GIT: UNA INTRODUCCIÓN AL CONTROL DE VERSIONES

CONTROL DE VERSIONES

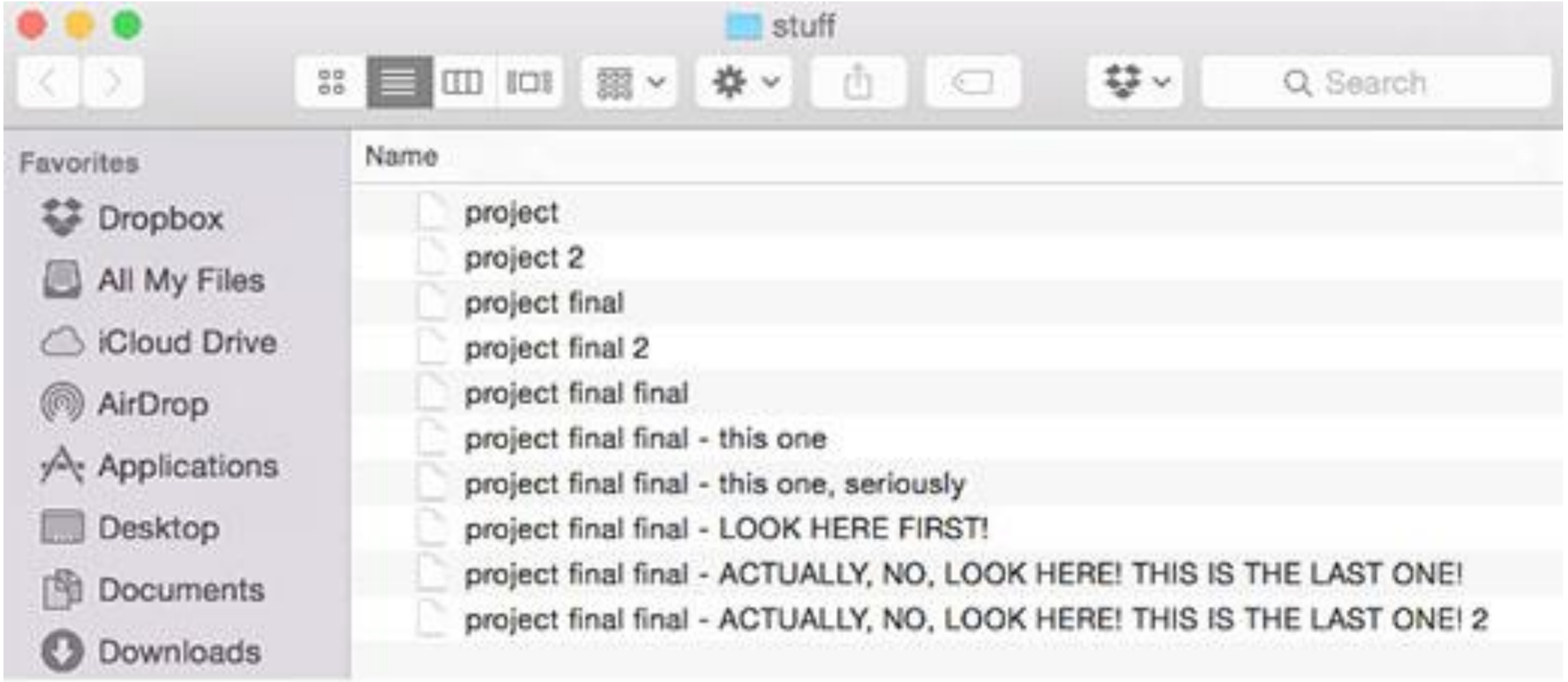
- ¿Qué es el control de versiones?
- Registra los cambios sobre un conjunto de archivos (repositorio) a lo largo del tiempo
- Nos permite
  - Revertir cambios
  - Mantener versiones paralelas
- Ver quien introdujo una variante

CONTROL DE VERSIONES

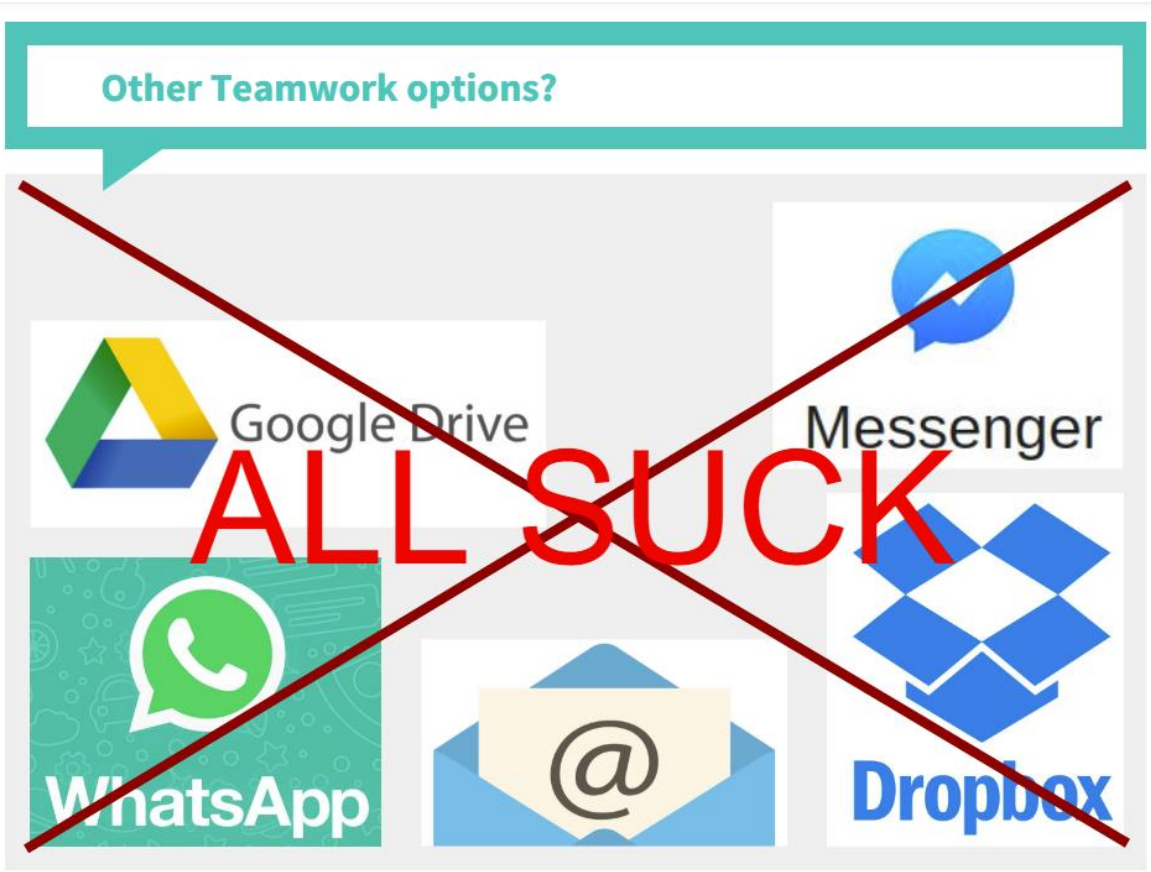
Tal vez sin saberlo ya hemos usado otros VCS

- Dropbox
- Drive
- Éste monstruo

CONTROL DE VERSIONES



CONTROL DE VERSIONES

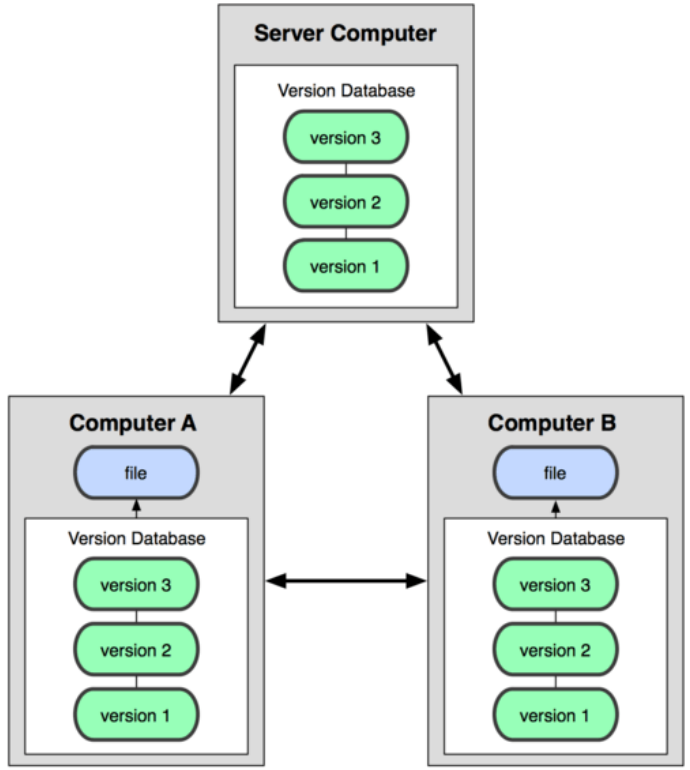


CONTROL DE VERSIONES

- Git presenta una serie de ventajas sobre estos métodos
  - En un proyecto grande, permite guardar el estado general
    - Almacena el estado conjunto de todos los archivos, y no una secuencia de individualidades
  - Es seguro
  - Muy eficiente en memoria
  - Puedo trabajar simultáneamente en dos versiones
  - Permite seguir el trabajo sobre un archivo particular, o de una determinada persona

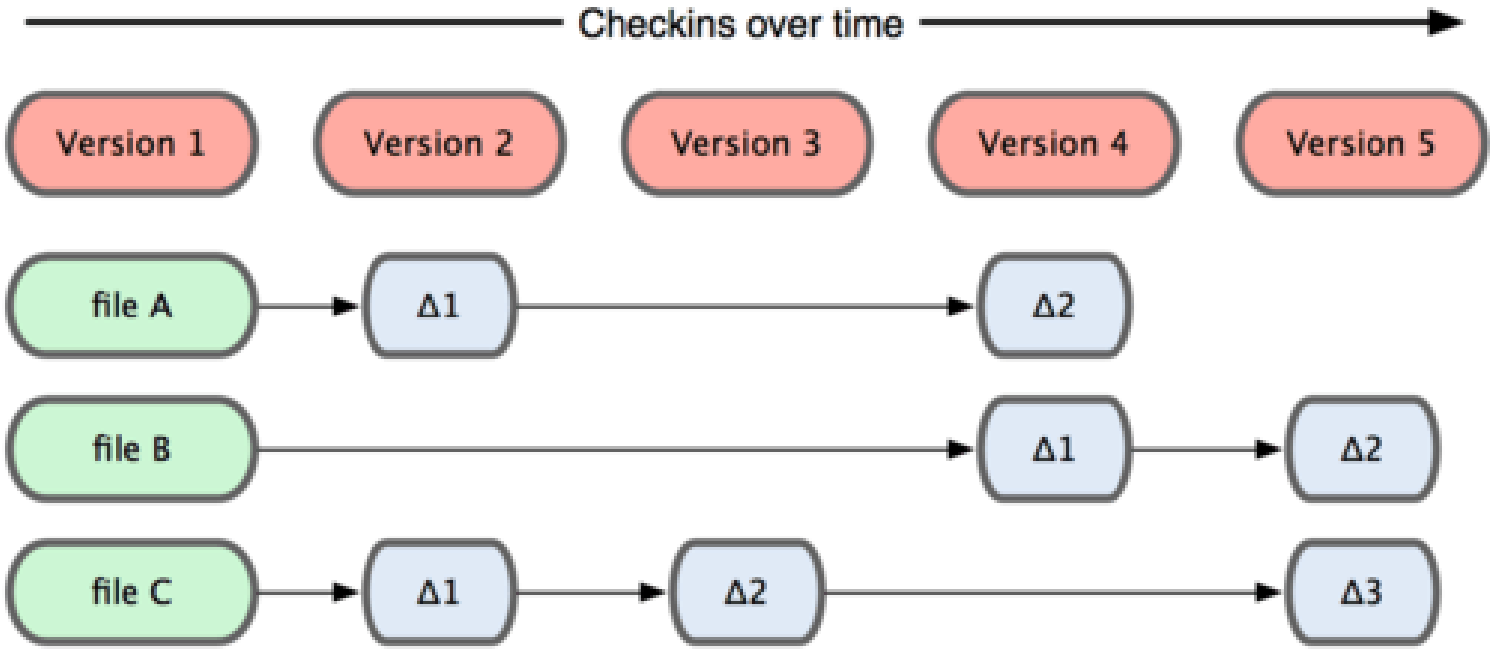
CONTROL DE VERSIONES DISTRIBUIDO

- Todos los clientes replican el repositorio
- Cada uno tiene una copia exacta del mismo
  - El hecho de tener un repositorio hosteado (GitLab, GitHub, BitBucket) no cambia el hecho de que el control sea distribuido
- Ejemplos
  - Git, Mercurial, Bazaar
- Esta arquitectura impacta directamente en la metodología de trabajo



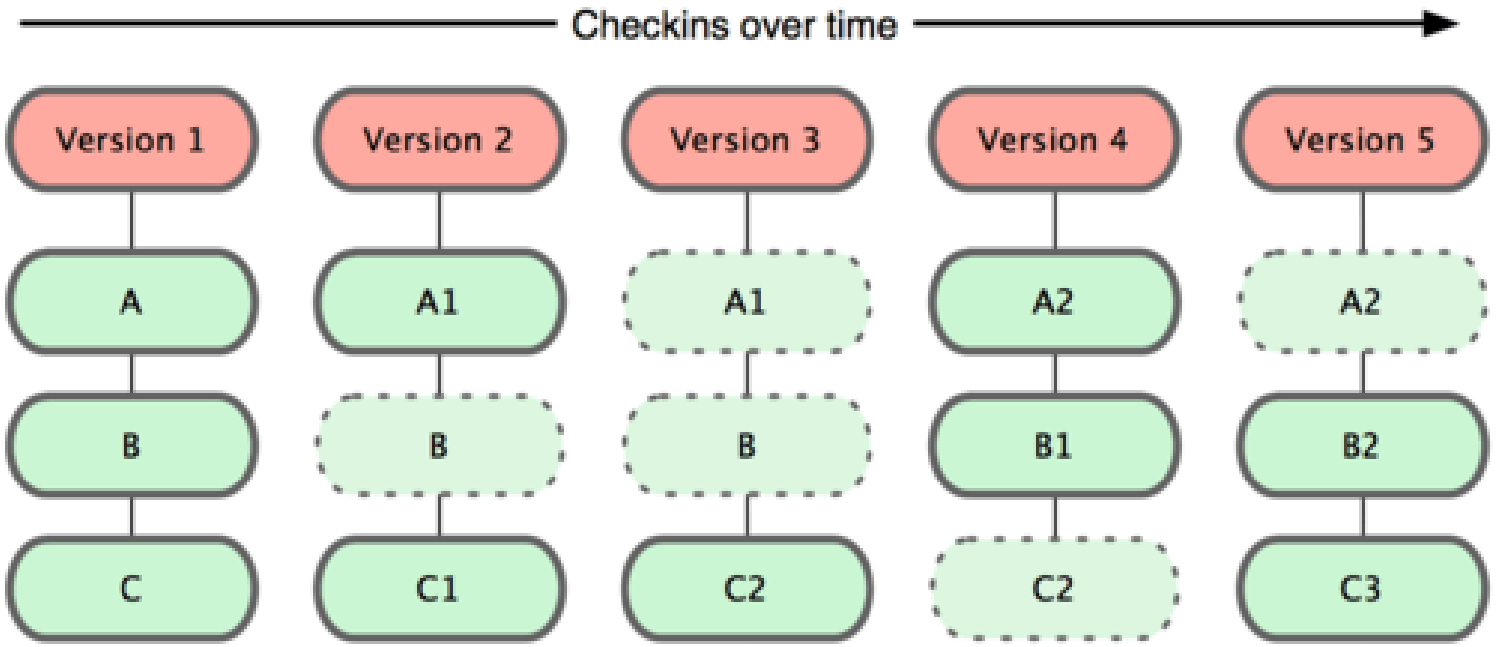
CÓMO SE ALMACENAN LOS ARCHIVOS?

- Tradicionalmente los VCS almacenan la información original y la serie de diferencias aplicadas (delta-based version control)



CÓMO SE ALMACENAN LOS ARCHIVOS?

- Git funciona distinto. Contiene todas las versiones de cada archivo y las organiza mediante punteros

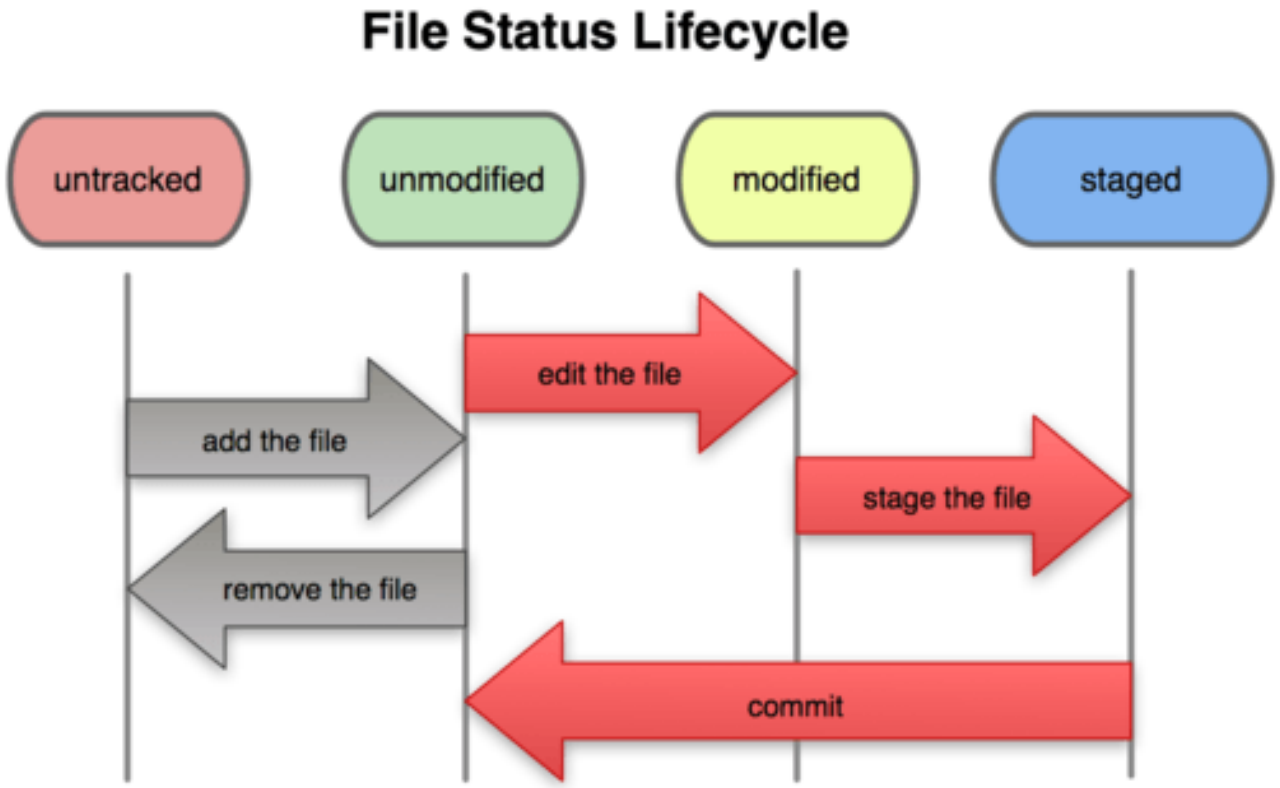


CÓMO SE ALMACENAN LOS ARCHIVOS?

- Adicionalmente, Git realiza tantas operaciones de forma local como es posible.
  - Estas características hacen que aplicar una acción en GIT sea instantáneo.
  - Además permite trabajar sin conexión a internet, sincronizando los cambios cuando sea posible
- En general, Git solo añade información
  - Es muy dificil perder información
    - Se puede! Tener especial precaución al usar la opción `--force`
  - Permite experimentar sin miedo a romper

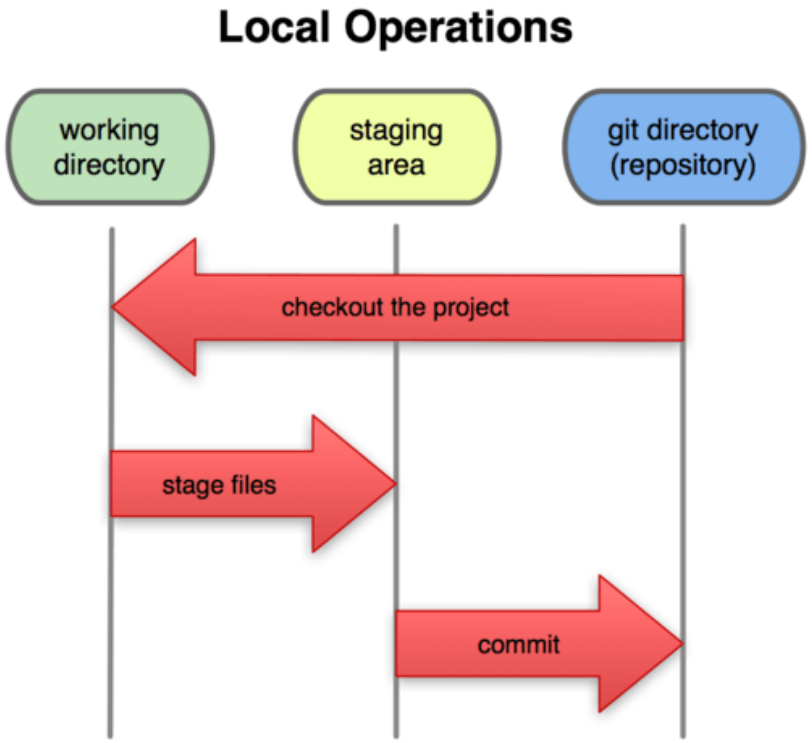
WORKFLOW DE GIT

- Cada archivo tiene tres posibles estados
  - commited (confirmado)
    - Los datos están almacenados en el repositorio local
  - modified (modificado)
    - Archivo modificado pero no agregado al repositorio local
  - staged (preparado)
    - El archivo fue modificado, y se validó para su inclusión en el próximo commit



WORKFLOW DE GIT

- El repositorio local se encuentra en una carpeta oculta dentro del raíz del Proyecto (.git)
- Toda la información histórica del proyecto se encuentra comprimida en ese lugar, así como el área de preparación (staging area)

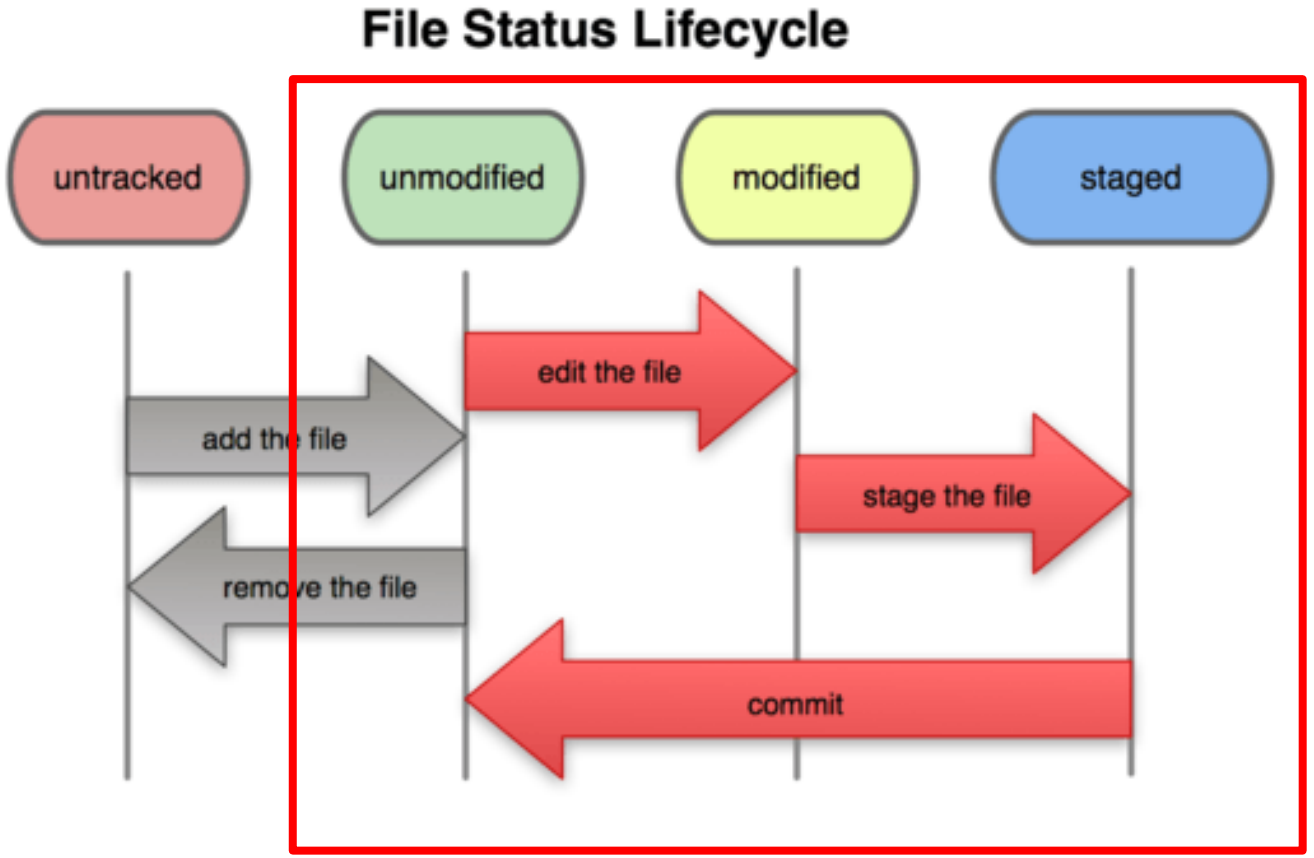


WORKFLOW DE GIT

- El flujo de trabajo básico de GIT es el siguiente
  1. Se modifican una serie de archivos en el directorio de trabajo
  2. Se preparan los archivos, agregándolos al staging area
     `git add [files]`
  3. Se confirman los cambios, almacenándolos en el directorio de Git
     `git commit –m [mensaje de commit]`
     Los mensajes de commit siempre deben describir la funcionalidad agregada por el código nuevo (e.g. "Agrega soporte para periféricos I2C")

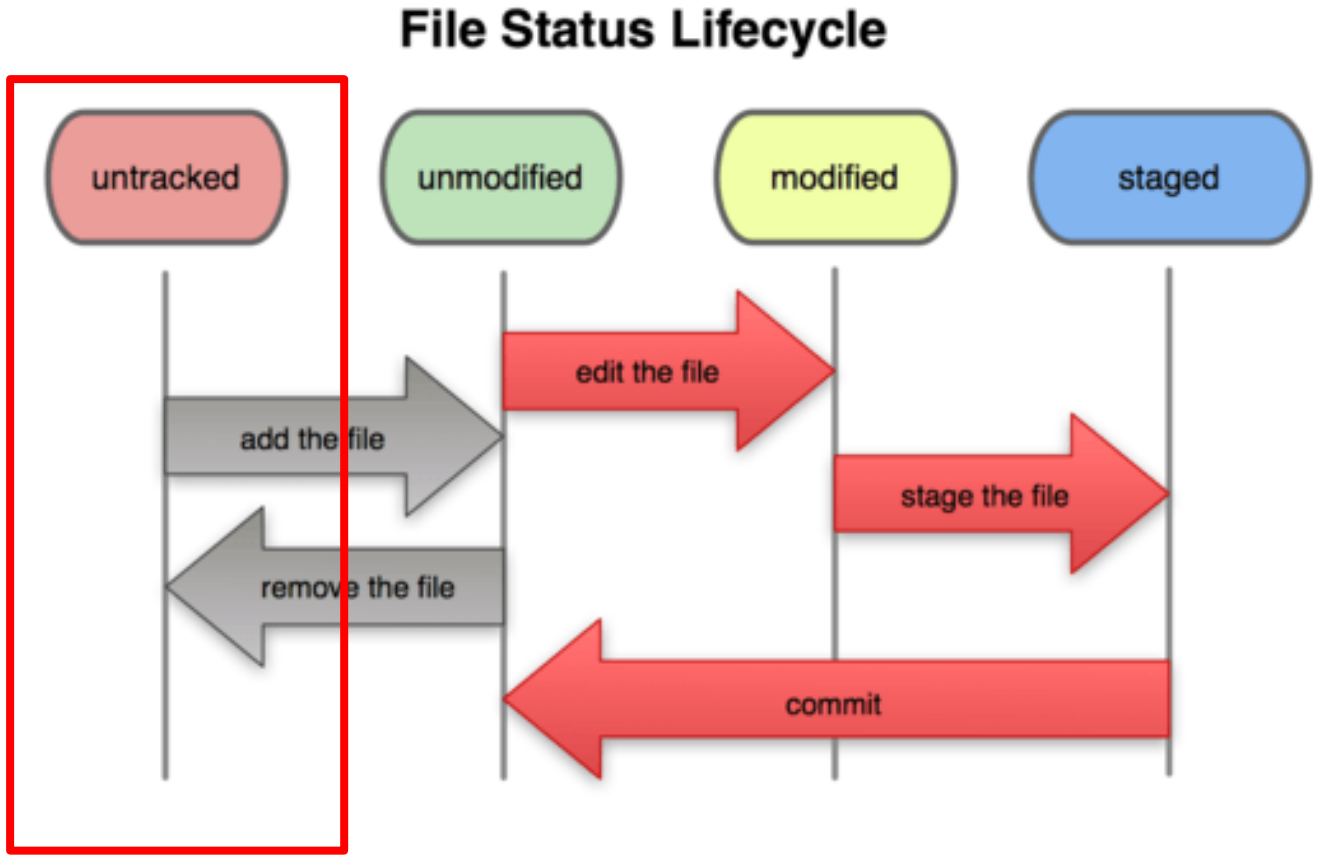
CICLO DE VIDA DE LOS ARCHIVOS

- Los archivos pueden tener dos estados:
  - **Tracked**
    - Existen en la última imagen del repo
    - Los achivos con seguimiento (tracked) pueden estár sin modificar, modificados o preparados.
    - Un archivo modificado puede pasarse al stage, y cuando se cierra un commit pasa a estar no modificado (cambia la imagen en el repo).
  - Untracked
    - No existen en la última imagen



CICLO DE VIDA DE LOS ARCHIVOS

- Los archivos pueden tener dos estados:
  - Tracked
    - Existen en la última imagen del repo
    - Los achivos con seguimiento (tracked) pueden estár sin modificar, modificados o preparados.
    - Un archivo modificado puede pasarse al stage, y cuando se cierra un commit pasa a estar no modificado (cambia la imagen en el repo).
  - **Untracked**
    - No existen en la última imagen



CICLO DE VIDA DE LOS ARCHIVOS

- Existen muchos tipos de archivos que nunca deben ser monitoreados por GIT (tracked)
  - En general son archivos que se crean a partir del código fuente de la aplicación
  - .o, ejecutables (.exe), binarios (.bin, elf), html de Doxygen...
- Para esto Git cuenta con .gitignore, un archivo en el que se definen reglas para ignorar otros archivos
  - .gitignore acepta sintaxis glob estandar

RAMIFICACIONES

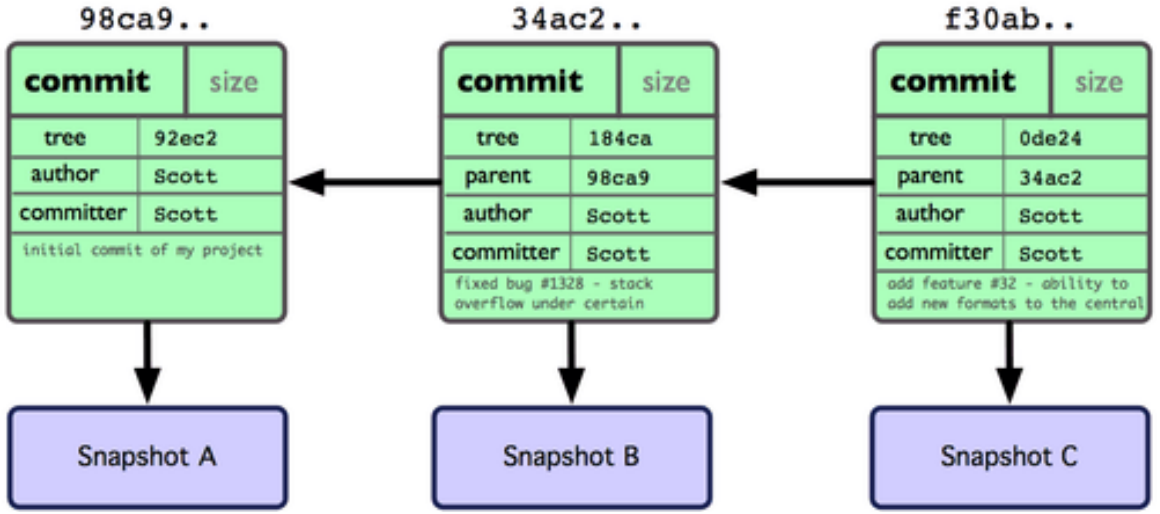
- Definimos como una ramificación (branch) un desarrollo paralelo desde el flujo de trabajo principal
- En cualquier equipo de desarrollo moderno, las nuevas funcionalidades se desarrollan paralelamente en branches para luego ser integradas al flujo principal (master)
- El sistema de ramificaciones es el punto fuerte de Git
- Intrínsecamente ligado a metodologías de desarrollo modernas (Agile)

RAMIFICACIONES

- Como ya dijimos anteriormente, Git almacena copias completas de todos los estados de todos los archivos, y organiza las versiones del repositorio mediante punteros.
  - Cada commit contiene
    - Punteros a los archivos contenidos en el commit
    - La organización en directorios de los archivos contenidos
    - Metadata del autor, instante de creación del commit, mensaje asociado…
    - Punteros al commit anterior (padre). Pueden ser múltiples

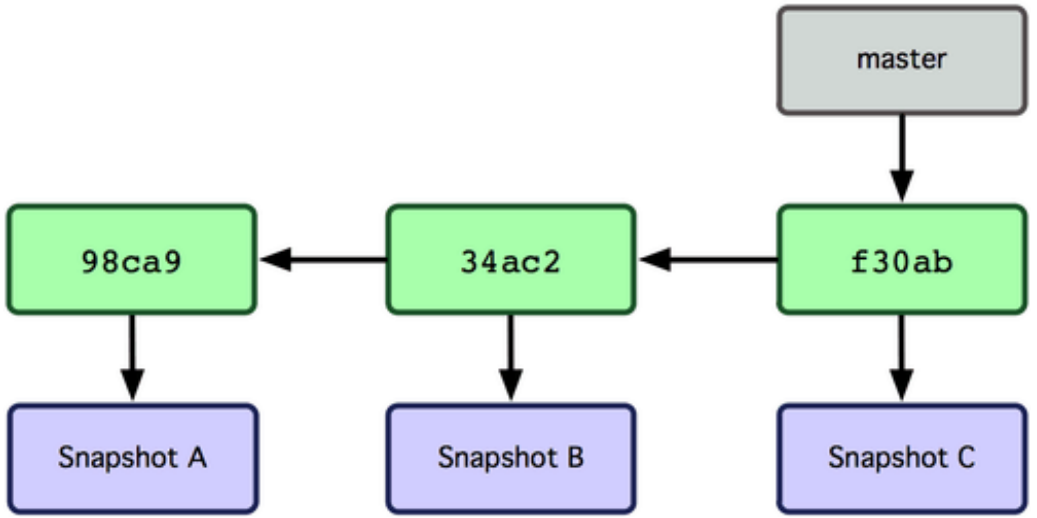
RAMIFICACIONES

- El branch por defecto es master
- Los branches avanzan automáticamente con los commits
- No hay que confundir el nombre de un branch con el hash ID de un commit



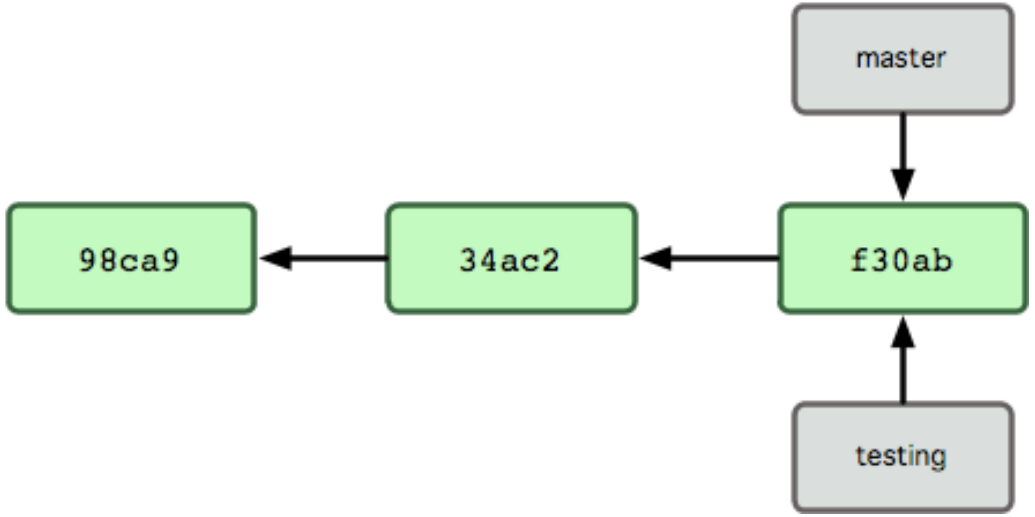
RAMIFICACIONES

- Que pasa cuando creamos una nueva rama?



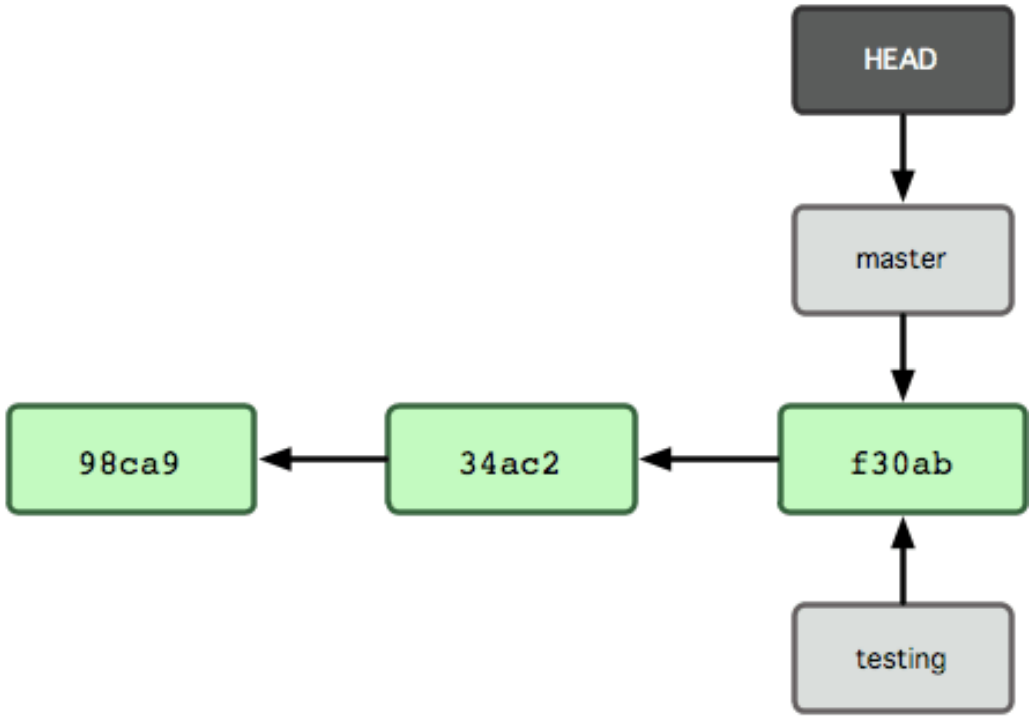
RAMIFICACIONES

- Que pasa cuando creamos una nueva rama?
  - Git crea un nuevo puntero al branch actual
- Como sabe Git cual es el branch actual?



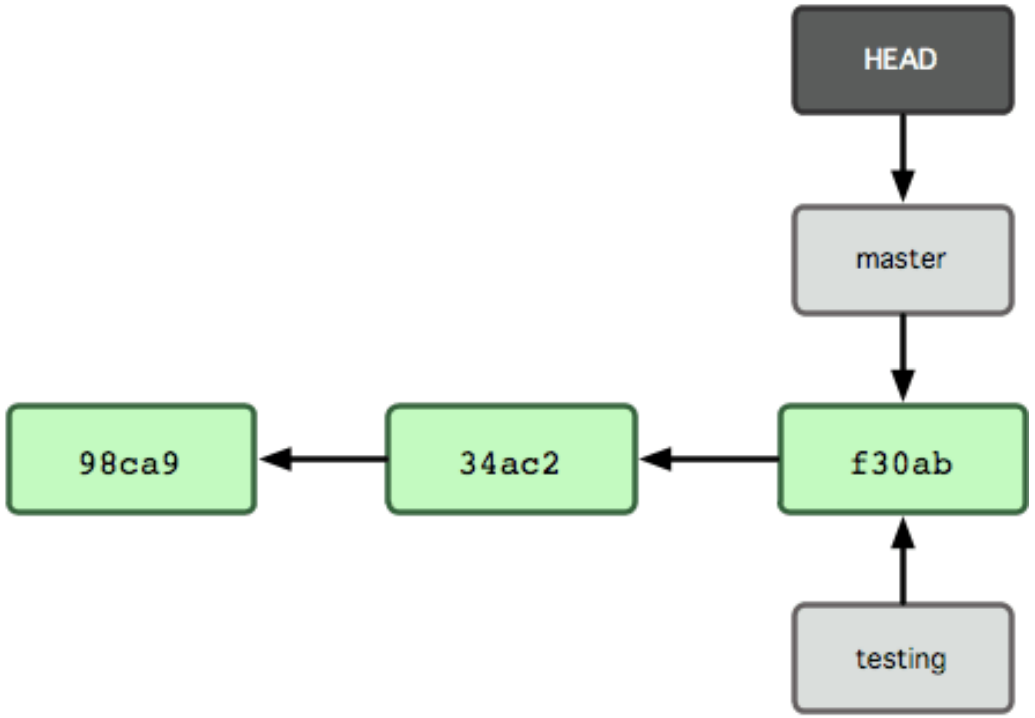
RAMIFICACIONES

- Que pasa cuando creamos una nueva rama?
  - Git crea un nuevo puntero al branch actual
- Como sabe Git cual es el branch actual?
  - Mediante un puntero especial, denominado HEAD



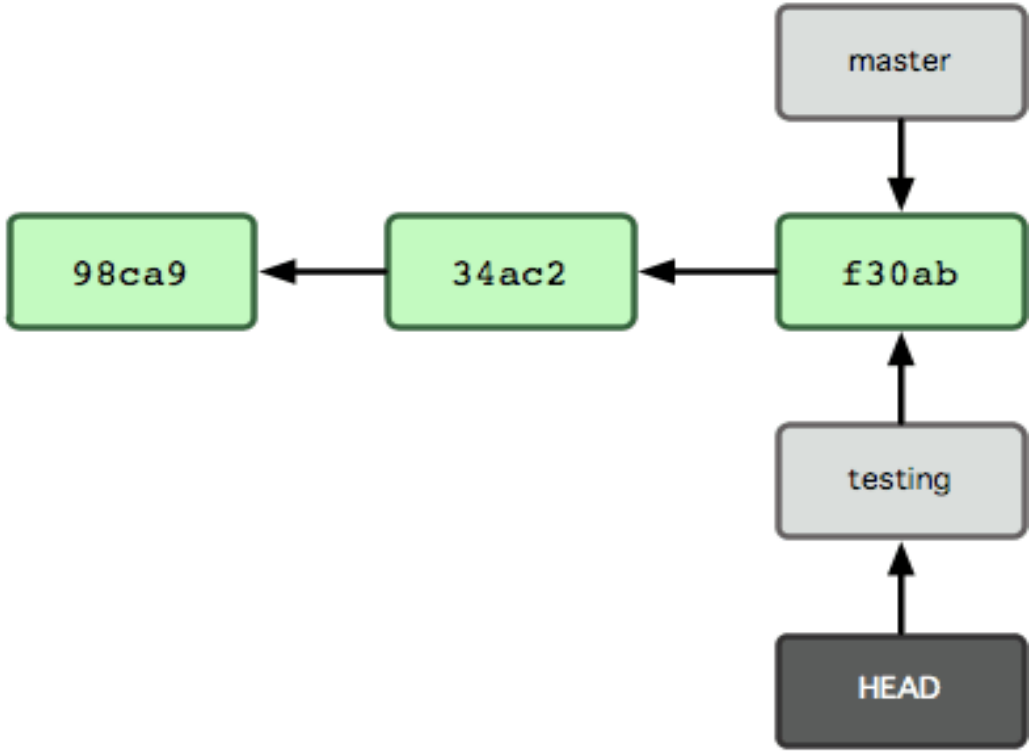
RAMIFICACIONES

- Cuando creamos un branch, Git no cambia de rama, solamente la crea.
  `git branch testing`



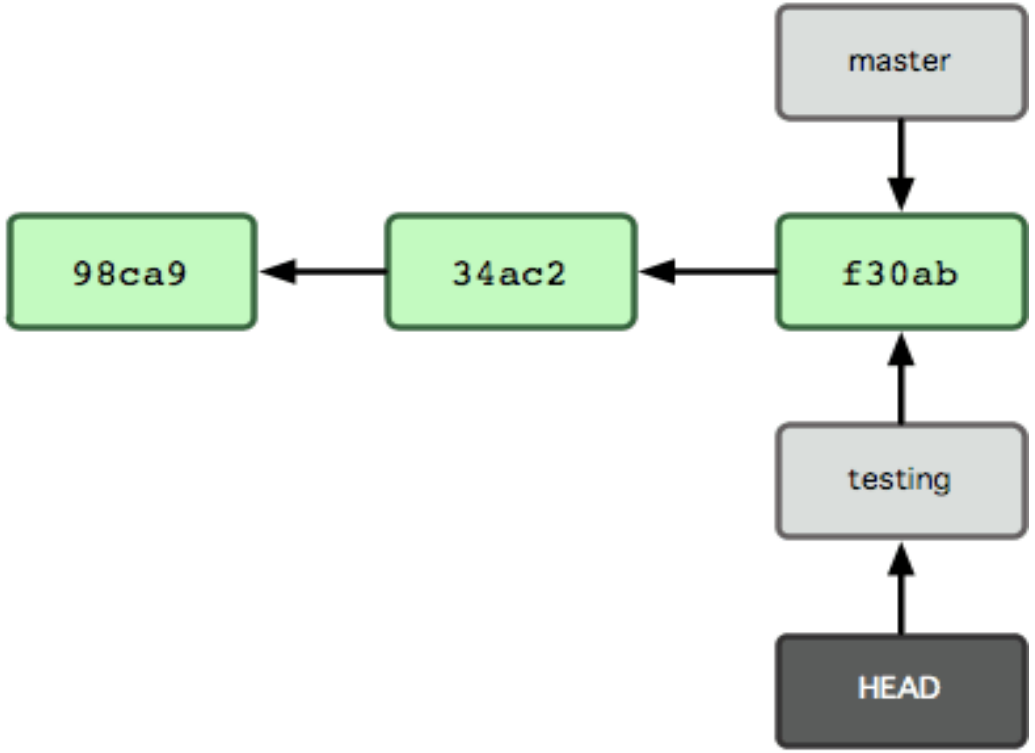
RAMIFICACIONES

- Cuando creamos un branch, Git no cambia de rama, solamente la crea.
- El comando checkout nos hace cambiar de rama
  `git checkout testing`



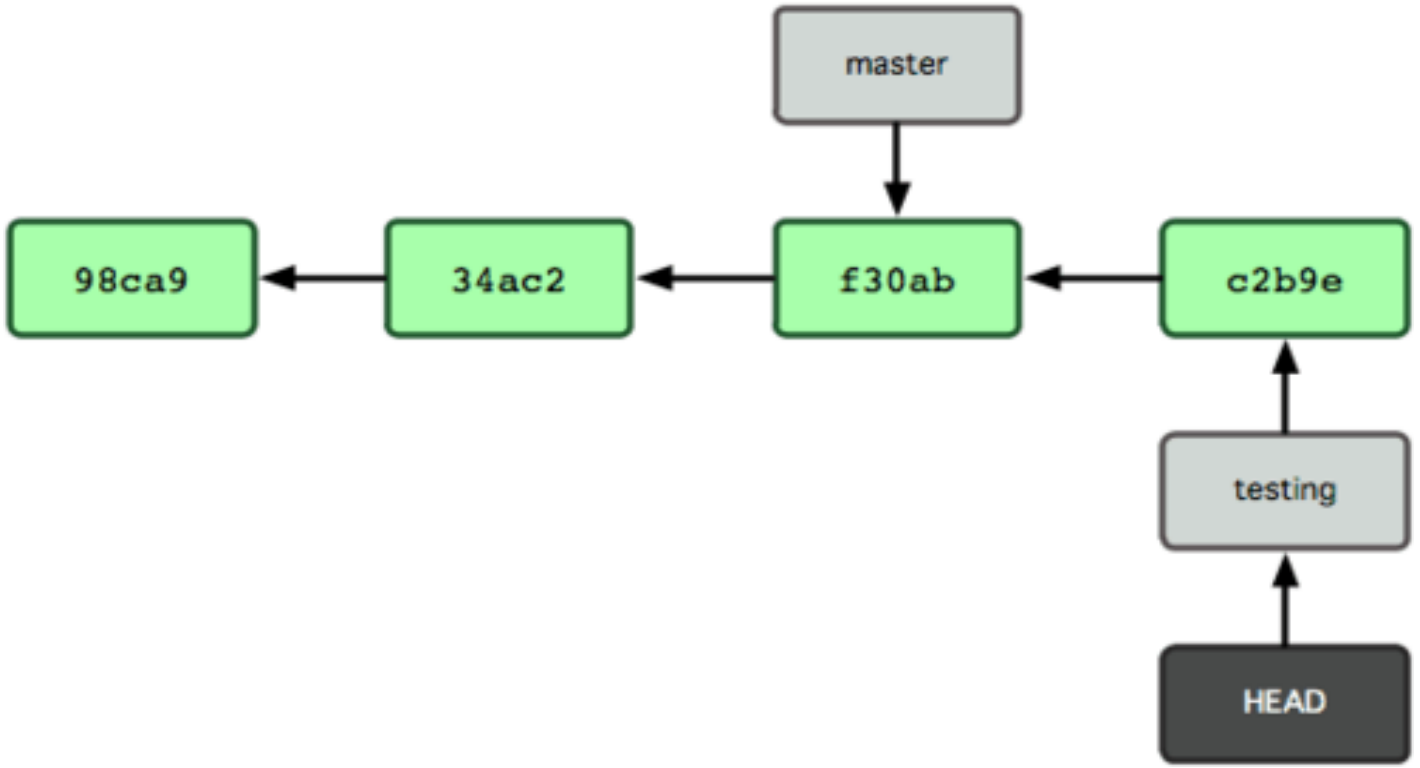
RAMIFICACIONES

- Que pasa si hacemos un commit en testing?



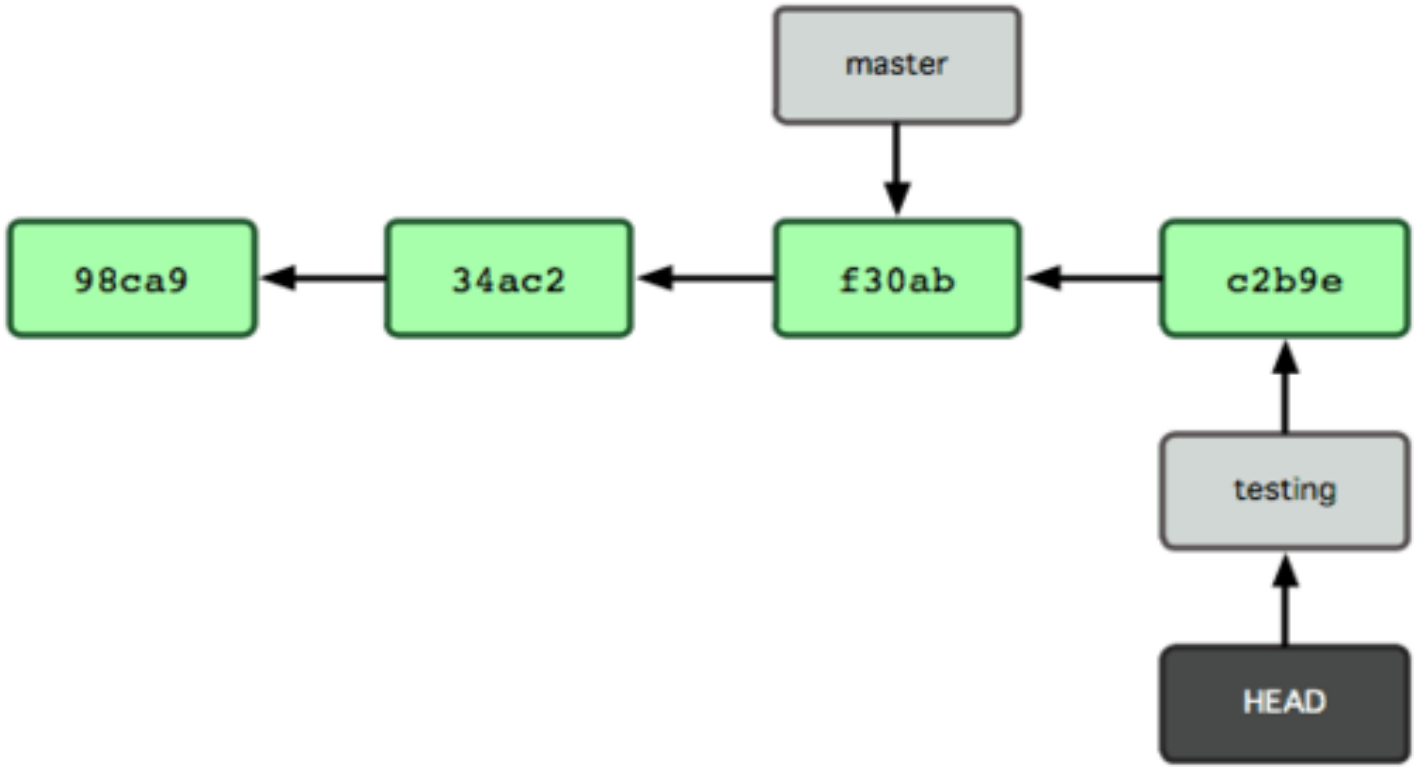
RAMIFICACIONES

- Que pasa si hacemos un commit en testing?



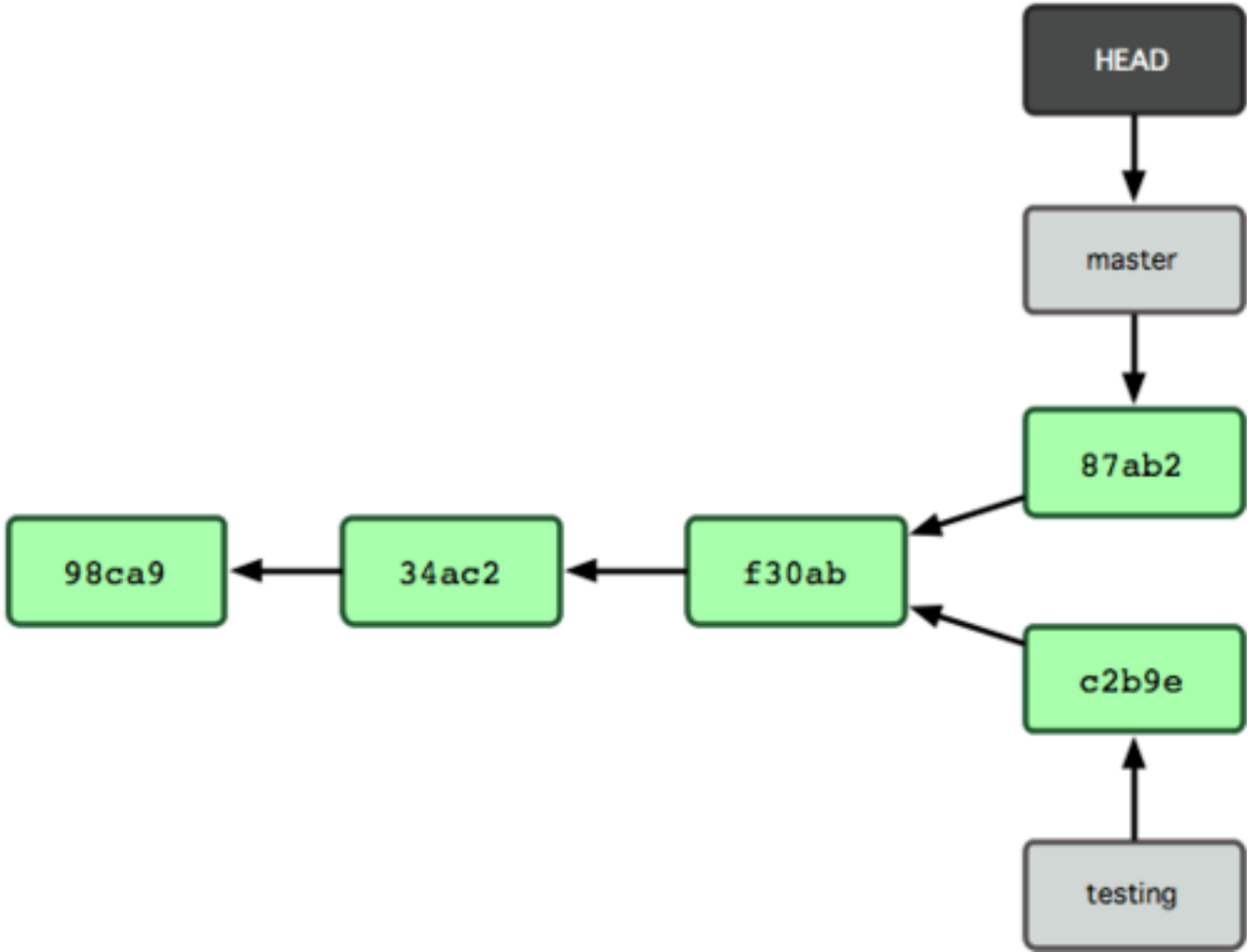
RAMIFICACIONES

Y si hacemos lo mismo en master?



RAMIFICACIONES

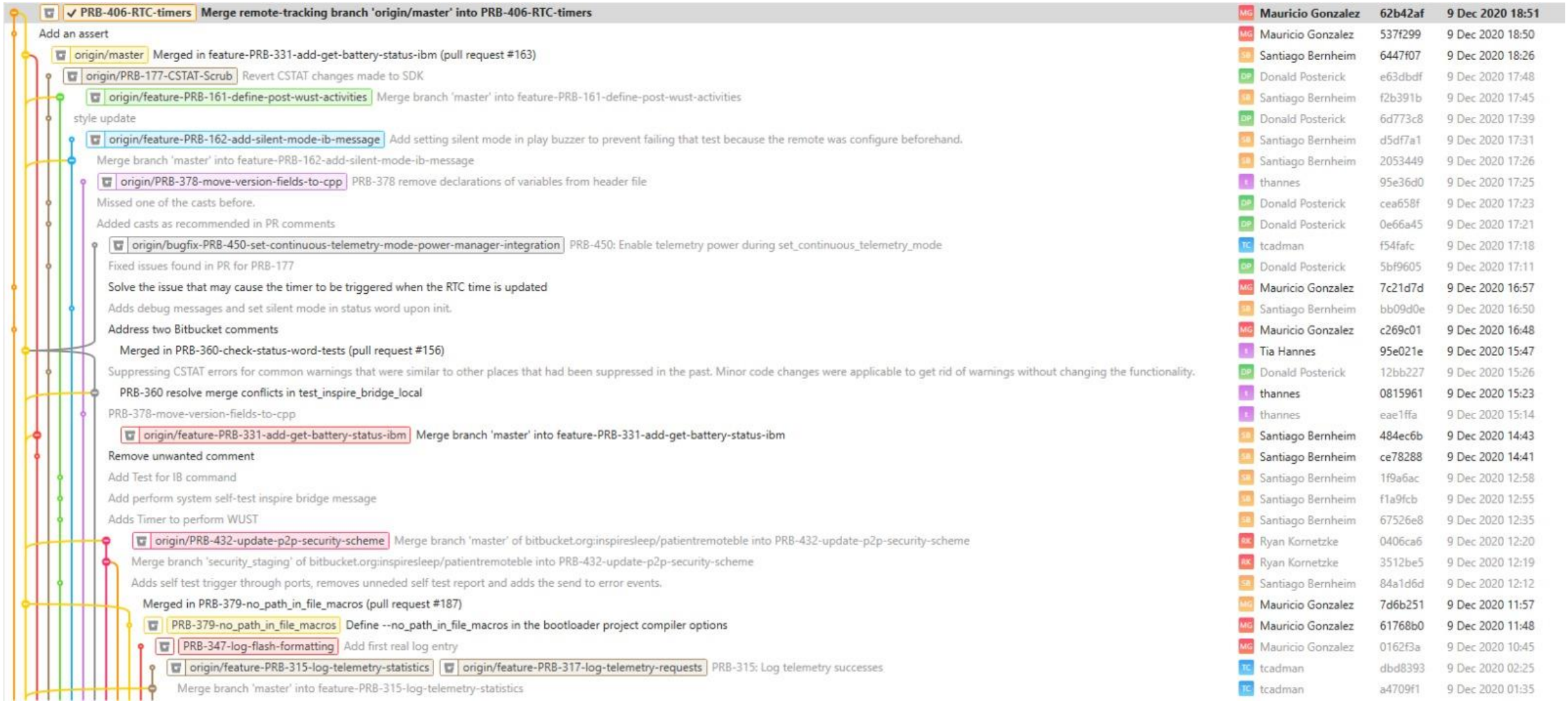
Y si hacemos lo mismo en master?



RAMIFICACIONES

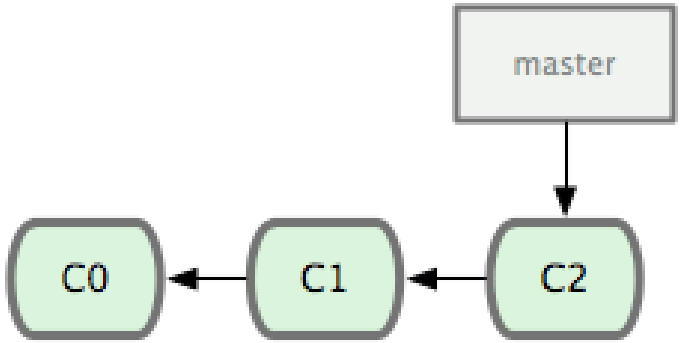
- En Git es sumamente rápido y sencillo cambiar de branch, ya que solamente se necesita una nueva asignación de punteros
- Esta capacidad motiva a los desarrolladores a crear código divergente
- Hay utilidades para fusionar branches que han crecido de manera independiente

RAMIFICACIONES: EJEMPLO REAL



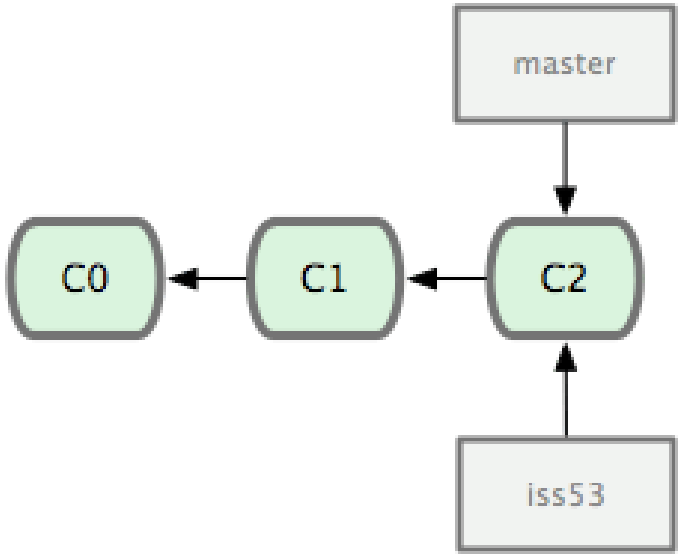
RAMIFICACIONES: EJEMPLO PRÁCTICO

- Estamos trabajando en un proyecto



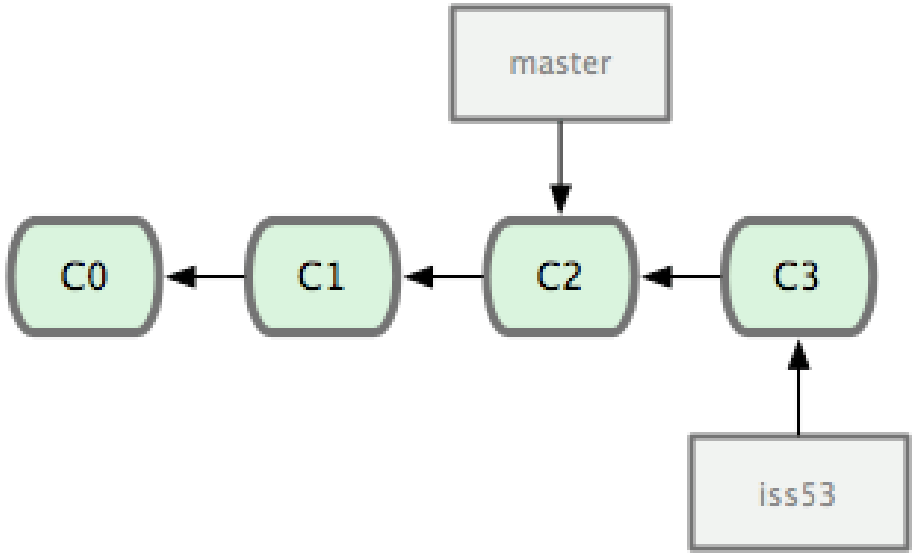
RAMIFICACIONES: EJEMPLO PRÁCTICO

- Empezamos a trabajar en una nueva funcionalidad



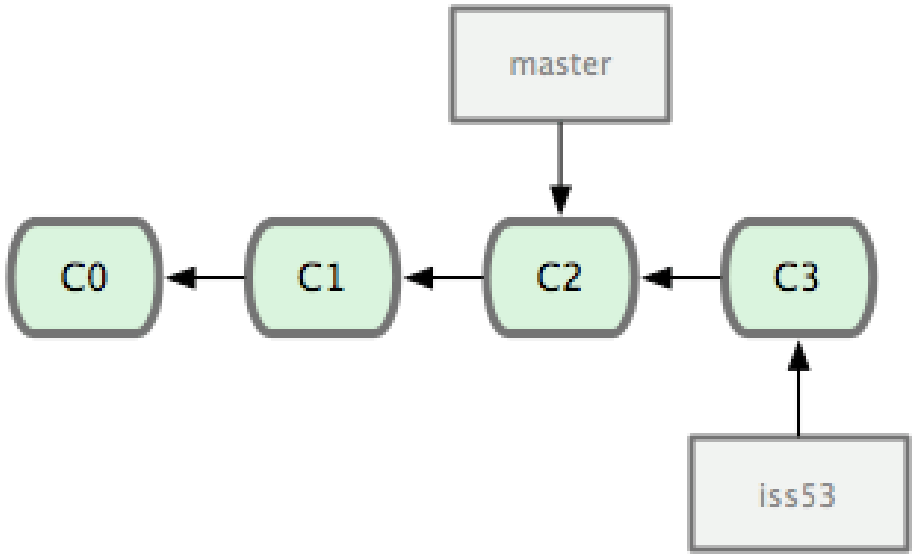
RAMIFICACIONES: EJEMPLO PRÁCTICO

- Empezamos a trabajar en una nueva funcionalidad



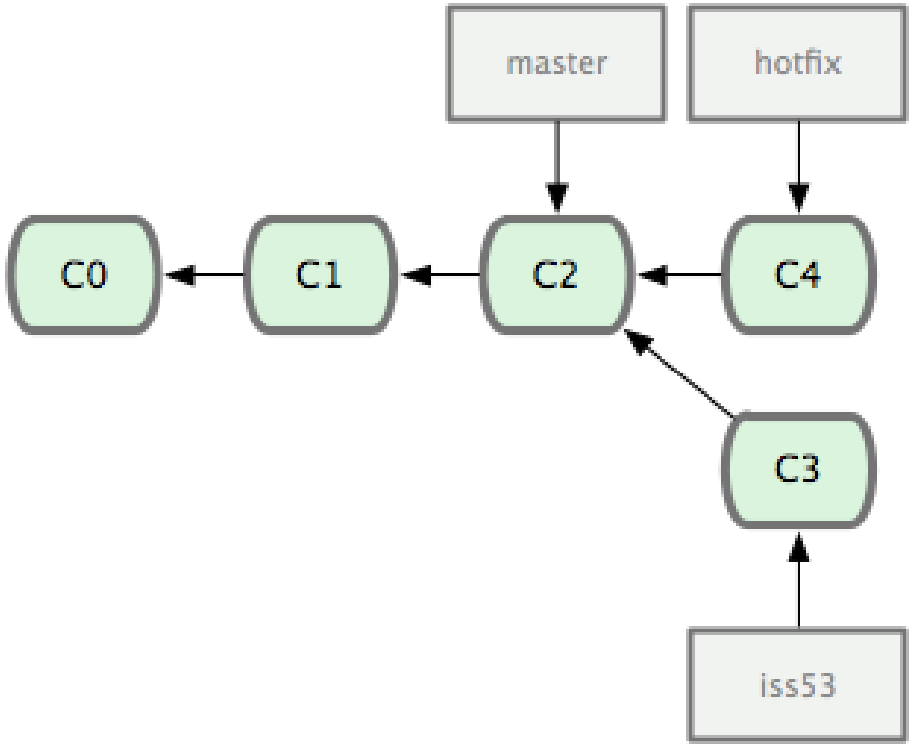
RAMIFICACIONES: EJEMPLO PRÁCTICO

- Llama el cliente. Hay que resolver un fallo urgente!!

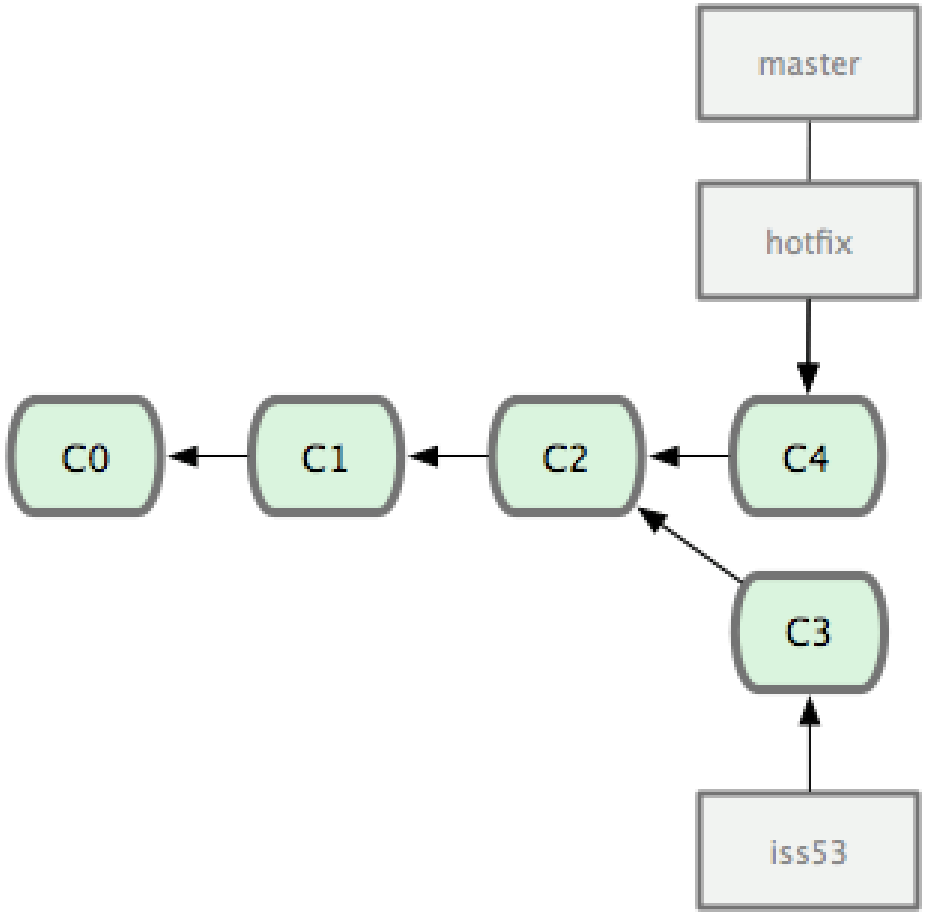


RAMIFICACIONES: EJEMPLO PRÁCTICO

- Llama el cliente. Hay que resolver un fallo urgente!!



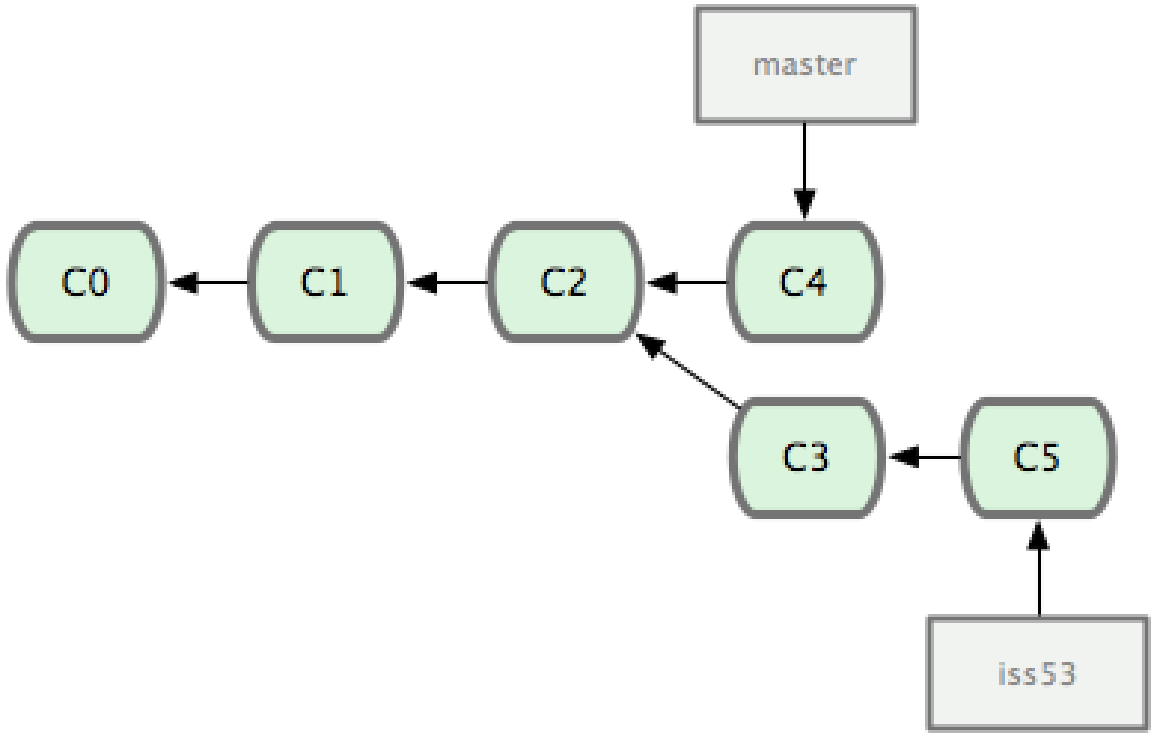
RAMIFICACIONES: EJEMPLO PRÁCTICO



- El parche funciona perfectamente, entonces se integra a master

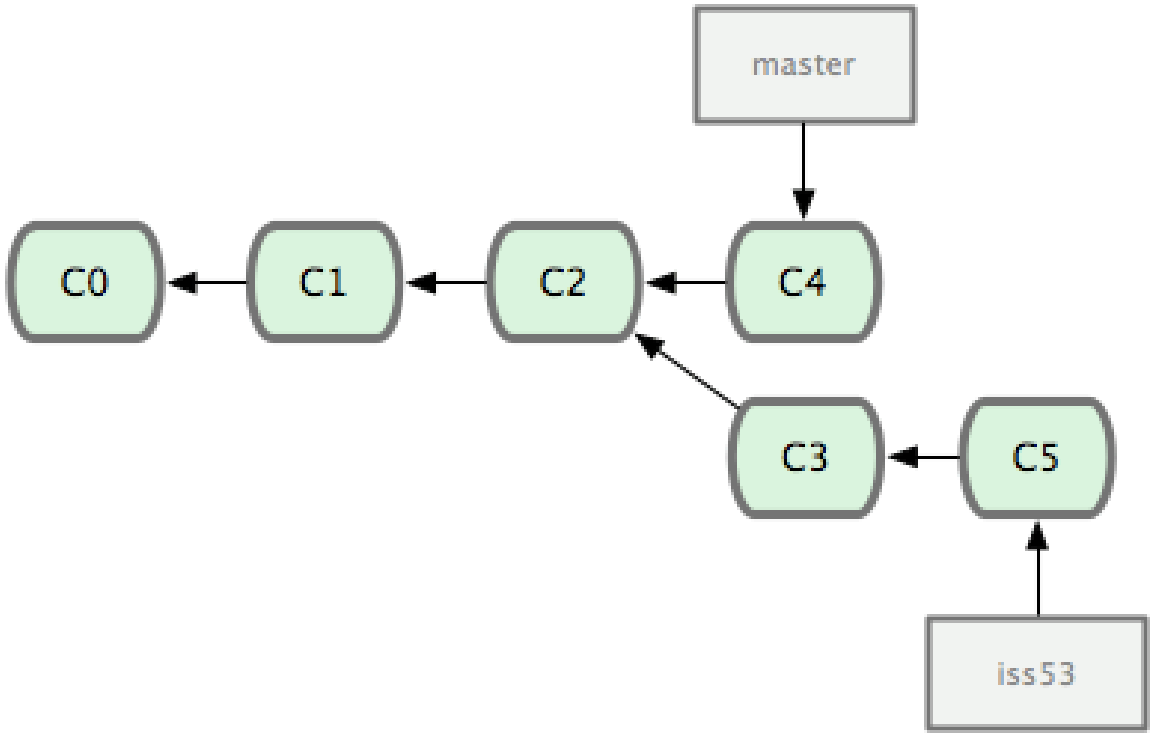
RAMIFICACIONES: EJEMPLO PRÁCTICO

- Seguimos trabajando en la funcionalidad anterior

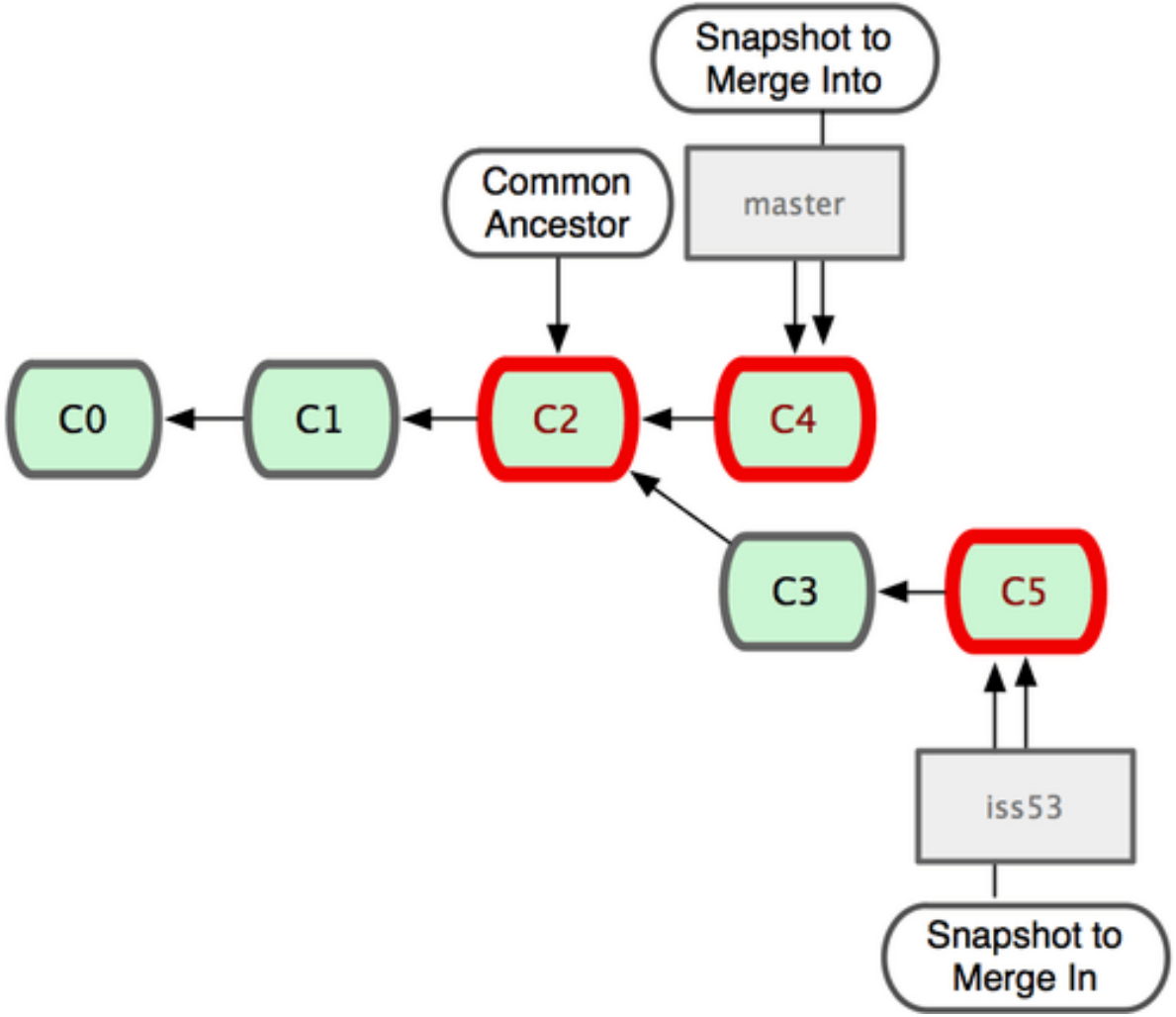


RAMIFICACIONES: EJEMPLO PRÁCTICO

- La nueva funcionalidad está completa, pero tiene el bug de C2

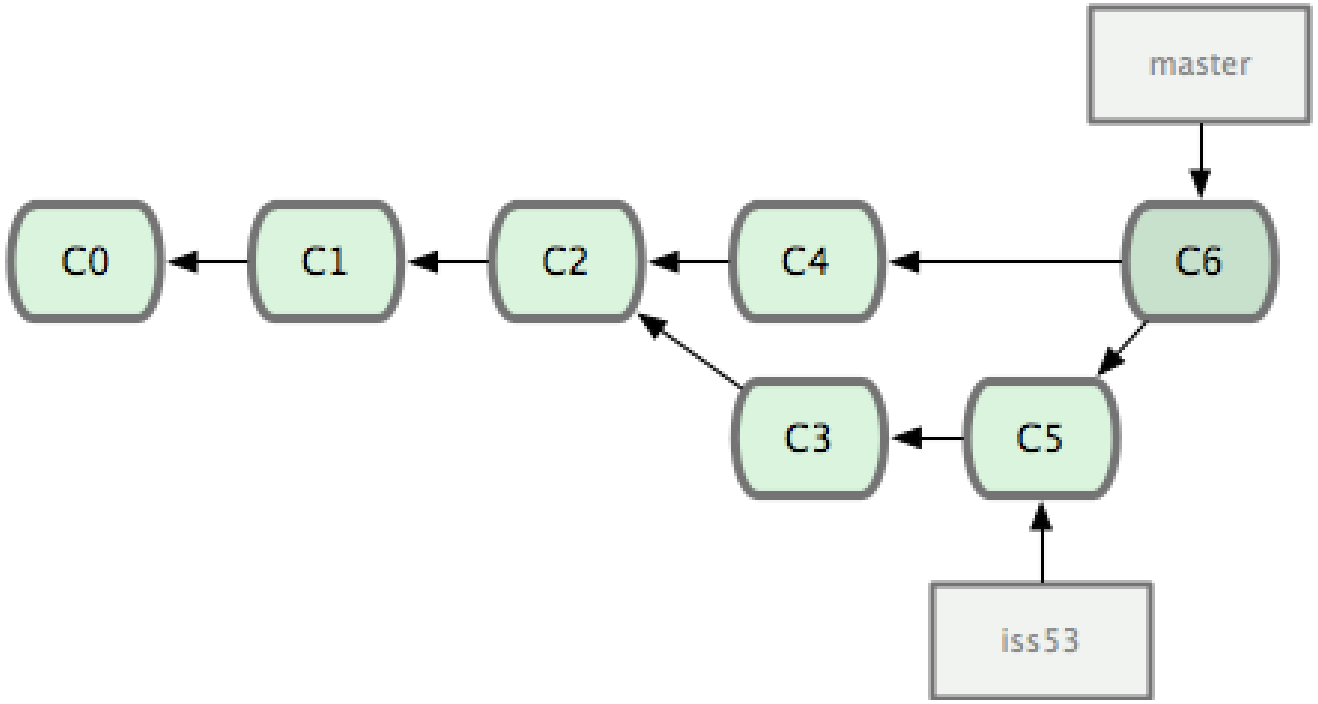


RAMIFICACIONES: EJEMPLO PRÁCTICO



RAMIFICACIONES: EJEMPLO PRÁCTICO

- `git chekout master`
- `Git merge iss53`



LINKS DE INTERÉS

- Documentación oficial de Git
  - https://git-scm.com/book/es/v1/Empezando
- Los cheatsheets son sumamente prácticos
  - https://www.git-tower.com/blog/git-cheat-sheet/
- Buenas prácticas de Git
  - https://www.git-tower.com/learn/git/ebook/en/command-line/appendix/best-practices
- Si les interesan las metodologías Agiles hay muchísimo material online
- Vale la pena leer la historia de Linus Torvalds