

Git desde Zero

Richard Marin Benavides

Fuentes: Git from Scratch, Morten Rand-Hendriksen

Learning Git and GitHub, Ray Villalobos

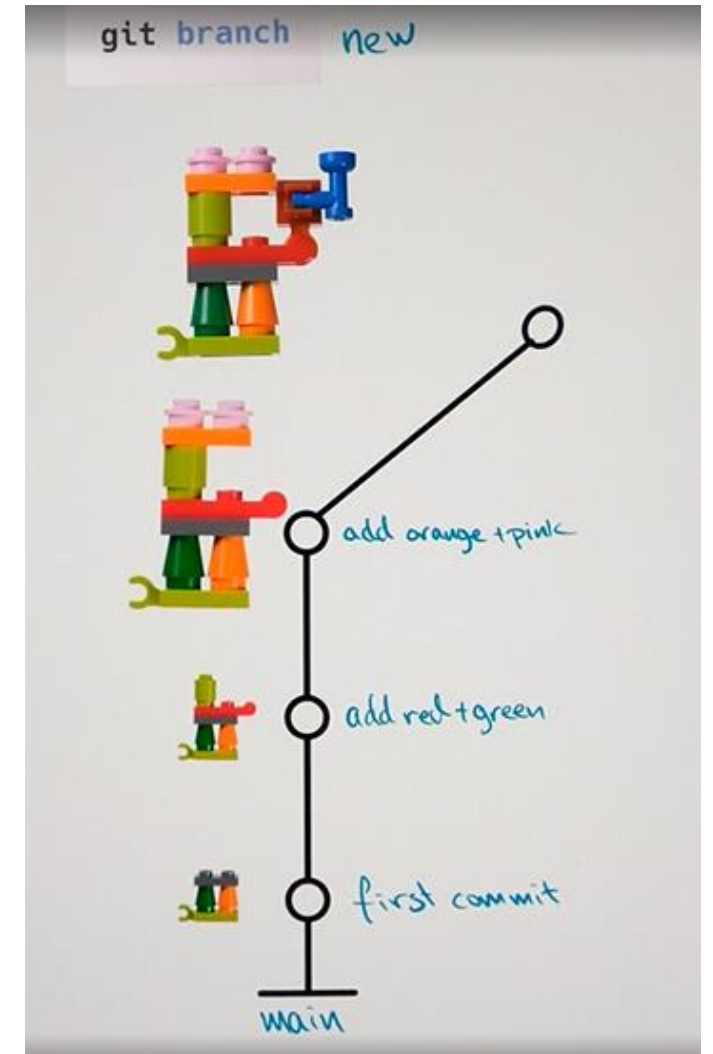
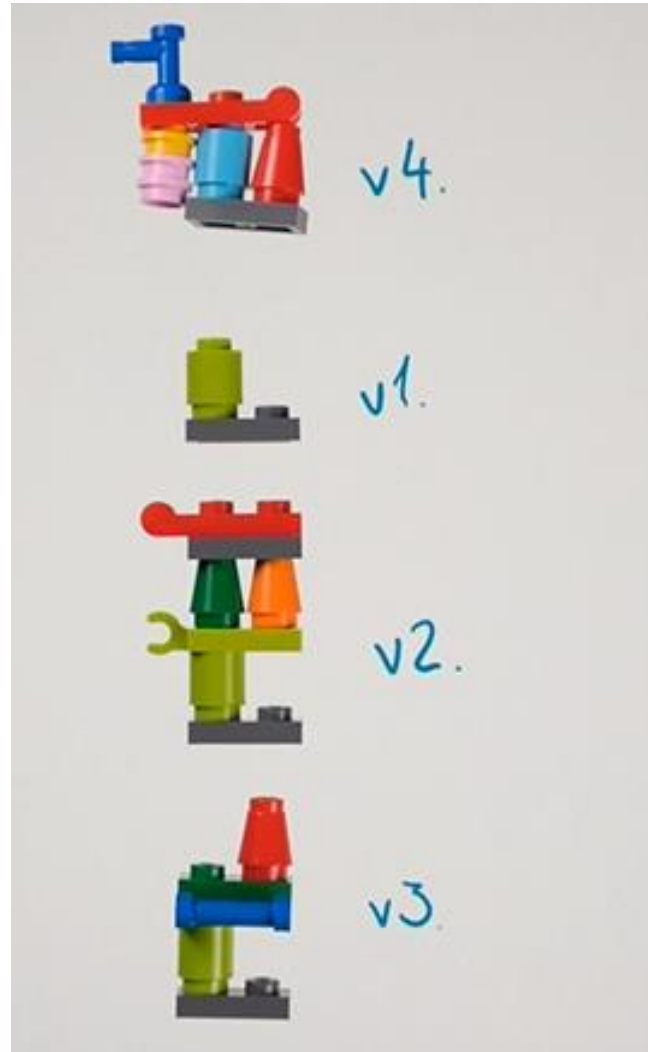
Qué es Git?

- Control de Versiones
- “Máquina del tiempo”
- Puntos de control (commits)
- “Multiverso” (branches)
- Sincronizador (merging)

Commandos Básicos

Porqué usar control de Versiones?

- Se puede ir atrás en el tiempo y arreglar errores!
- Se hacer “fotografías” y se puede devolver a ellas en cualquier momento.
- Se pueden hacer “branches”, las cuales son ramas o líneas de tiempo alternativas
- líneas alternativas de tiempo se pueden unir.



Git Init: Inicializa un Repositorio



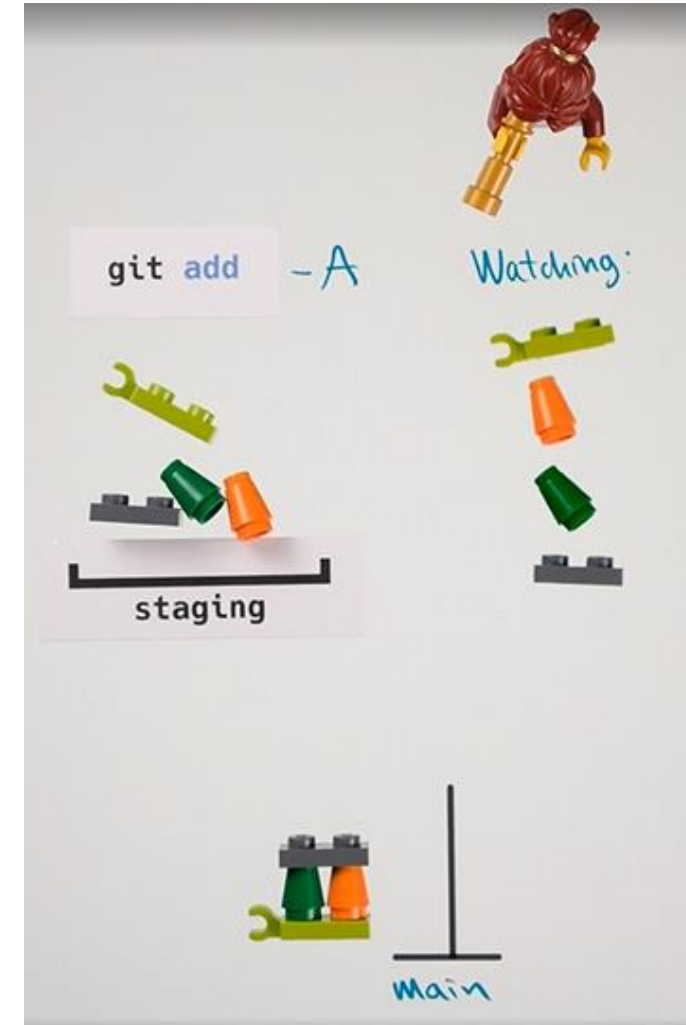
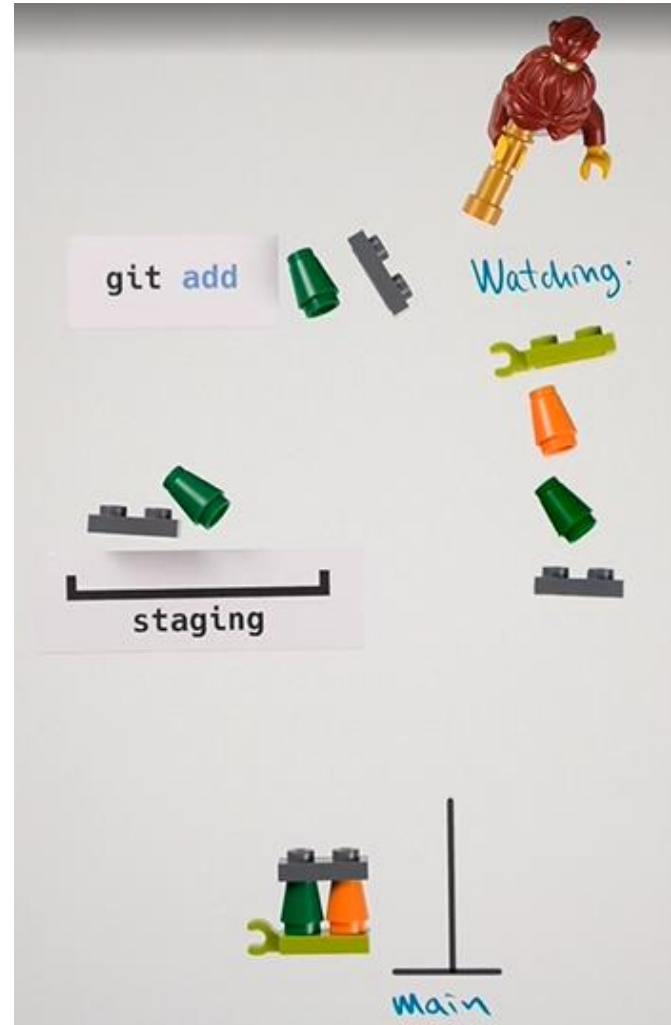
Ignorar control de versiones en ciertos archivos

- Agregarlos en el .gitignore del folder raiz de git



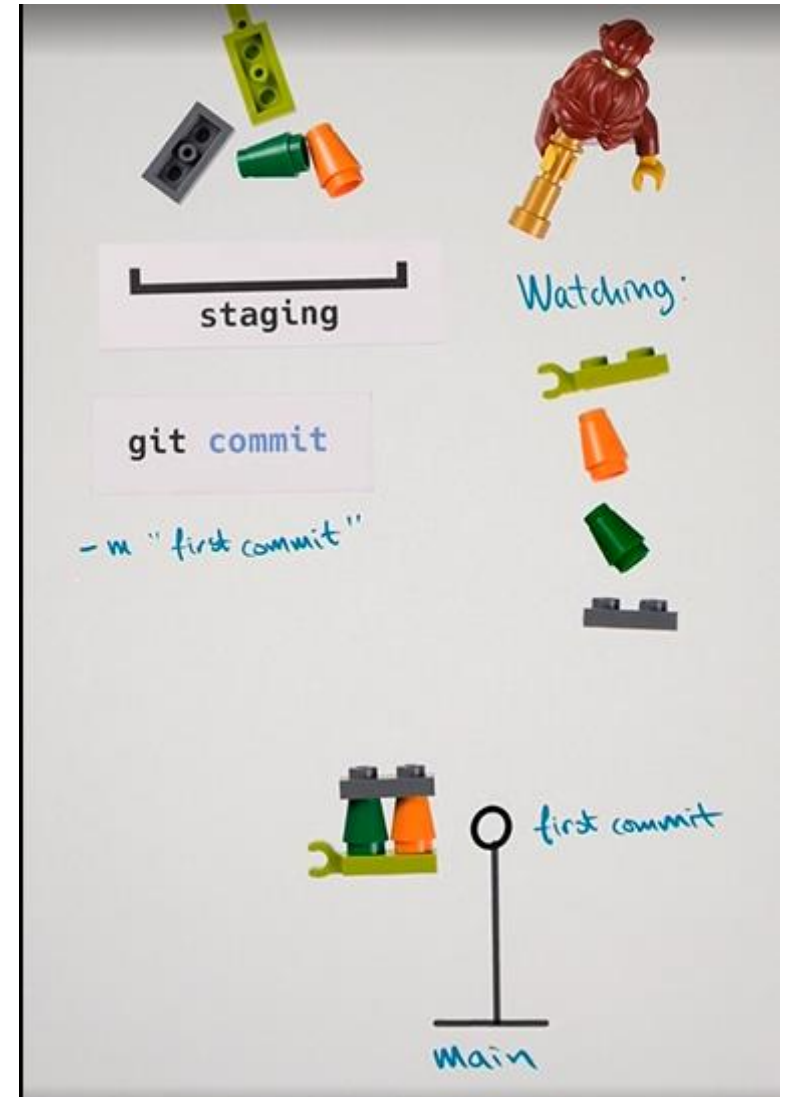
Git add: Agrega Cambios

- Los cambios que nos interesa agregar, primero se agregan al área de preparación: “staging area”.
- Estos cambios son los que vamos a querer que se vean en la “historia”



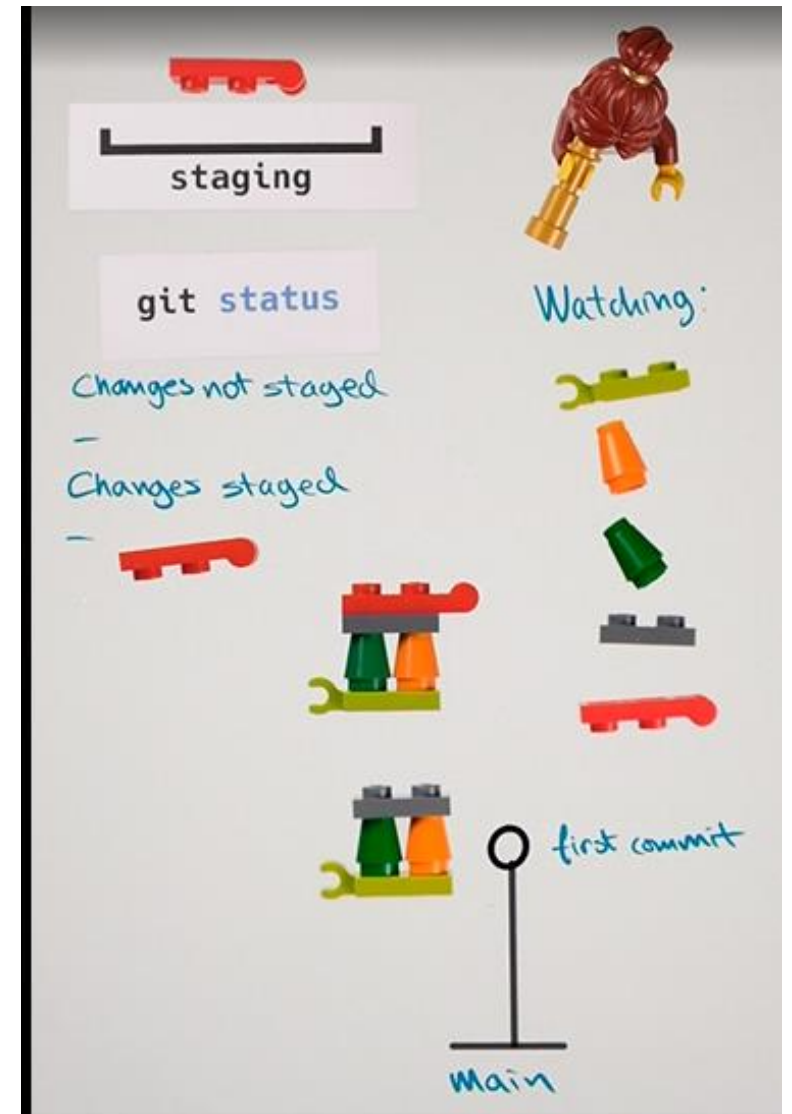
Git commit


- El commando “commit” ingresa o escribe los cambios del area de preparación a la historia del Proyecto en el branch que estamos de manera local.
- Se recomienda hacer commits “atomicos” es decir frecuentes y con cambios muy especificos.
- El message de commit ayuda an entender que se estaba haciendo



Git Status

- Obtiene el estatus de los cambios que se han dado, pero no los que se les hizo commit (estos se ven con git log --raw):
 - Cambios que estan en la zona de preparación (Stagging).
 - Y También cambios que NO estan en el stagging.



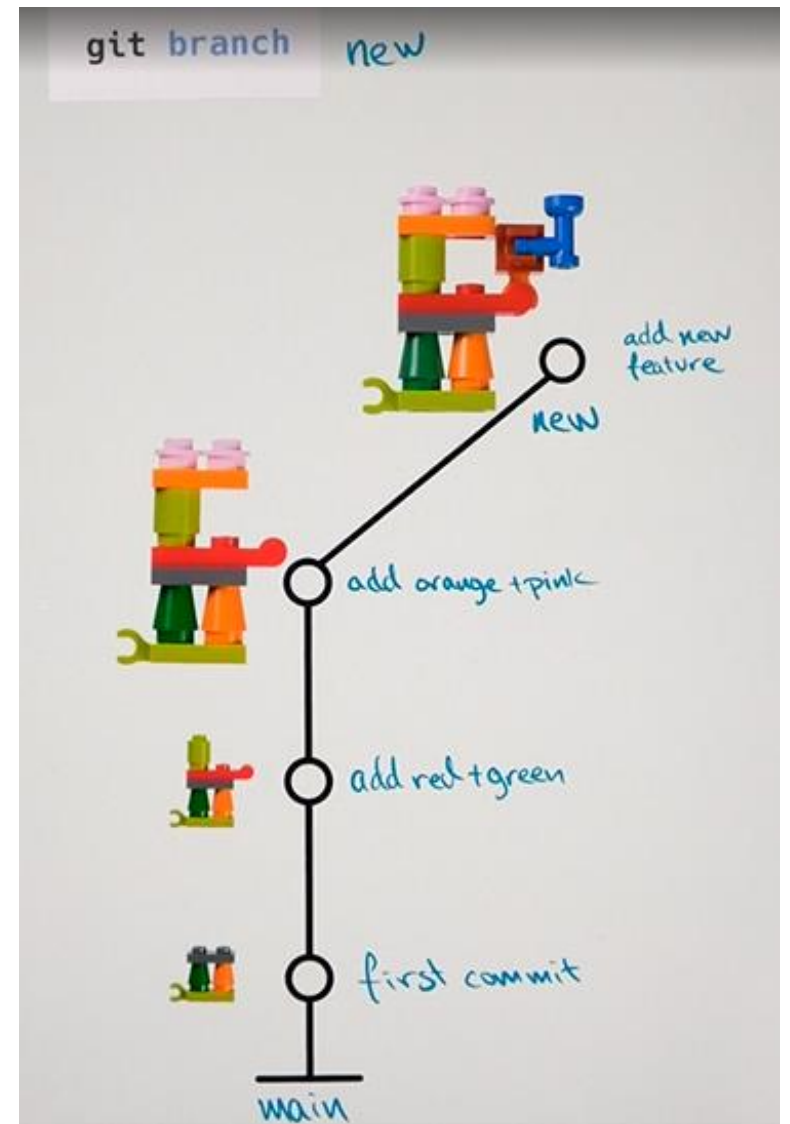


Branches: “Lineas
alternativas de tiempo”

Git branch

- Git branch
nombre_del_branch:

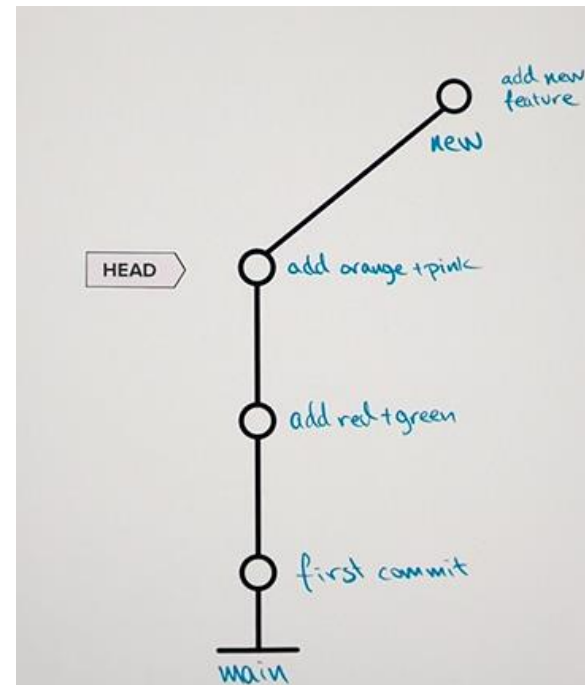
Crea una nueva linea de tiempo diferente de la principal (main/master).



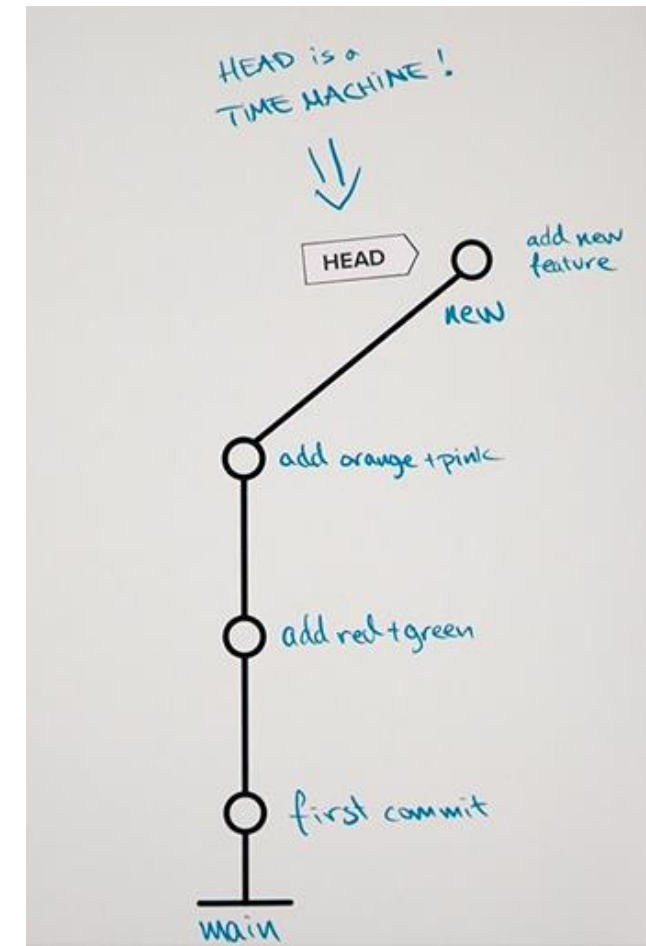
Head: Una introduccion

- Es el cambio que está a la cabeza del branch activo.
- Es decir el último cambio al que se le ha hecho commit del branch en el que estamos!

Estamos en main branch

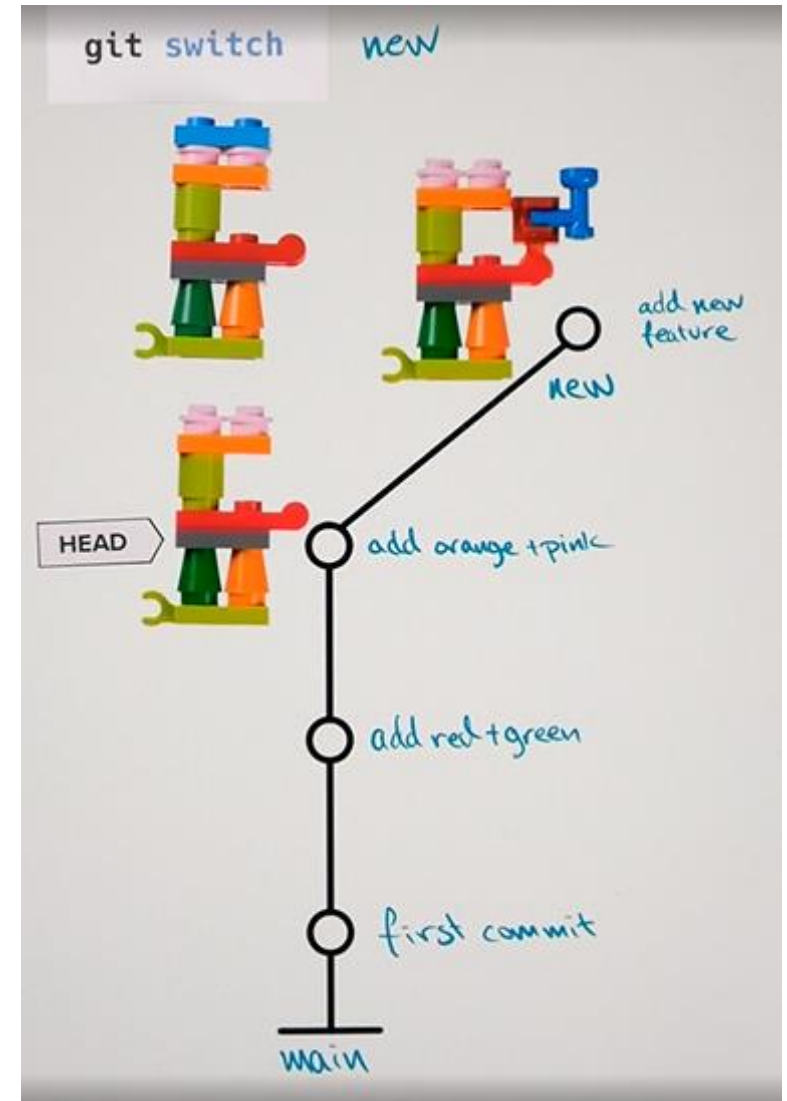


Estamos en new branch



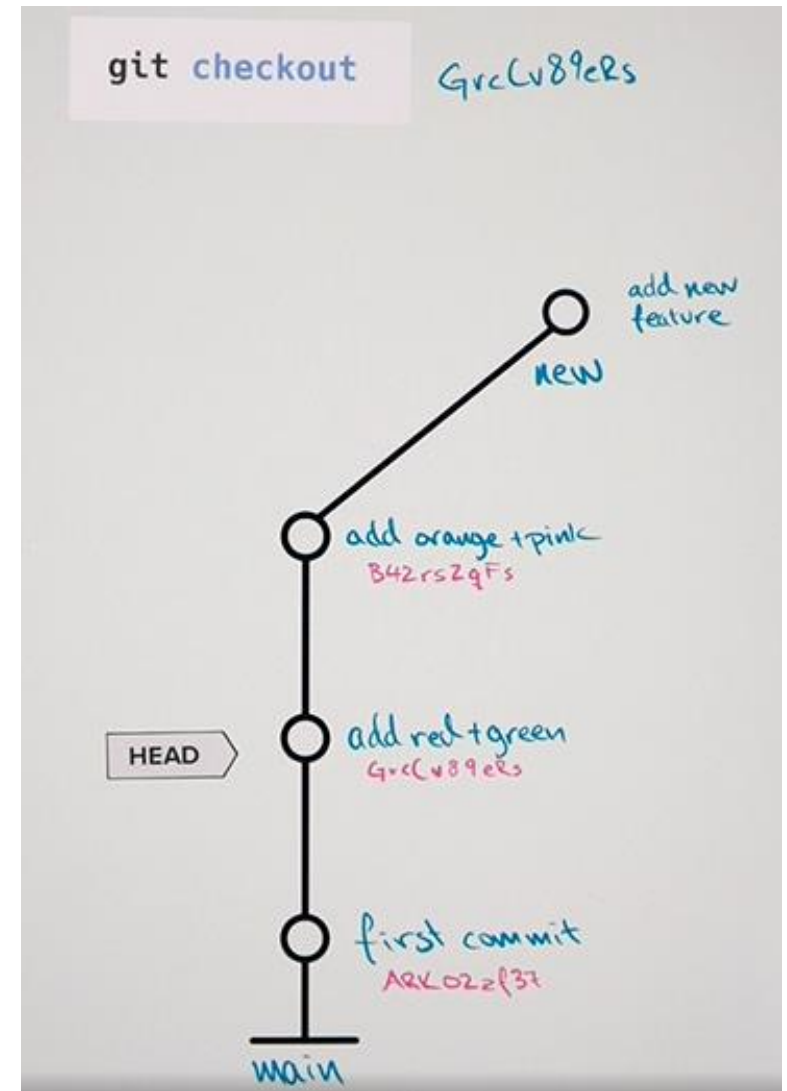
Git switch/checkout

- Cambia de un branch a otro, ejemplo para movernos del branch main al branch “new”:
 - `git switch new`
 - o `git checkout new`



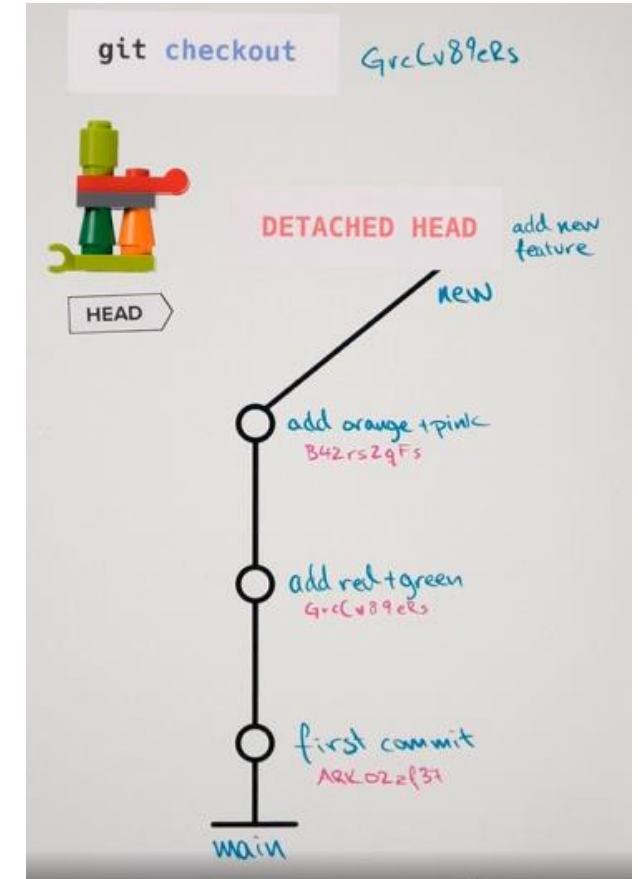
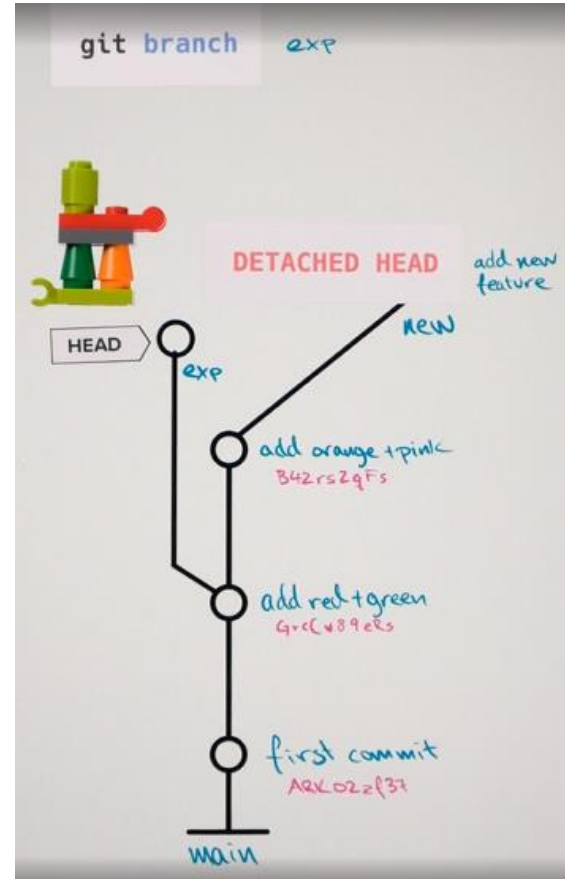
Git Checkout, más que solo cambiar de branch

- Sirve para ir al “pasado” en la historia del Proyecto:
- Ejemplo: `git checkout grccv89ers`



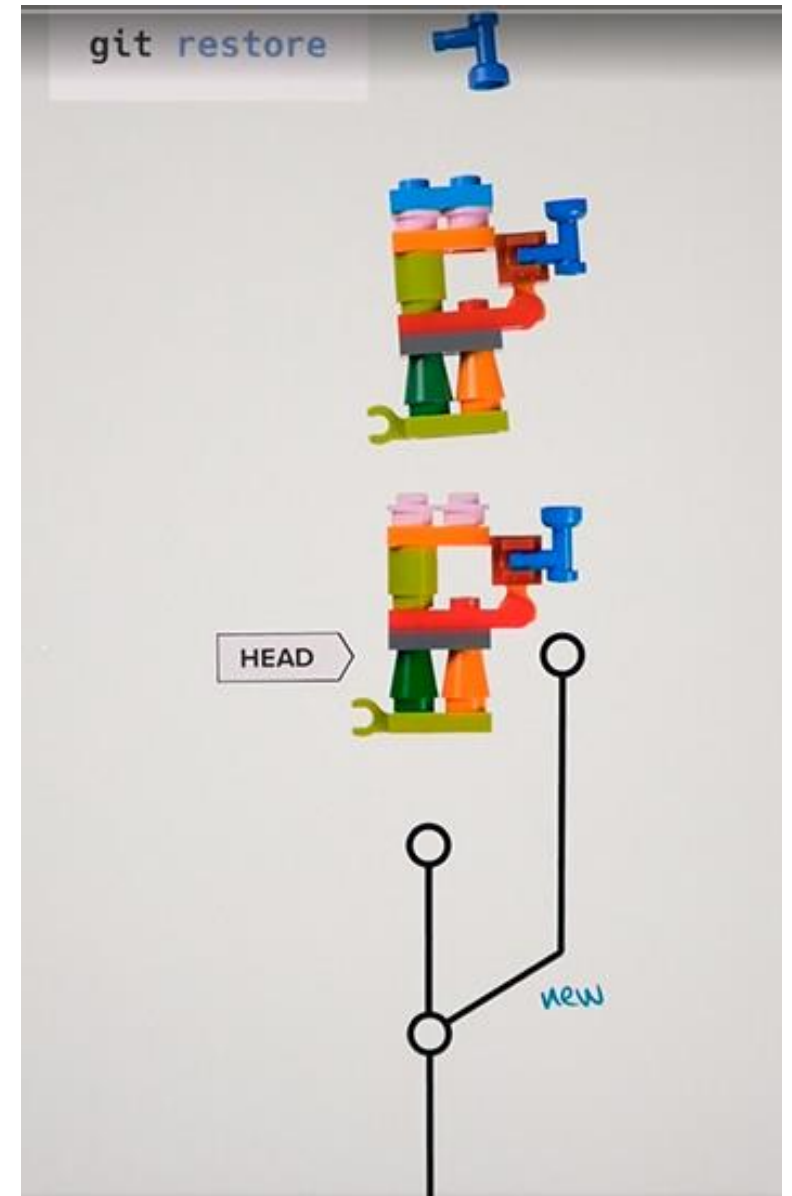
Detached Head: Una breve explicación

- Cuando se hace un checkout de un commit previo se va a mostrar un mensaje de precaución: “detached head”.
- Ya que esto podría cambiar la línea de tiempo y de cierta forma reescribir la “historia”
- Para solucionarlo: hacer un branch



git restore

- Restaura un archivo a la ultima revision que se le hizo commit
- Git checkout también se puede usar para lo mismo.
- Notese como git checkout tiene varias funcionalidades: para moverse de branch, para moverse entre commits y para restaurar un archivo al ultimo commit

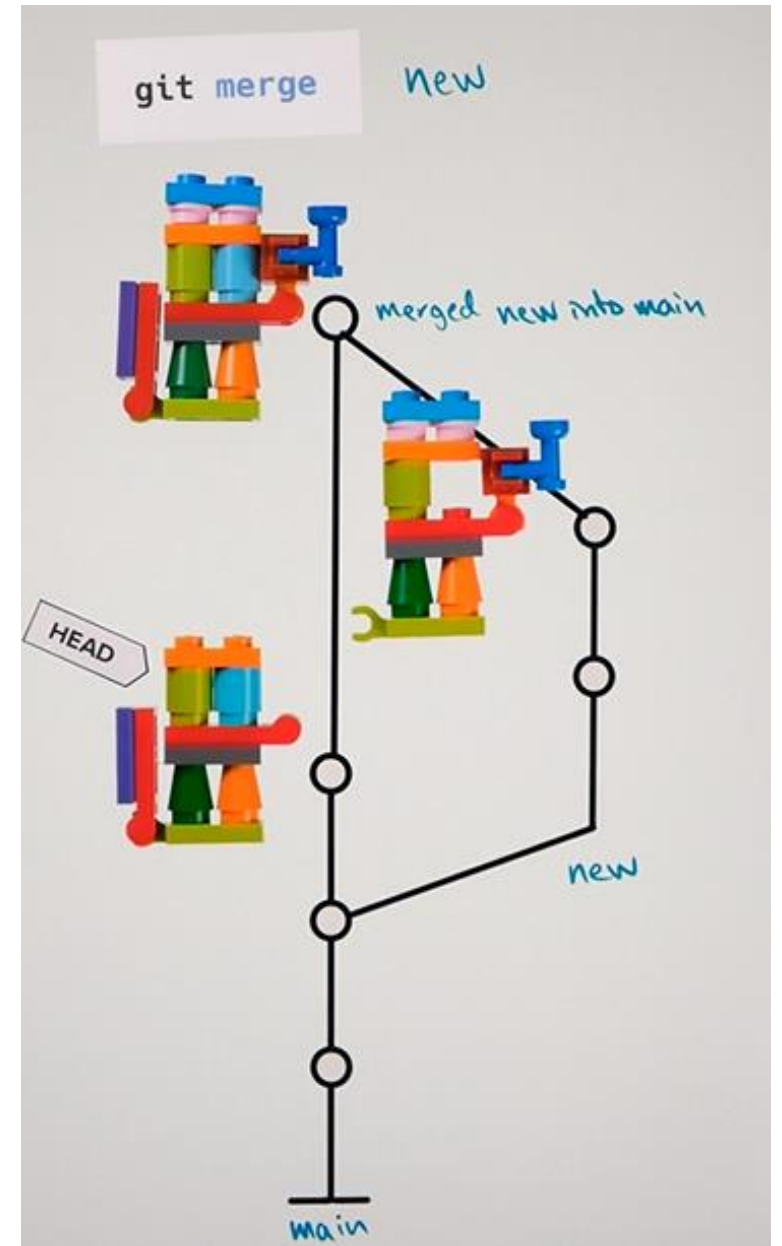




Changing History

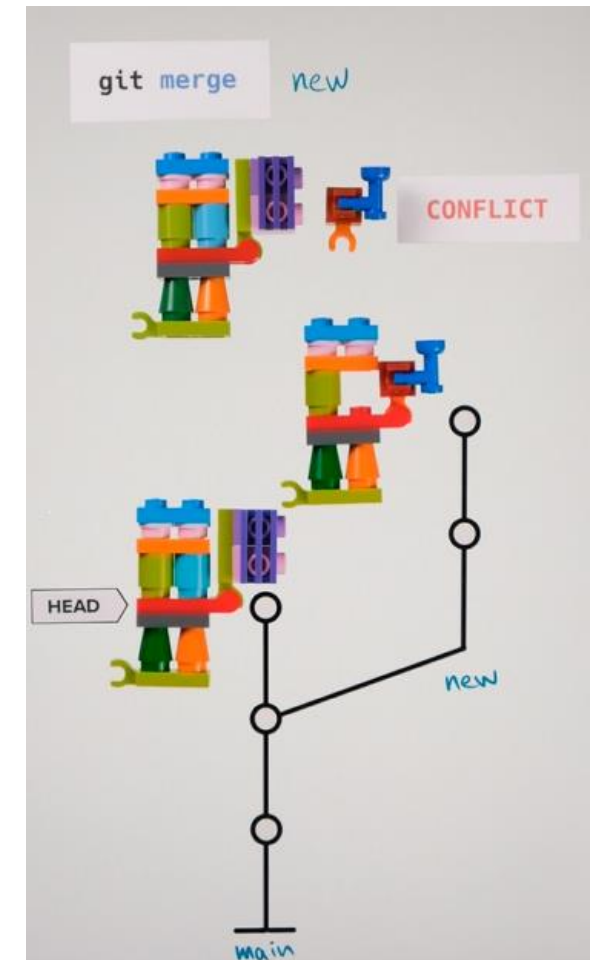
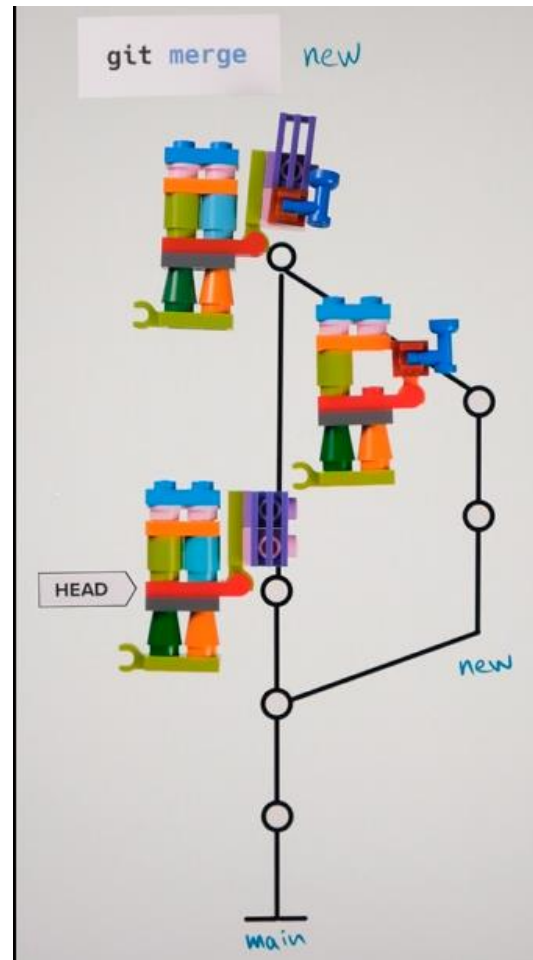
Git Merge

- Se integran/combinan los cambios de un branch a otro.
- Si estamos posicionados en main y hacemos:
- > Git merge new
- Estamos integrando los cambios de new en main.
- Después del merge, el branch experimental, en este caso “new” sigue existiendo



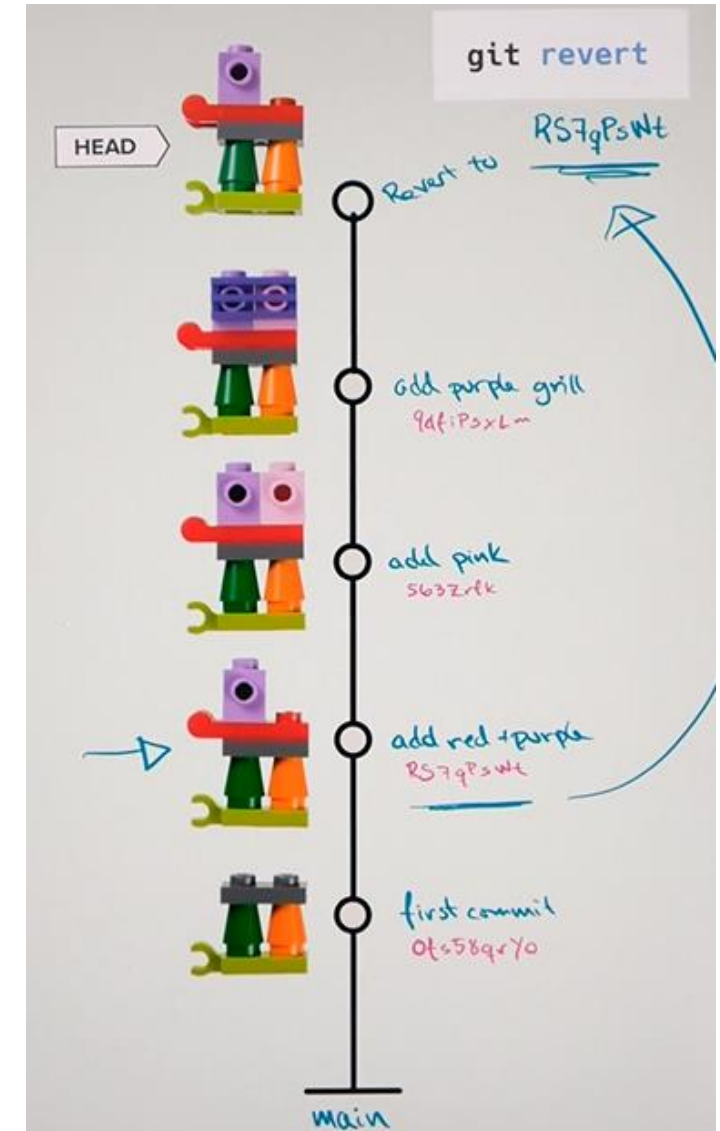
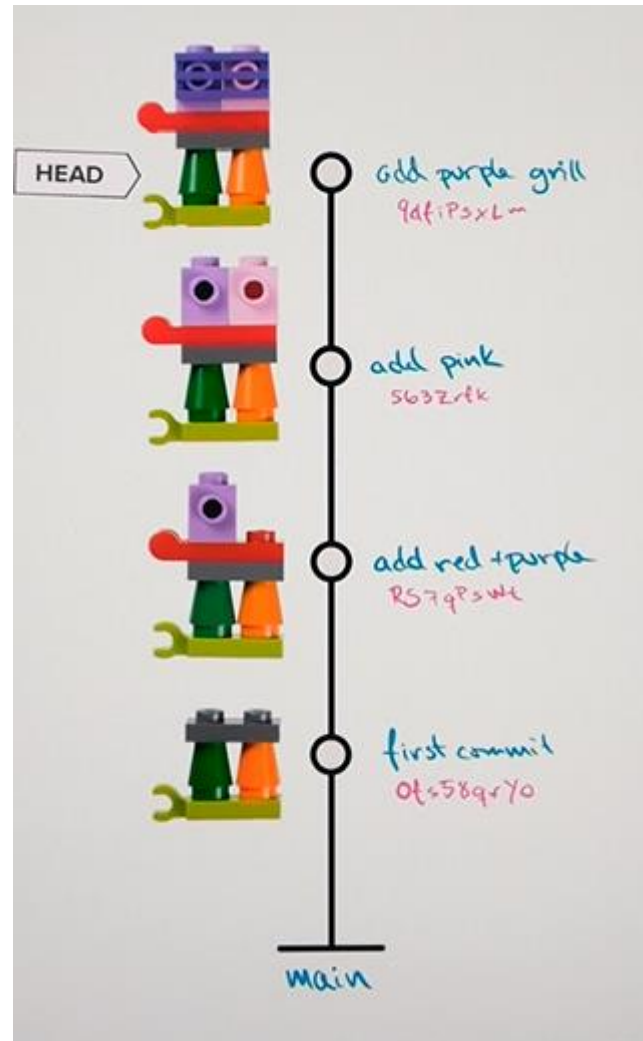
Conflictos

- Como arreglar los conflictos?
- El usuario tiene que editar y aceptar uno de los dos cambios o combinarlos en uno solo.



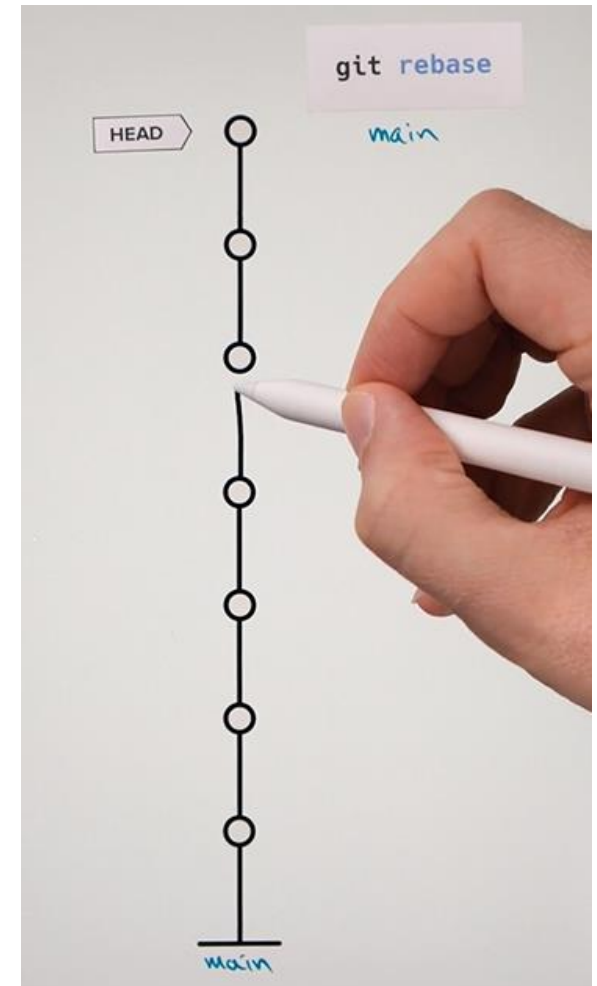
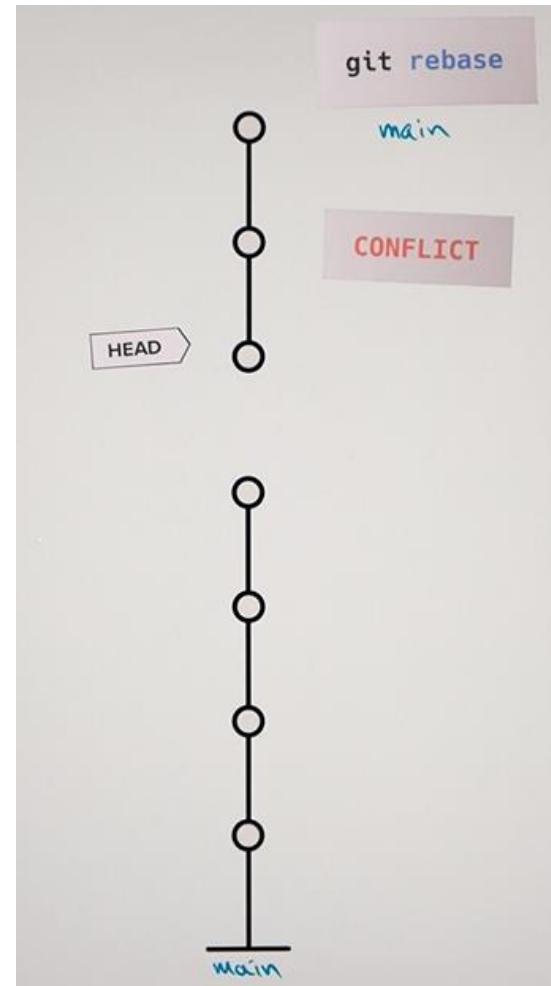
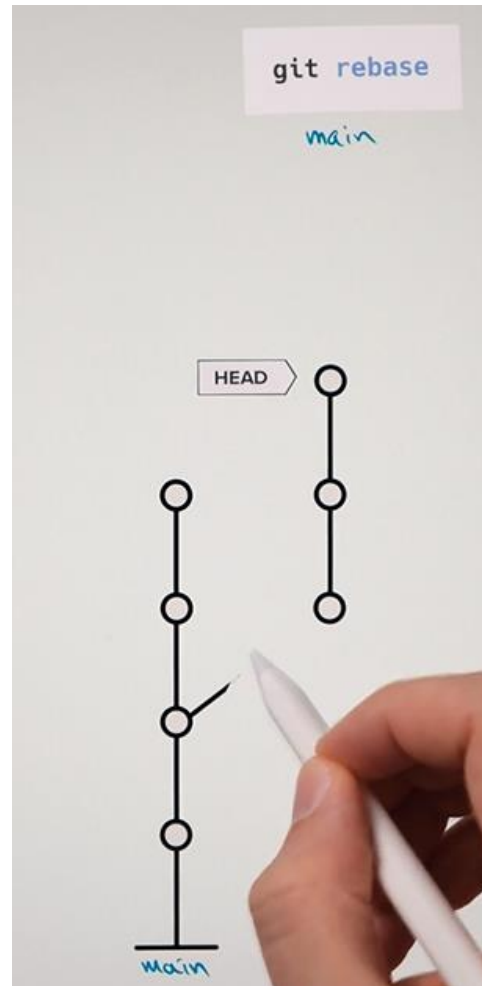
Git revert

- Revierte los cambios (deshacer/undo)
- Se necesita el id del cambio (git log)
- La ventaja de git revert es que no destruye la historia



Git rebase

- En algunos casos si se necesita reescribir la historia!





Git flow

Git Flow

- Es uno de los flujos comunes que se utilizan para manejar los repositorios con Git. Esto hace más fácil trabajar con otros:
 - Crear el “branch” para el “feature/fix”
 - Hacer los cambios
 - Integrar con master (merge)
 - Borrar el “branch” antiguo



GitHub

Porqué GitHub?

En principio GitHub es un medio de almacenamiento en la nube donde los desarrolladores de código o contenido puede trabajar en conjunto:

- Repositorio en la nube
- Desarrollo colaborativo
- Manejo de Proyectos



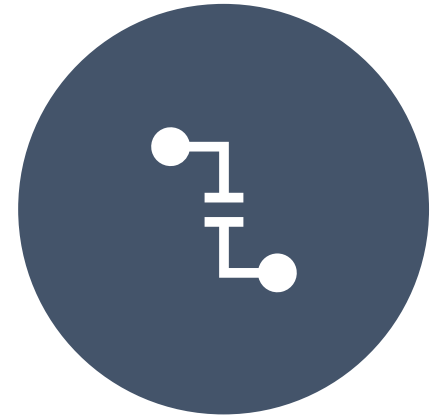
Trabajando con GitHub



SET UP REMOTE



PUSH



FETCH/PULL

Git fetch/pull

- Fetch: Sincroniza la información del repositorio local y remoto pero no hace merge de los cambios (los branches remotos se pueden ver con `git branch -a`).
- Pull: es una combinación de fetch y merge.

Git push

- Actualiza los commits locales en el repositorio remoto
 - `git push` (para el branch actual)
 - `git push -a` (para todos los branches)