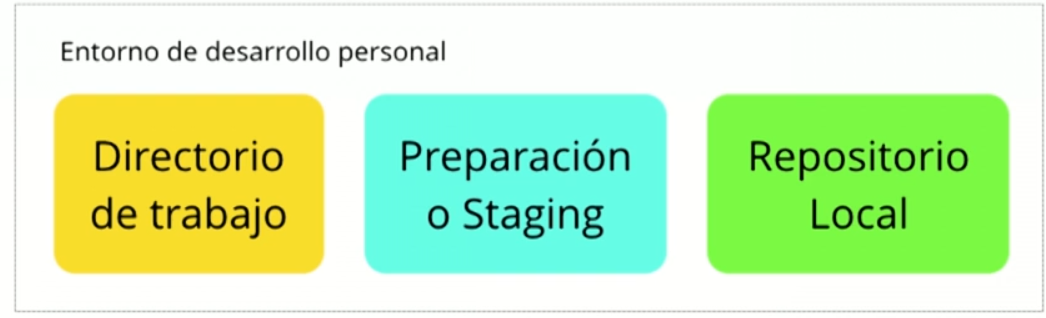
* **Si lo qeremos cambiar, simplemente hay q hacerle un commit.**

# Git flow

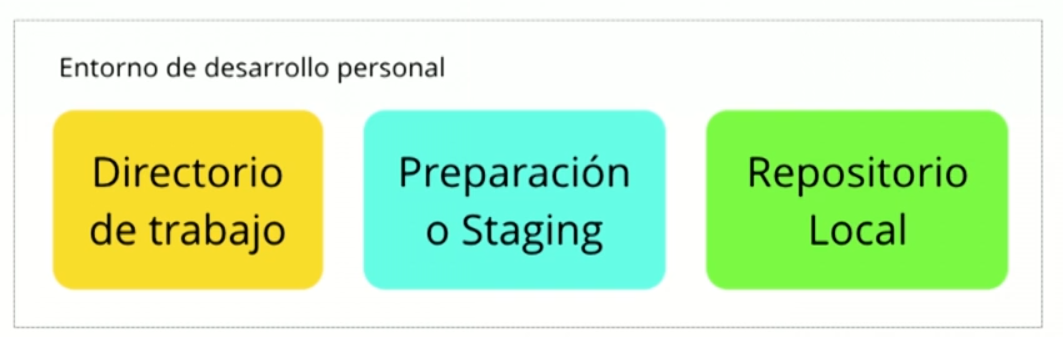
# How to integrate Git Flow in your team? - DEV Community Flujo de trabajo basico en Git

## Como trabajar con repositorios remotos en GIT

* 1. **Git init**:



* 2. En el repositiroio hay copias de todo lo q haz hecho.
* 3. **Git add**🡪git lo esta rastreando ese arrchivo “Tracked” o rastreado.
* 4. **Git commit** 🡪Todo lo q esta en stagging se envia al repositorio local
* Pero que pasa si es un equipo?--> Servidor remoto (Repositorio remoto, ejemploL github, git lab).
* En vez de hacer git init, hacemos git clone url.
* **Git fetch**🡪traerme un cambio de trabajo de un archivo en un repositorio.

1. 
2. Diagrama

   Descripción generada automáticamente Diagrama

   Descripción generada automáticamente
3. Texto

   Descripción generada automáticamente con confianza media

**Equipo de trabajo**

1. Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

   Descripción generada automáticamente
2. **En vez de git init, hacemos(crea la base de datos de todos los cambios en el repositorio local**
3. Diagrama

   Descripción generada automáticamente
4. Diagrama

   Descripción generada automáticamente
5. **Alguien mas hizo un cambio y quiero traerlo a mi computador**

Diagrama

Descripción generada automáticamente

**6.** Diagrama

Descripción generada automáticamente

7. Es un **git merge** y un **git fetch** fusionados en el **git pull**. Copio el repositorio local, la base de datos de cambios de cambios y lo copio al directorio. De esta manera tengo una copia actualizada.Diagrama

Descripción generada automáticamente

## Que son y como funcionan las ramas o bracnges de Git

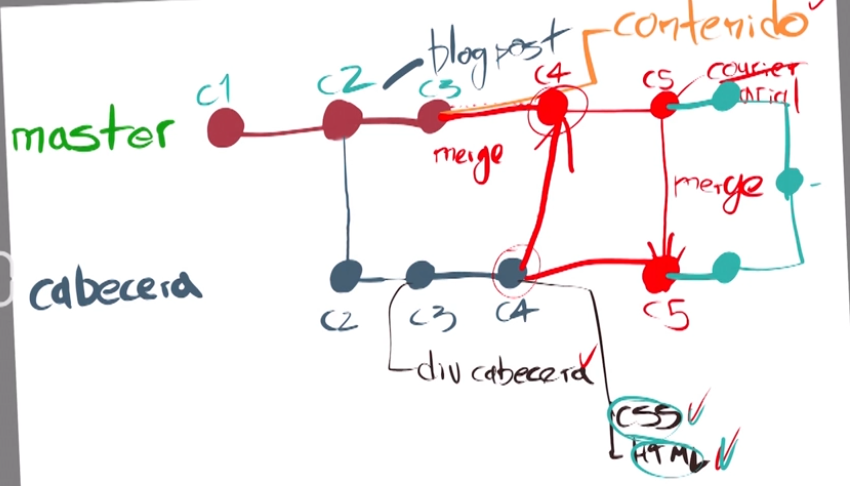
* Master. Las ramas son formas donde hacemos cambios sin afectar la rama principal. La creacion de una rama es una copia del ultimo commit.
* En este ejemplo creamos una rama cabecera. Lo q se crea en una rama es crear una copia del ultimo commit.
* Git status🡪para ver en q rama estoy.
  + Mirar donde aputan el HEAD
* Git branch **cabecera🡪** para cambiar a la rama cavera (hice una copia exacta del ultimo commit de la rama donde estaba (**master)**
* Git checkout cabecera🡪para moverme a la rama cabecera
  + Modificamos lo q necesitemos modificar en esta rama
  + Git add .
  + Git commit -m”comentario”
* Git chekcout **master🡪**vuelve a la rama master con su contenido

## Que es Git merge y como funciona la fusion de ramas

* Spoiler : Git merge ->la rama cabecera desaparece y se une al merge.
* Git add .
* Git commit -m”estoy en la rama cabeera modificando”
* Git checkout master🡪modifico esta rama
* Git add .
* Git commit -m”modificando la rama master”
* El merge siempre va a ocurrir en donde estoy. Quiero traerme a **cabera** desde **master.**
* **Estoy en master. 🡪** git branch para ver las ramas donde estyo
* Git merge **cabecera**

## Como solucionar confilctos al hacer git merge

* Cuando 2 o mas personas cambian **exactamente la misma linea de codigo.**
* **Modifico las 2 mismas lineas de codigo**
* Git add .
* Git commit -m”modifico las mismas lineas de codigo en rama **master”**
* Git add .
* Git commit -m”modifico las mismas lineas de codigo en rama **cabecera”**
* Aparece los conflictos en el **IDE**  VSC.--> Accept Current Change||Acept comming change.
* Luego hago un **git add .**
* **Git commit -m”Solucione el conflicto”**



# Trabajando con repositorios remotos

GitHub🡪Servidor de git para clonar nuestros repositorios.

🡪Interfaz de tus repositorios (la red social de nuestros repositorios)