Sistema de Control de versiones (CVS)

MÓDULO.- DESPLIEGUE DE APLICACIONES WEB

CFGS DESARROLLO DE APLICACIONES WEB

Contenidos

- Instalación, configuración y uso de sistemas de control de versiones.
- Operaciones avanzadas.
- Seguridad de los sistemas de control de versiones.
- Historia de un repositorio

VERSIÓN

Una versión, desde el punto de vista de la evolución, se define como la forma particular de un objeto en un instante o contexto dado.

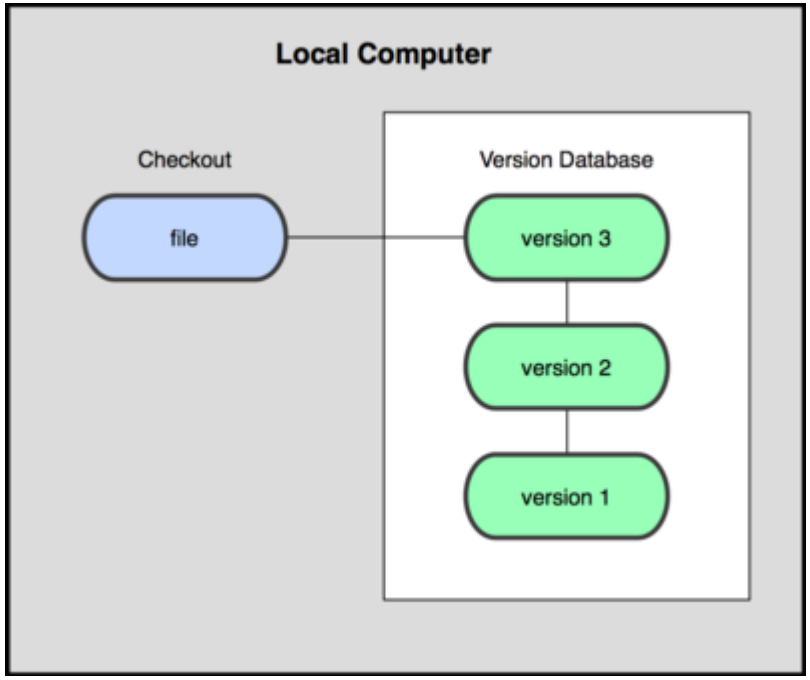
SISTEMA DE CONTROL DE VERSIONES

Cuando estamos desarrollando software, el código fuente está cambiando continuamente, siendo esta particularidad vital.

Un CVS es una herramienta que nos permite:

- Trabajar en el mismo proyecto de forma simultánea, sin que pisen unos a otros.
- Volver a una versión estable previa de código fuente

Sistema de control de versiones locales



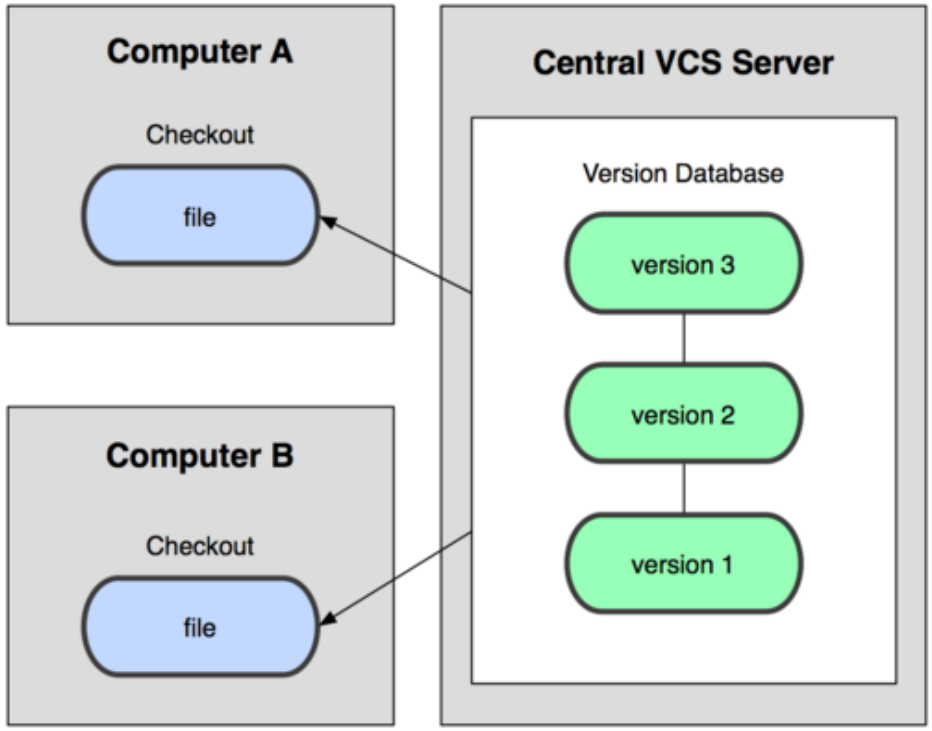
- Método copiar los archivos a otro directorio (indicando fecha y hora).
- Técnica simple
- Muchos errores (olvidar que directorio o sobreescribir archivos)
- Evitar los problemas crearon una bases de datos locales por cada cliente donde registraban todos los cambios realizados en los archivos.

Sistemas de control de versiones centralizados

- Nos permite colaborar con desarrolladores en otros sistemas.
- CVS: **Subversion y Perforce**
- Un sólo servidor central

VENTAJAS frente al sistema locales:

- Nos permite ver que hace el resto de colaboradores del equipo de desarrollo de un proyecto.
- El administrador tiene control total de que hace cada uno.

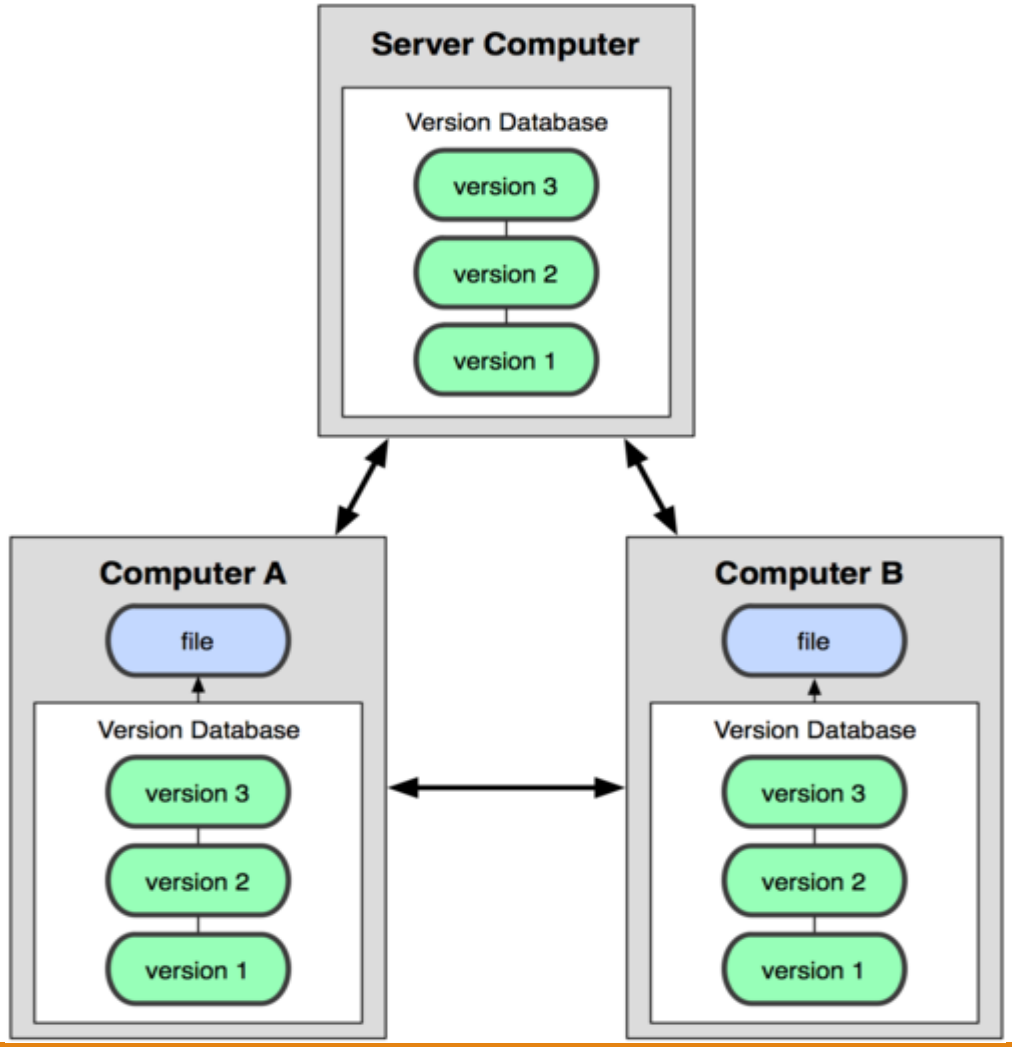


Desventajas CVS centralizado

Servidor centralizado tiene un punto único de fallo:

- Si cae el servidor, no hay colaboración con el servidor, ni se piede guardar el versionado
- Si el disco duro donde guarda los cambios se corrompe, y no hay copia de seguridad, se pierde todo.

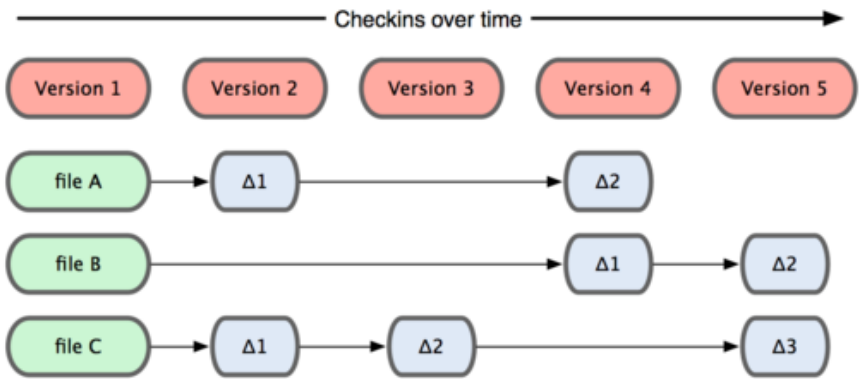
SISTEMA DE CONTROL DE VERSIONES DISTRIBUIDO (DVCS)



SISTEMA DE CONTROL DE VERSIONES DISTRIBUIDO (DVCS)

- Git, Mercurial, Bazaar o Darcs
- Los clientes no solo descargan la última instantánea de los archivos: replican completamente el repositorio.
- Si un servidor muere, los sistemas estaban colaborando a través de él, cualquiera de ellos puede restaurarlo.

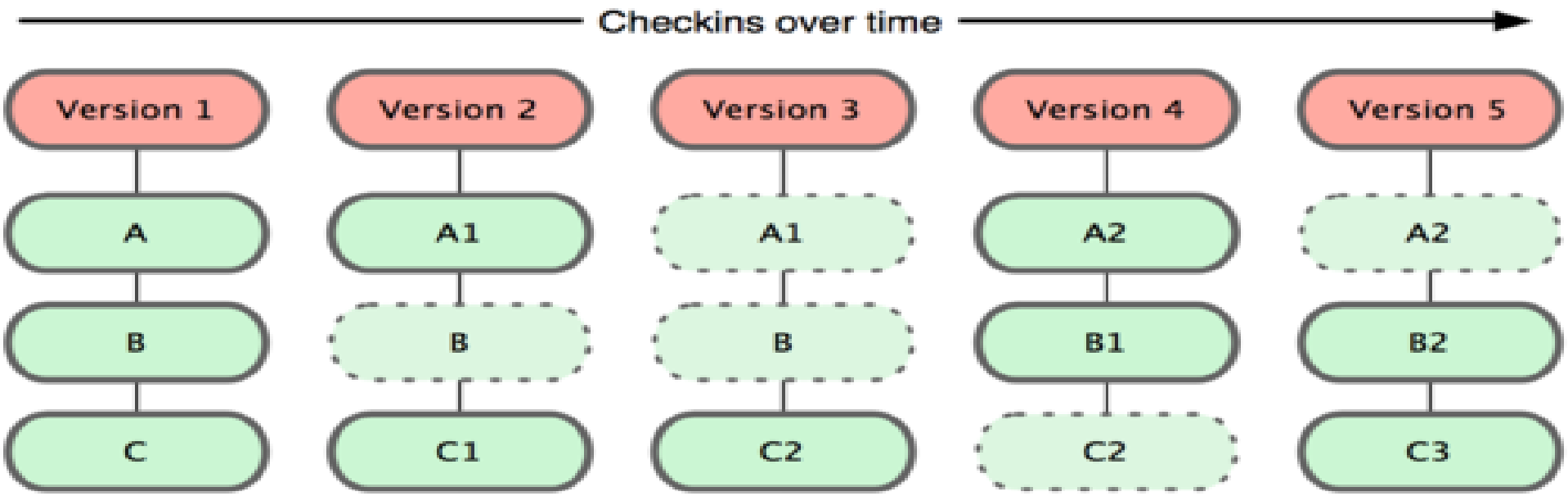
Instantáneas, no diferencias



Otros sistemas tienen a almacenar los datos como cambios de cada archivo respecto a una versión base.

CVS Centralizado el modelo de datos es un conjunto de archivos y las modificaciones hechas sobre cada uno de ellos a lo largo del tiempo

Instantáneas, no diferencias



Git modela sus datos como un conjunto de instantáneas de un mini sistema de archivos.

Si un archivo no se ha modificado, Git no almacena el nuevo archivo, sólo un enlace al archivo anterior.

¿qué es git?

- **Sistema de control de versiones distribuido.**
- **Cualquier operación en local**
- **Git tiene integridad**. Todo en Git es verificado mediante una suma de comprobación (checksum) antes de ser almacenado conocido como . Mecanismo hash SHA-1. Es imposible cambiar un archivo o /y directorio sin que Git no se de cuenta. Una función hash tiene este aspecto:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

- **Git normalmente solo añade información**

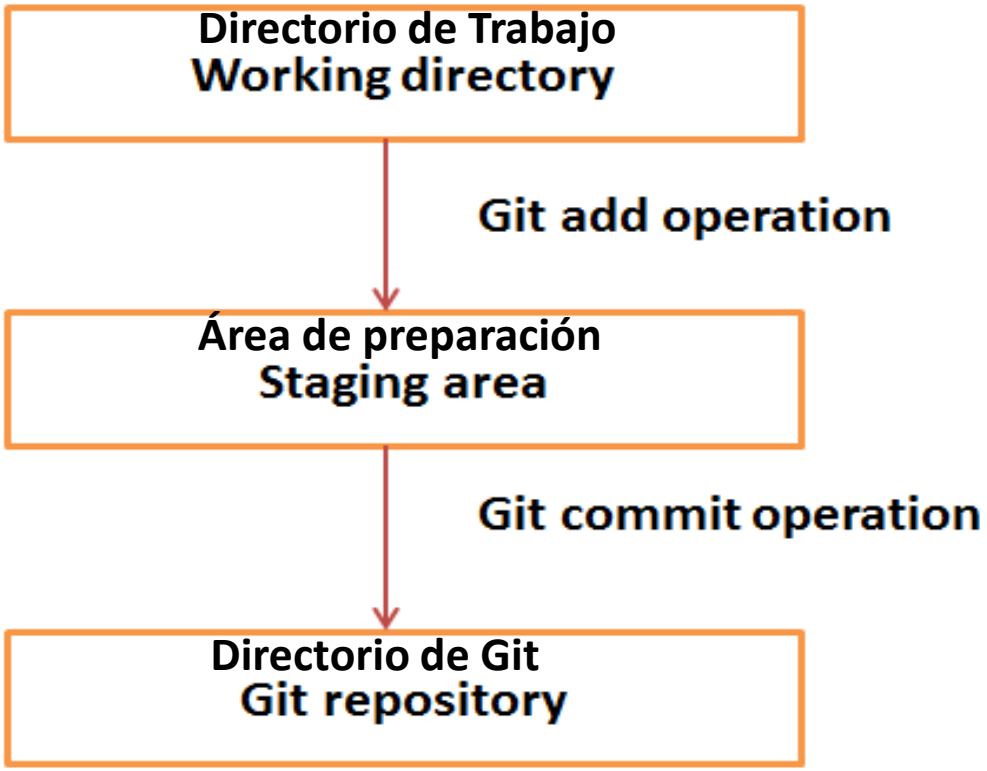
¿Qué es un repositorio?

- Un repositorio se utiliza generalmente para organizar un solo proyecto.
- Repositorios pueden contener carpetas y archivos, imágenes, vídeos, hojas de cálculo y bases de datos - todo lo que necesita su proyecto.
- Se recomienda incluir **un *README*** , o un archivo con información sobre su proyecto. *También ofrece otras opciones comunes, como un archivo de* ***licencia***.

Repositorio local:
tres secciones principales proyecto git

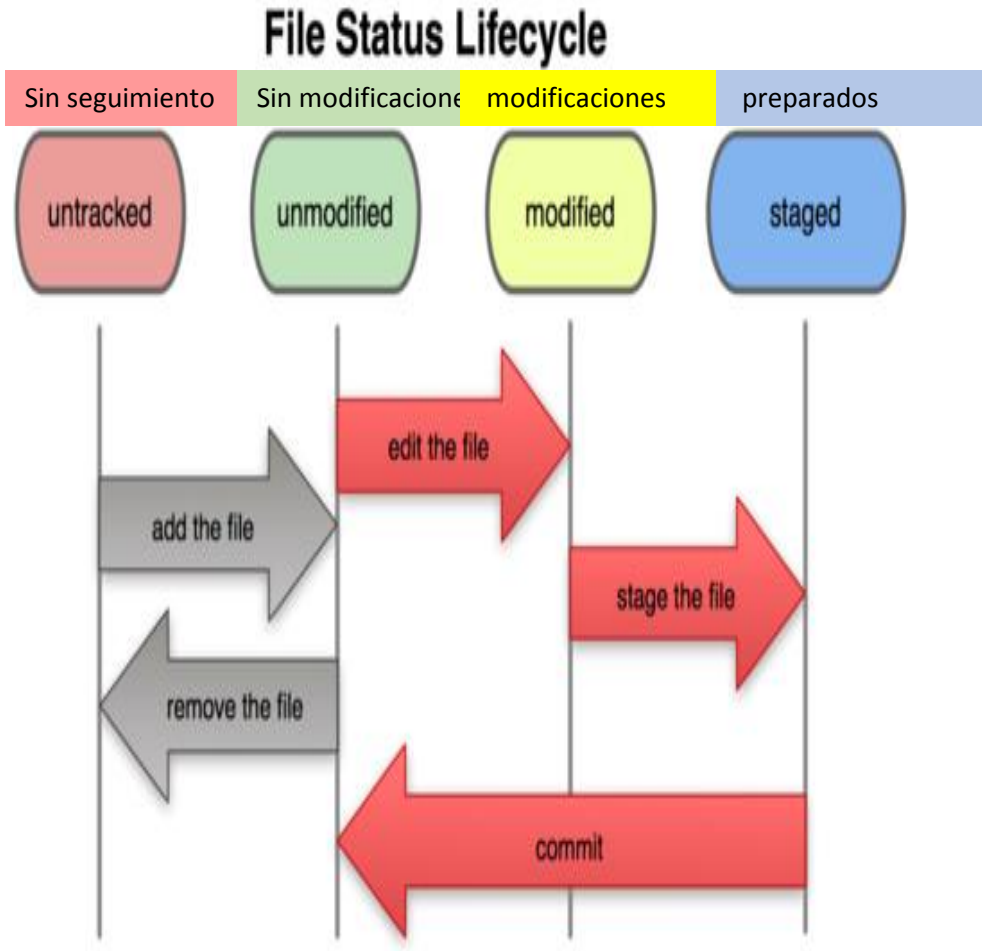
El flujo de trabajo básico en Git:

- Modificas una serie de archivos e
  tu directorio de trabajo.
- Preparas los archivos, añadiéndo
  a tu área de preparación.
- Confirmas los cambios



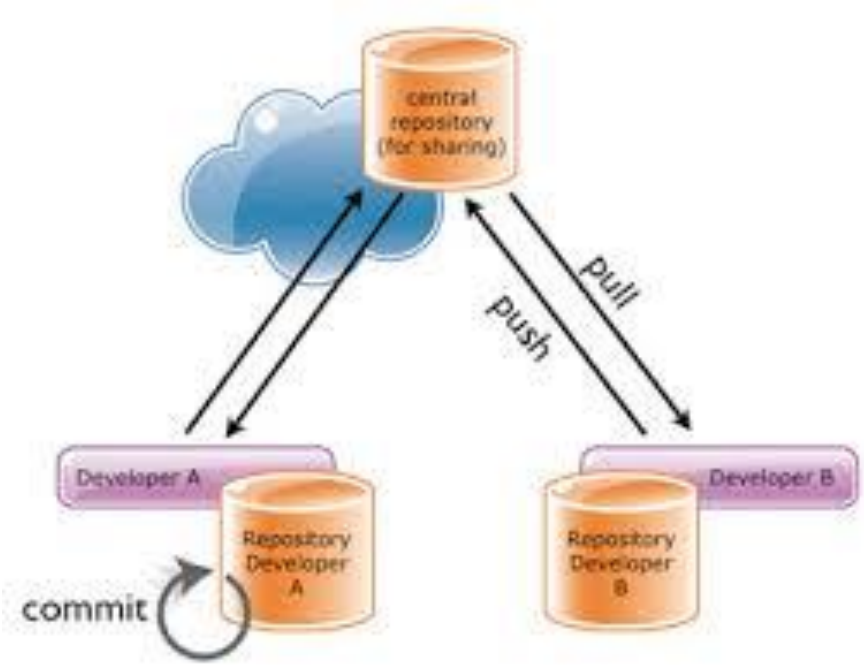
El ciclo de vida del estado de un archivo

- Git tiene tres estados en los que se pueden encontrar los archivos:
  - **Committed** – confirmado
  - **Modified** – modificado
  - **Stagged** – (preparado)

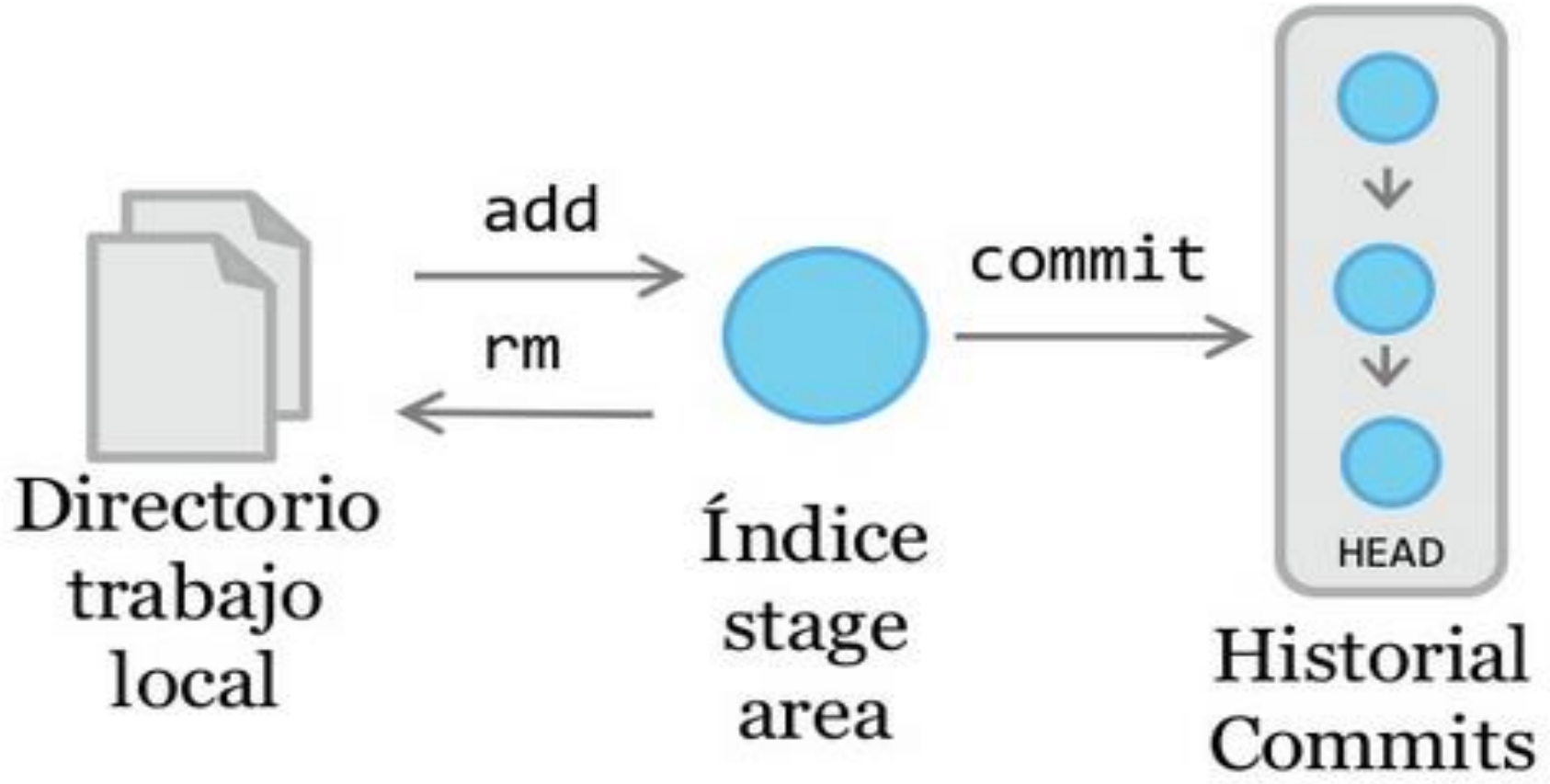


Repositorio remoto

- Son versiones de tu proyecto que se encuentran alojados en Internet o en algún punto de la red.
- Pueden existir varios, cado uno de los cuales puede ser de sólo lectura, o de lectura/escritura, según los permisos que tengas.

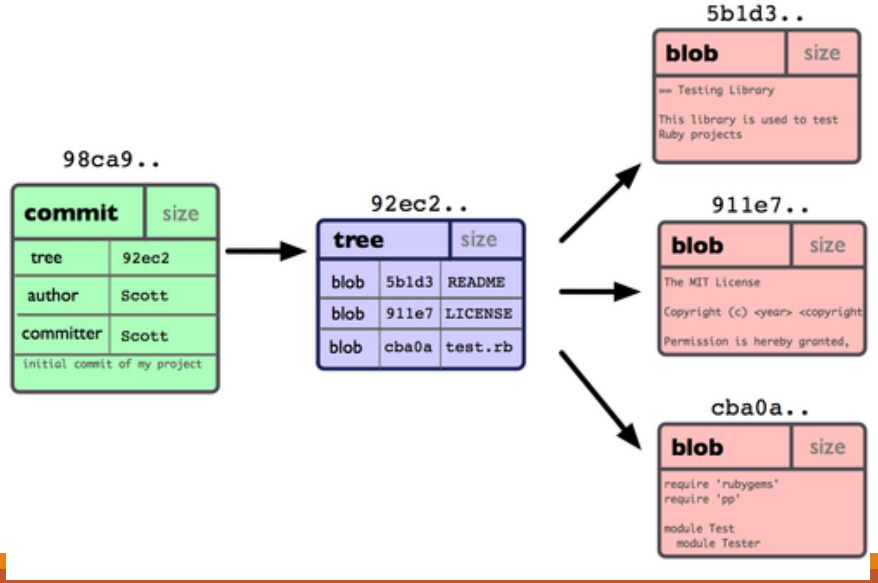


Operaciones repositorio local



Información almacena por cada punto de control

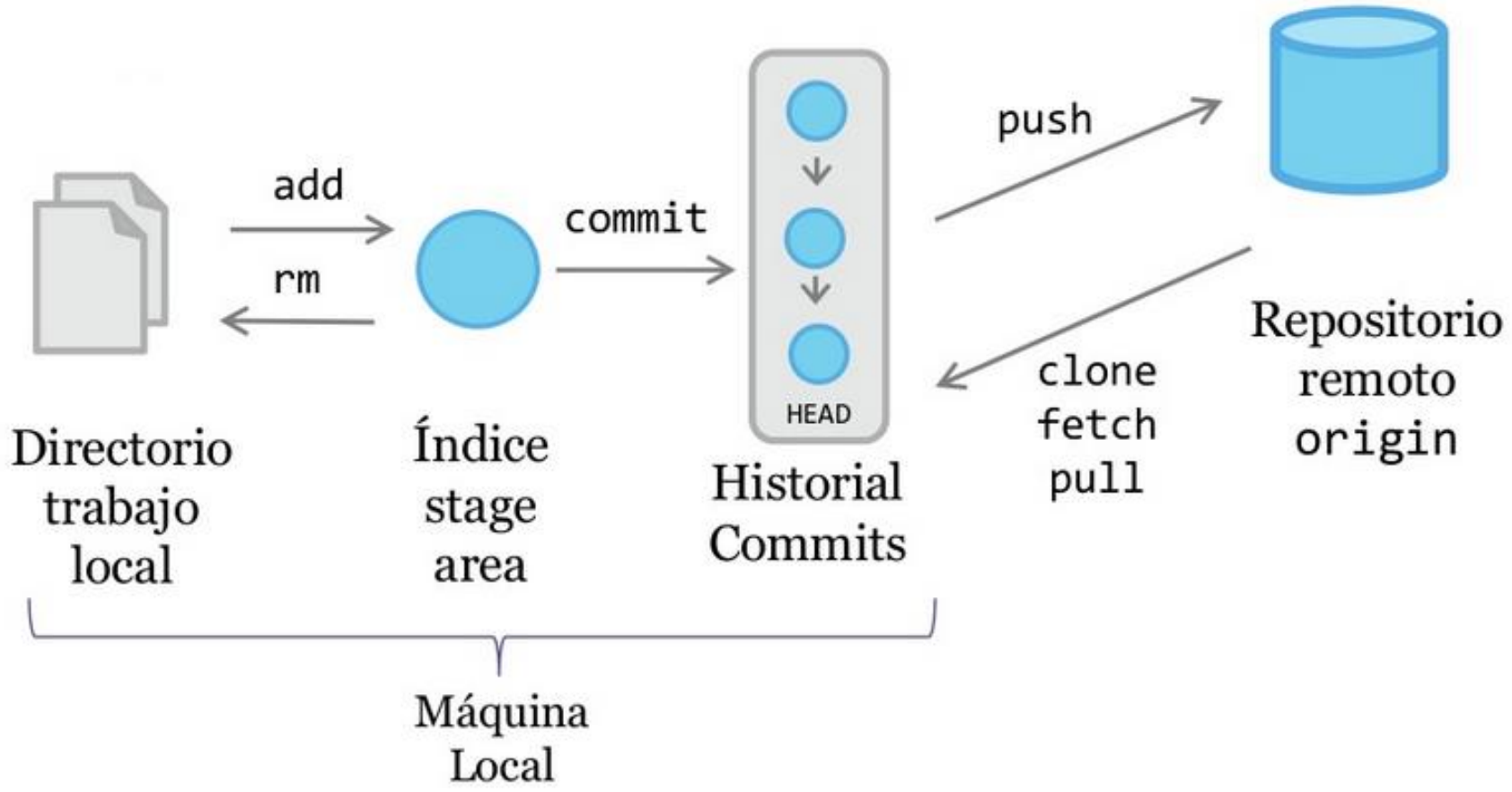
- Apuntador a la copia puntual de los contenidos preparados (staged)
- Metadatos con el autor
- Mensaje explicativo
- Uno o varios apuntadores de confirmaciones (commit) que sean padres directos de esta (un padre en caso normal) y múltiples padres (merge)




```
git add README test.rb LICENSE

git commit -m 'initial commit of my project'
```

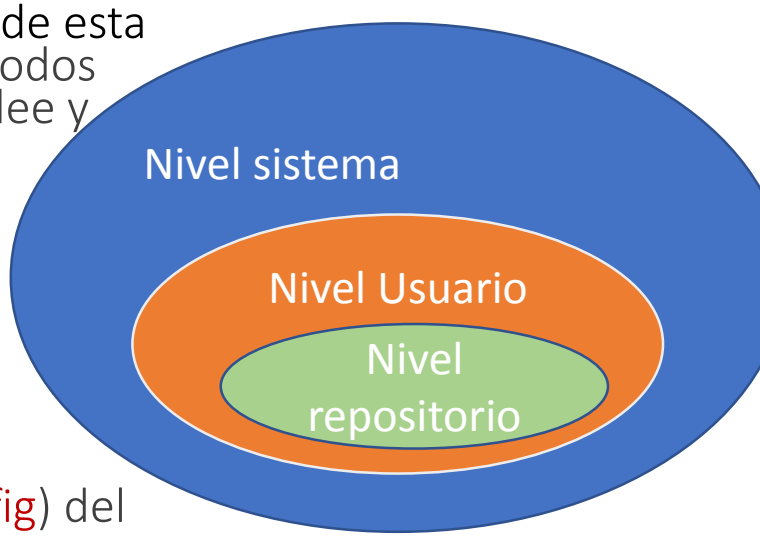
Operaciones con repositorio remoto



Configurando Git

La herramienta git config que te permite obtener y establecer variables de configuración. Estas varias se pueden almacenar en tres sitios distintos:

- Archivo /etc/gitconfig (se encuentra en el directorio donde esta instalado git): contiene todos los usuarios del sistema y todos sus repositorios. Si pasas la opción --system a git config, lee y escribe en este archivo.
- Archivo ~/.gitconfig



Usando fichero de configuración global

- ✓ git config - - global user.email “usuario@dominio”
- ✓ git config - - global user.name “usuario”

- Archivo config en el directorio de Git (es decir, .git/config) del repositorio que estés utilizando actualmente: Específico a ese repositorio.

Comprobando tu configuración

git config --list

Comandos básicos

- **git init** Inicializando el repositorio o un proyecto para controlar las versiones. Crea la carpeta .git con la estructura del repositorio
- **git add <file>** Agregar al área de trabajo. Todos archivos git add .
- **git status** para ver el estado de nuestro archivos
- **git commit** confirmación de cambios
- **git log** Ver el histórico de confirmaciones
- **git diff** Ver lo que no esta preparado (working stage).
- **git diff - - cached** (Ver lo que esta preparado para el siguiente commit)
- **git rm <file> -f** Borrada del archivo y desaparece del área de seguimiento
- git rm - - cached <file> Mantener el fichero en tu disco duro, pero interrumpir el seguimiento.
- **git mv <file_old> <file_new>** Renombrar un archivo del área de seguimiento

.gitignore

Archivos y carpetas que no hay que versionar. Como archivos log, o generados por compilador.

En dicho archivo puede contener los patrones de nombres que deseas que sean ignorados.

Por ejemplo.:

*.[oa]

*~

El primero que ignore todos los archivos que terminen por .o o .a.

El segundo que ignore todos los archivos que termine en ~.

Comandos subir repositorio local al remoto

git remote add origin http://servidor/usuario/nombreproyecto.git

Asociar al alias origin el repositorio remoto

git push -u origin master

Subir el repositorio local al repositorio remoto

Clonar un repositorio

git clone [URL] [directorio]

git clone https://servidor/usuario/nombreproyecto.git

Crea un directorio “nombredirectorio”, inicializa un directorio .git en su interior, descargar toda la información del repositorio, y saca una copia de trabajo de la última version.

Se puede asignar nombre al directorio indicándoselo por parámetro.

Trabajando con repositorios remotos

`git remote`

Mostrar los repositorios remotos

Si añadimos la opción –v, muestra la URL asociada a cada repositorio

`git remote add [nombre] [URL]`

Añadir repositorios

`git fetch nombre`

Recuperar toda la información de un repositorio – no la une automáticamente con tu trabajo ni modifica aquello en lo que estás trabajando-,

`git pull`

Recuperar y unir automáticamente la rama remota con tu rama actual

Trabajando con repositorios remotos

`git remote show [nombre]`
Inspeccionando un repositorio remoto

`git remote rename nombreviejo nuevoNombre`
Cambiar de nombre una rama

`git remote rm [nombre]`
Borrar una rama

Etiquetas

Listar etiquetas

```
git tag
```

Crear etiquetas

```
git tag –a nombreetiqueta –m ‘mensaje de la etiqueta’
```

Ver los datos de la etiqueta

```
git show nombreetiqueta
```

Viendo histórico de confirmaciones

- git log

Con el parámetro –p –número

- Indica las últimas “número” entradas.
- Nos muestra las diferencias por línea tras cada entrada

Si deseamos ver la diferencia por palabras añadimos

-- word-diff Útil para libros de texto como libros o tu propia tesis.

- Con el parámetro –U1, a nivel de cuantas líneas. Por ejemplo 1.

- **git log -- stat** Imprime una lista de archivos modificados, cuantas líneas han sido añadidas y eliminadas en cada uno de ellos.
- **git log – pretty=oneline Imprime cada confirmación en una línea.**
- **Otras opciones son short, full y fuller.**

Deshaciendo cosas

- Modificando la última confirmación

```
git commit –amend
```

- Sacar un file del area staged

```
git reset HEAD <file>
```

- Pasar fichero de modified a unmodified (los cambios del fichero desaparecen del working área)

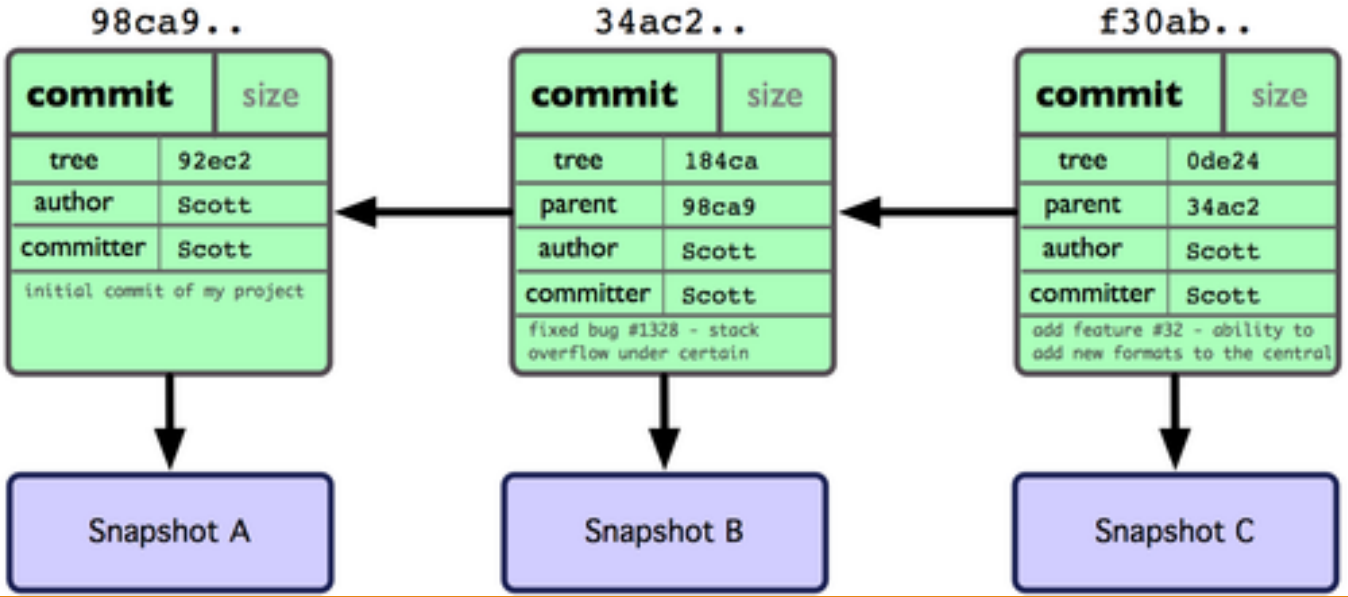
```
git checkout - -  <file>
```

RAMA GIT

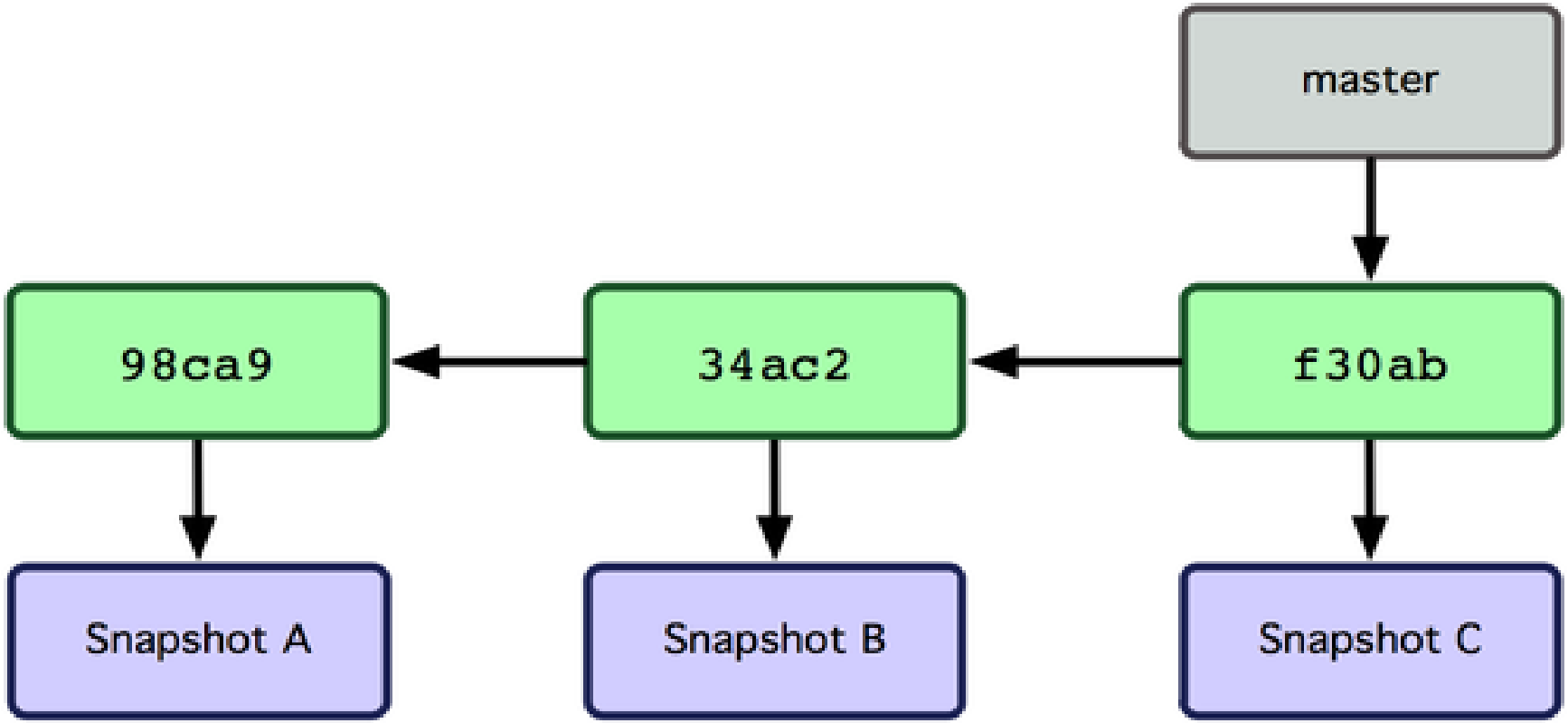
- Una rama Git es realmente un archivo que contiene 40 caracteres de una suma de control SHA-1, (representando la confirmación de los cambios a los que apunta), no cuesta nada crear y destruir ramas en Git.
- Crear una nueva rama es tan rápido y simple como escribir 41 caracteres en un archivo (40 caracteres y un retorno de carro).

Rama git

- Una rama Git es simplemente un apuntador móvil apuntando a una de esas confirmaciones.
- Es una línea de tiempo de nuestro proyecto, que nos sirve para arreglar errores, experimentar, hacer grandes cambios,etc.
- Datos tras varios commit



Rama “master”



El apuntador “master” apuntará siempre a la última confirmación.
La **rama “master”** es en donde comenzamos a trabajar, es la rama principal y estable de

TRABAJANDO CON RAMAS (BRANCHING)

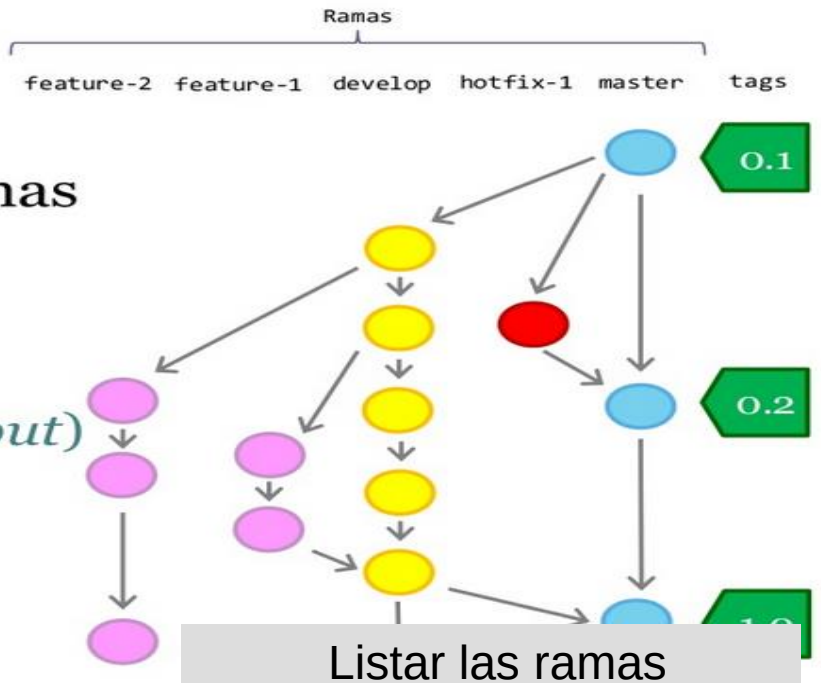
Significa que partimos de la rama principal de desarrollo (**master**) y apartir de ahí se sigue trabajando sin la rama principal (master).

Ramas

Git facilita gestión de ramas
- master = rama inicial

Operaciones:
- Crear ramas (*branch*)
- Cambiar a ramas (*checkout*)
- Combinar (*merge*)
- Etiquetar (*tag*)

Múltiples estilos de ramificación



Listar las ramas
g**it branch**

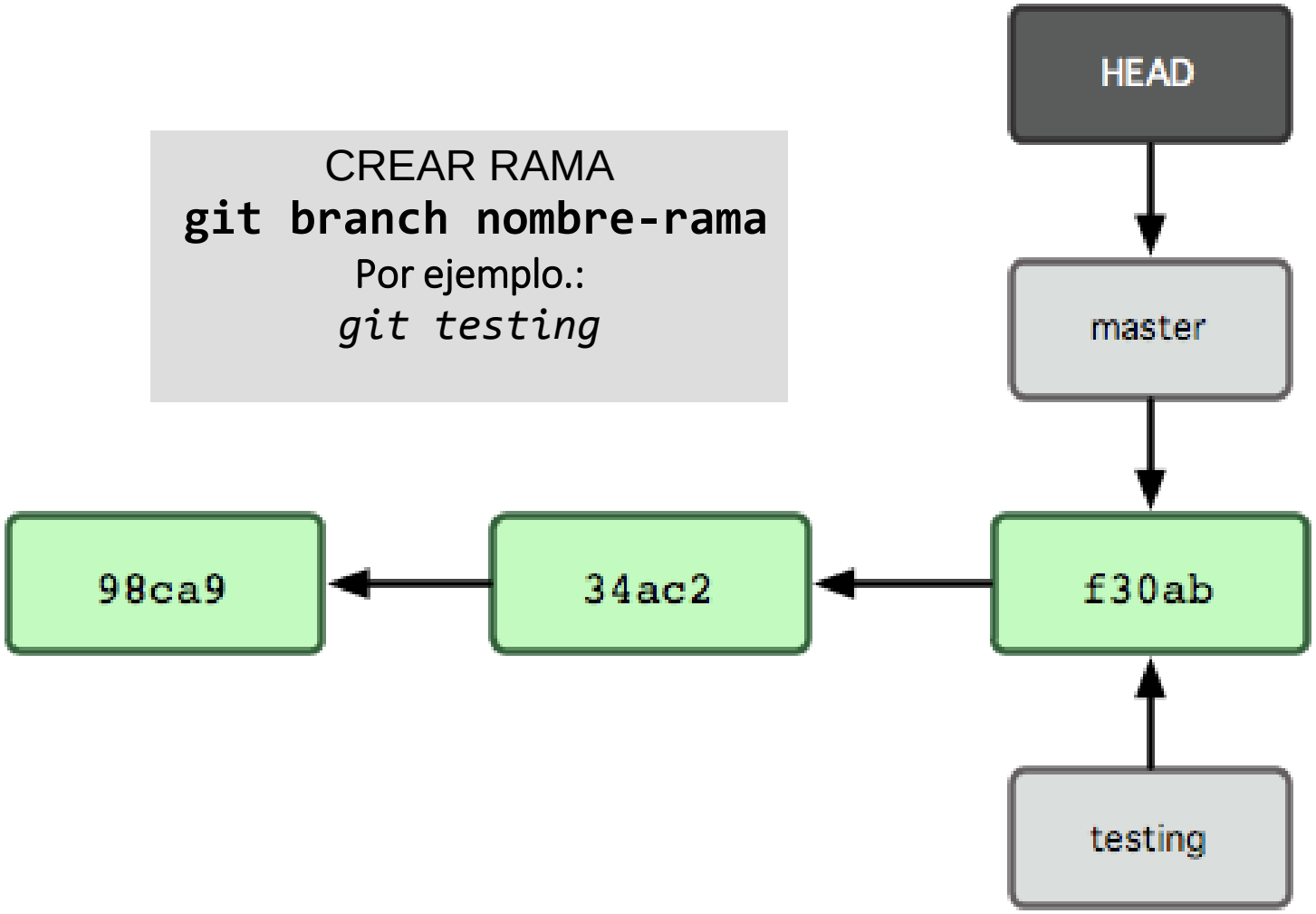
Creación de una rama

CREAR RAMA

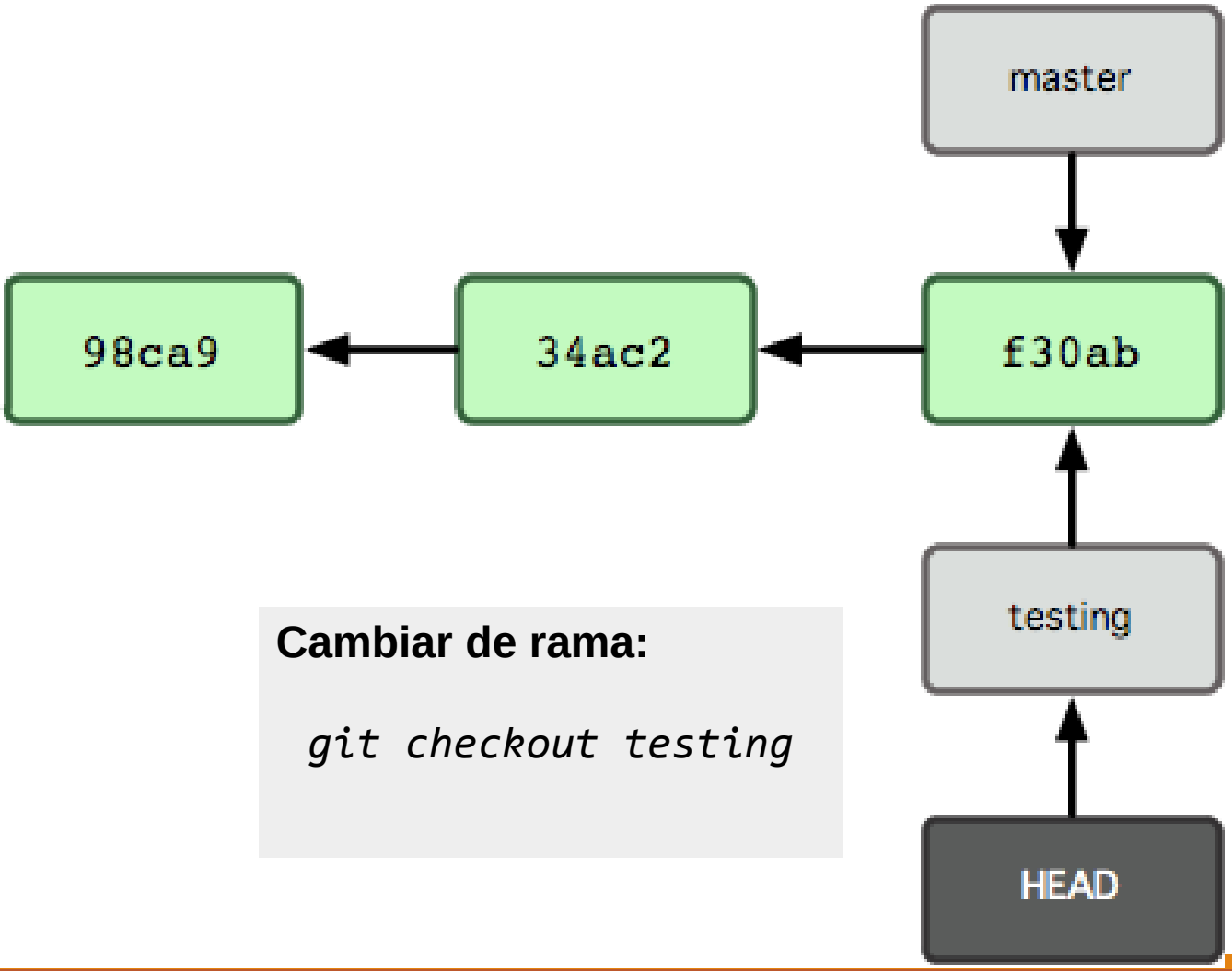
`git branch nombre-rama`

Por ejemplo.:

`git testing`



Saltar de rama

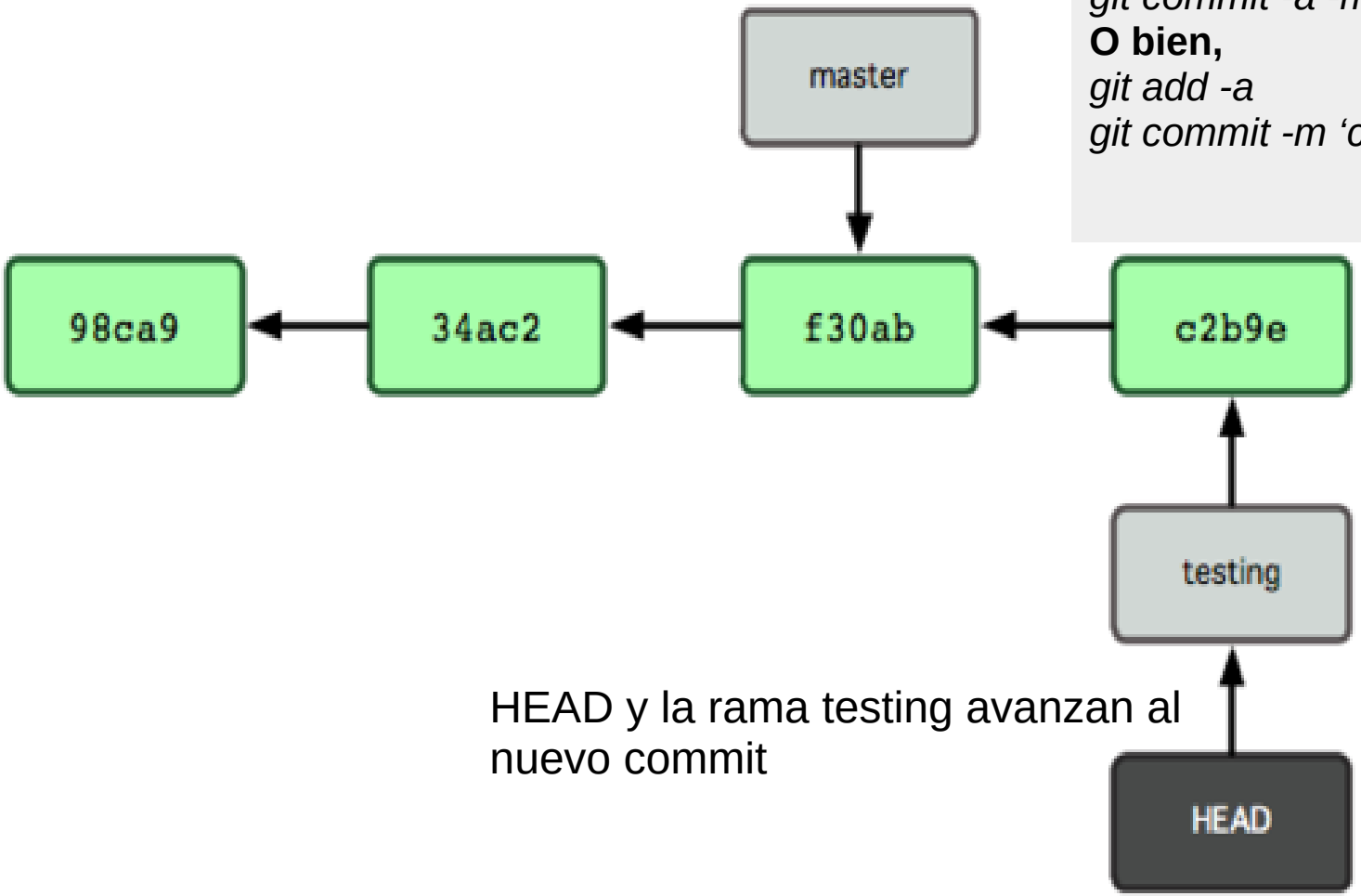


Cambiar de rama:

```
git checkout testing
```

Nuevo commit en la rama “TESTING”

Modificar cambios en los ficheros(ej. index.htm
Realizar un commit:
git commit -a -m 'cambio el estilo de header'
O bien,
git add -a
git commit -m ‘cambio el estilo de header'

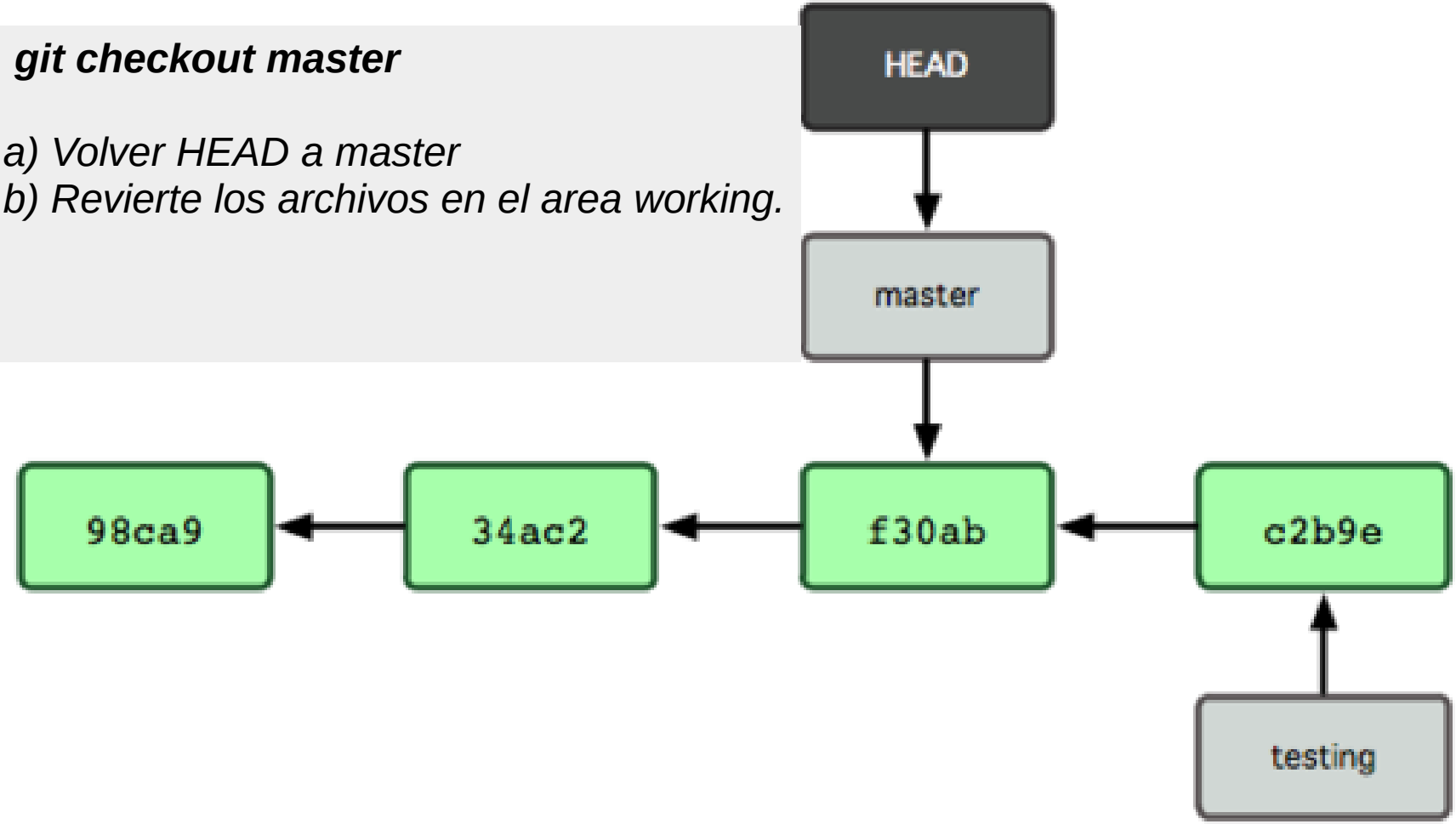


HEAD y la rama testing avanzan al nuevo commit

Volver a la rama “master”

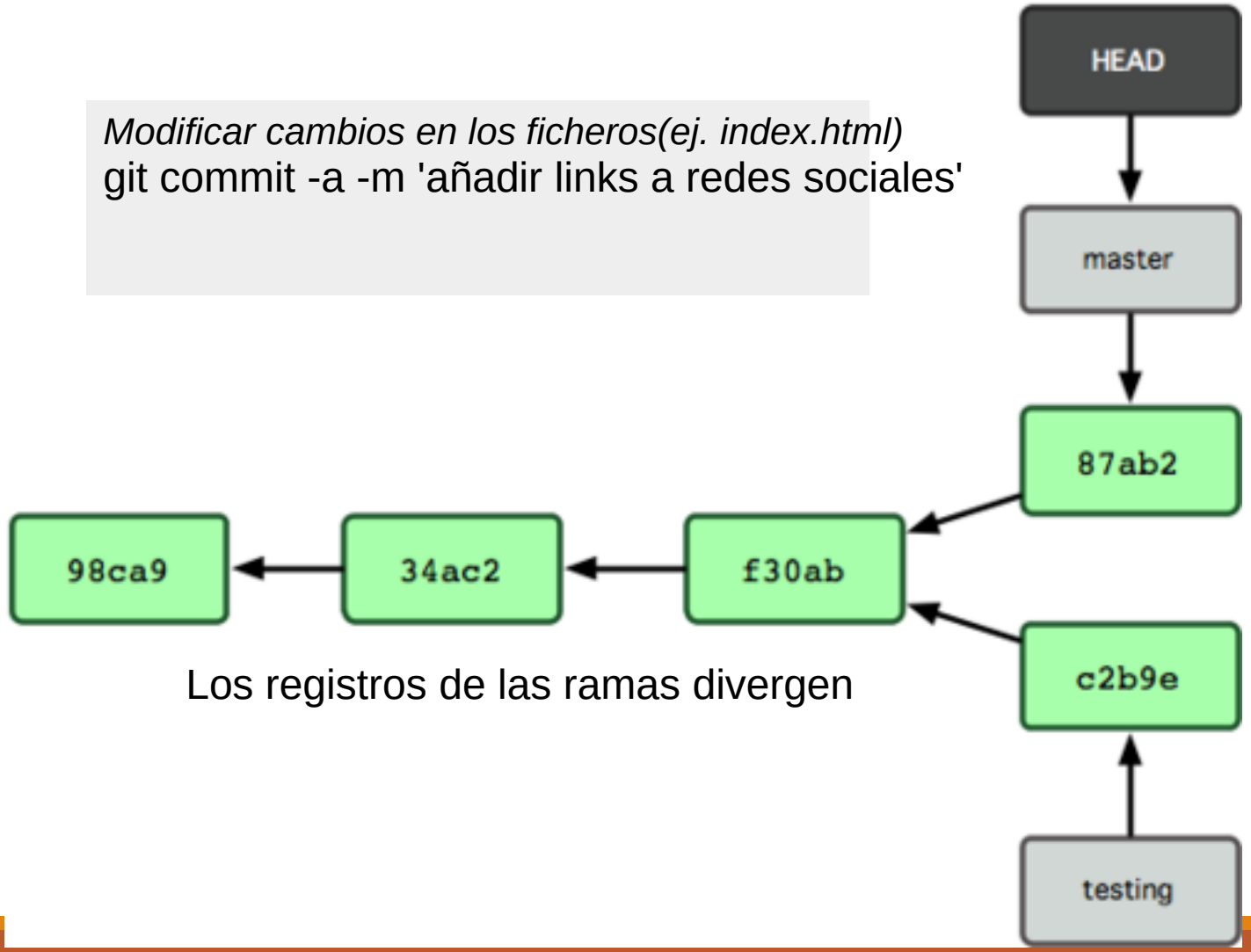
git checkout master

a) Volver HEAD a master
b) Revierte los archivos en el area working.



Commit en la rama master

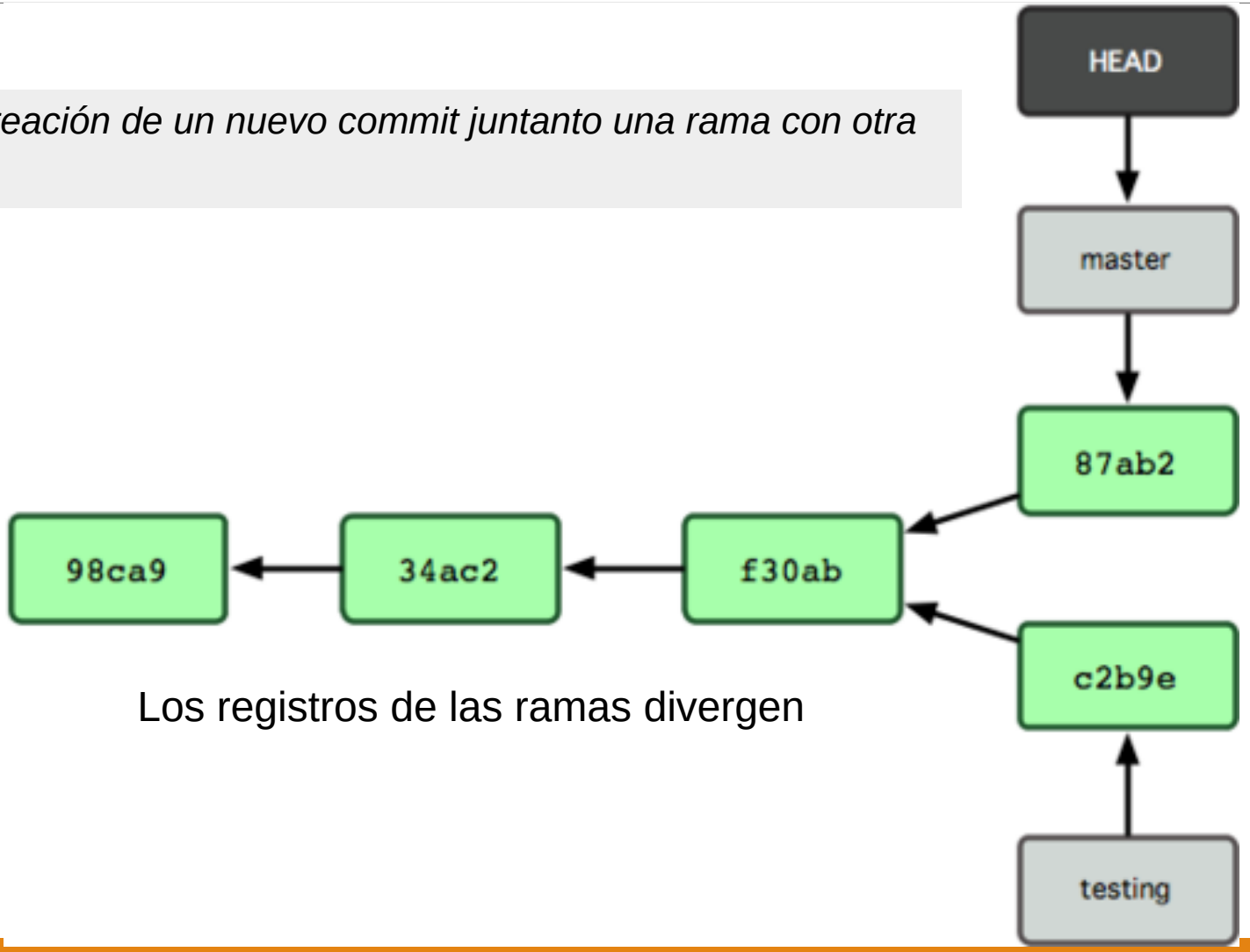
Modificar cambios en los ficheros(ej. index.html)
git commit -a -m 'añadir links a redes sociales'



Los registros de las ramas divergen

Fusiones

Es la creación de un nuevo commit juntanto una rama con otra



Los registros de las ramas divergen

Fusiones

Situarnos en la rama destino, donde se va a absorber

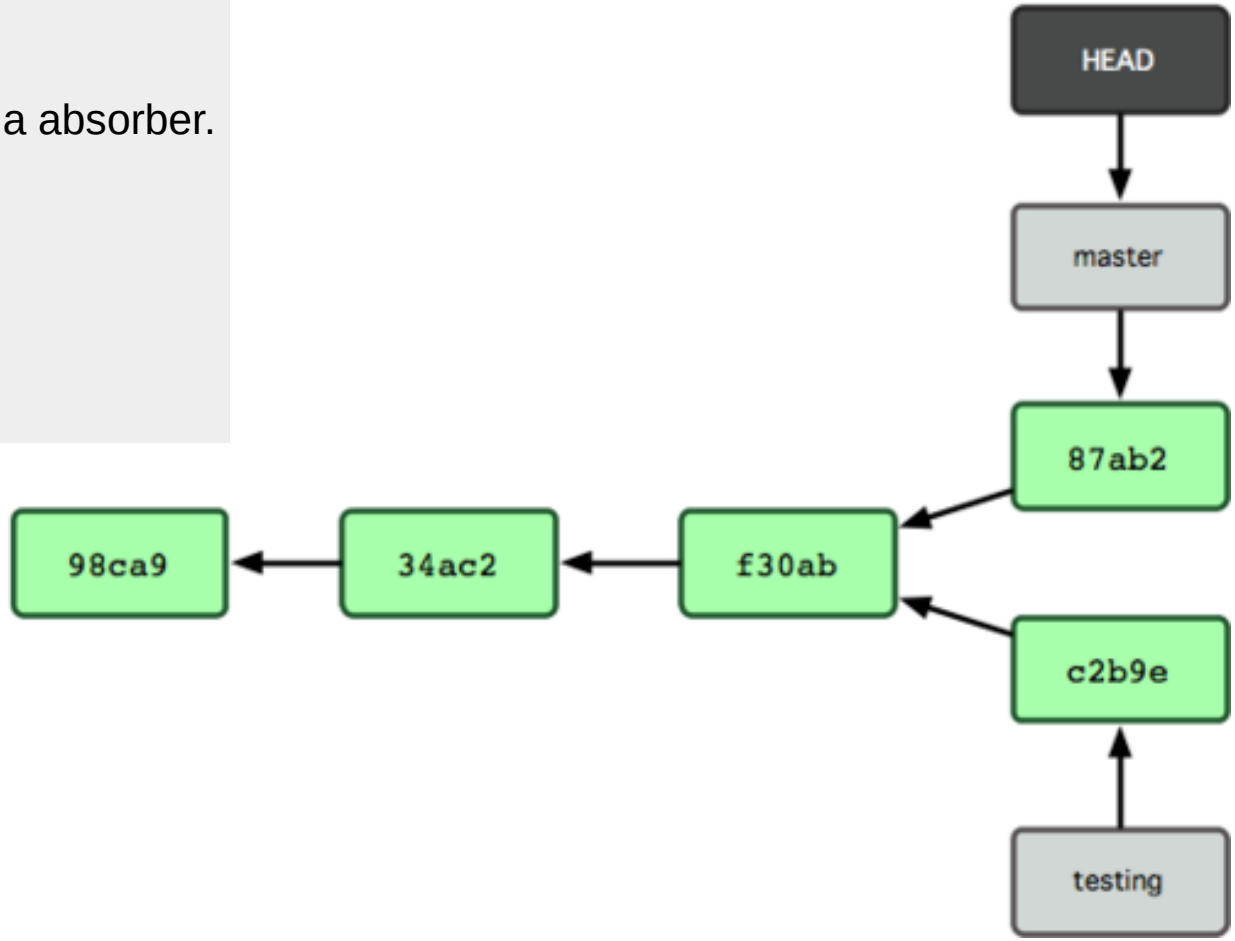
git checkout master

Fusionar rama, testing, que se va a absorber.

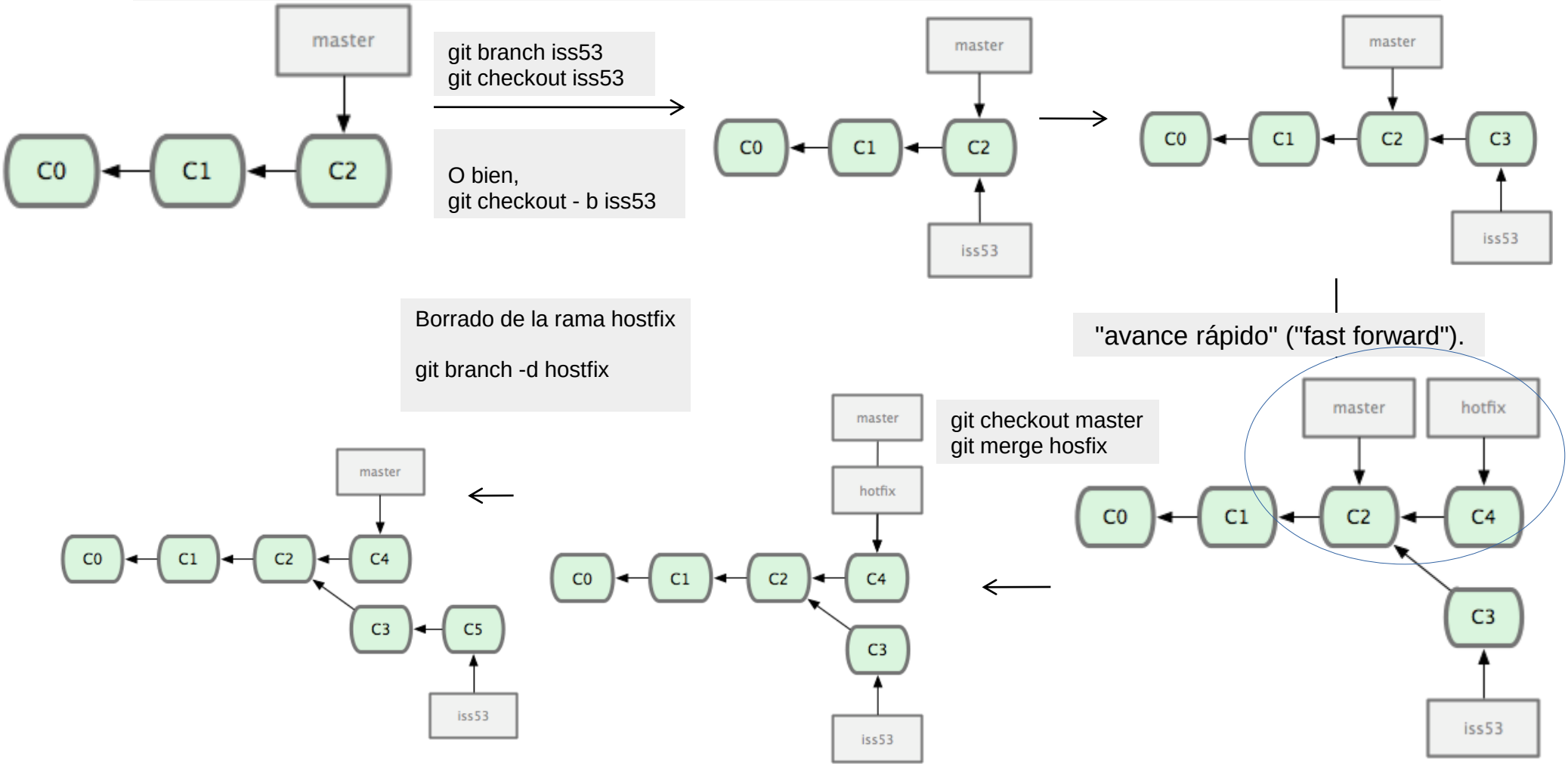
git merge testing

Borrado de rama testing

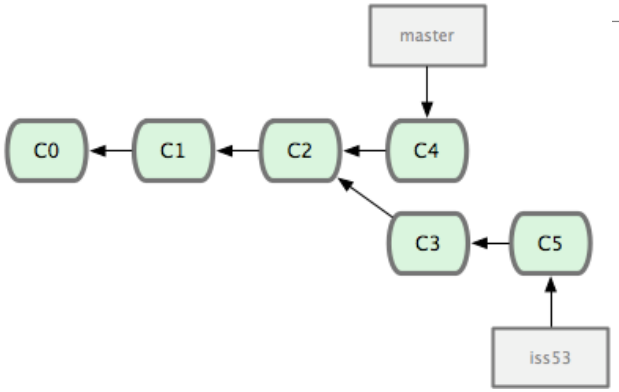
git branch -d testing



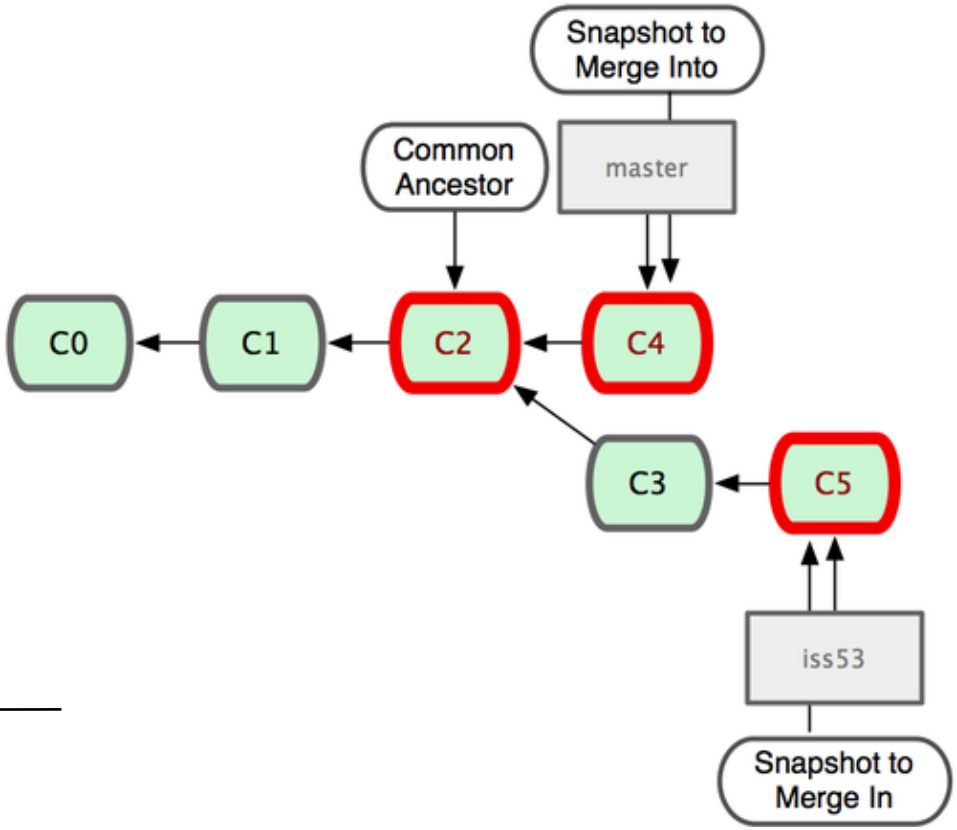
Fast forward (AVANCE RÁPIDO)

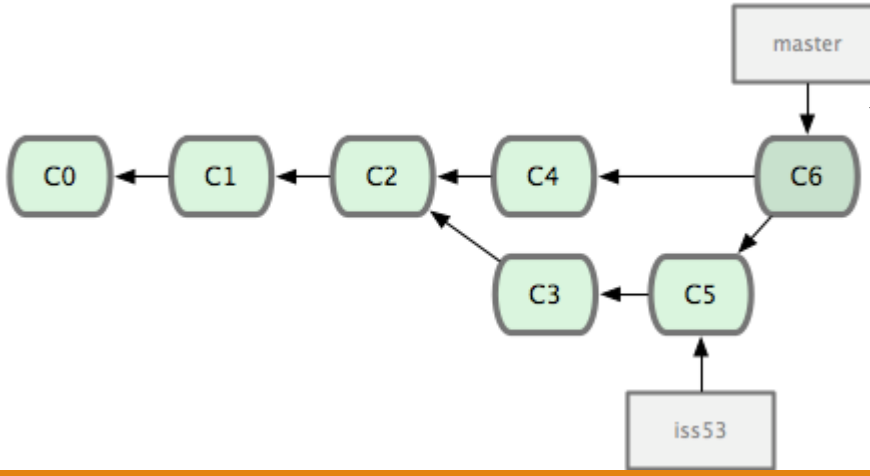


Procesos básicos de fusión (MERGE)



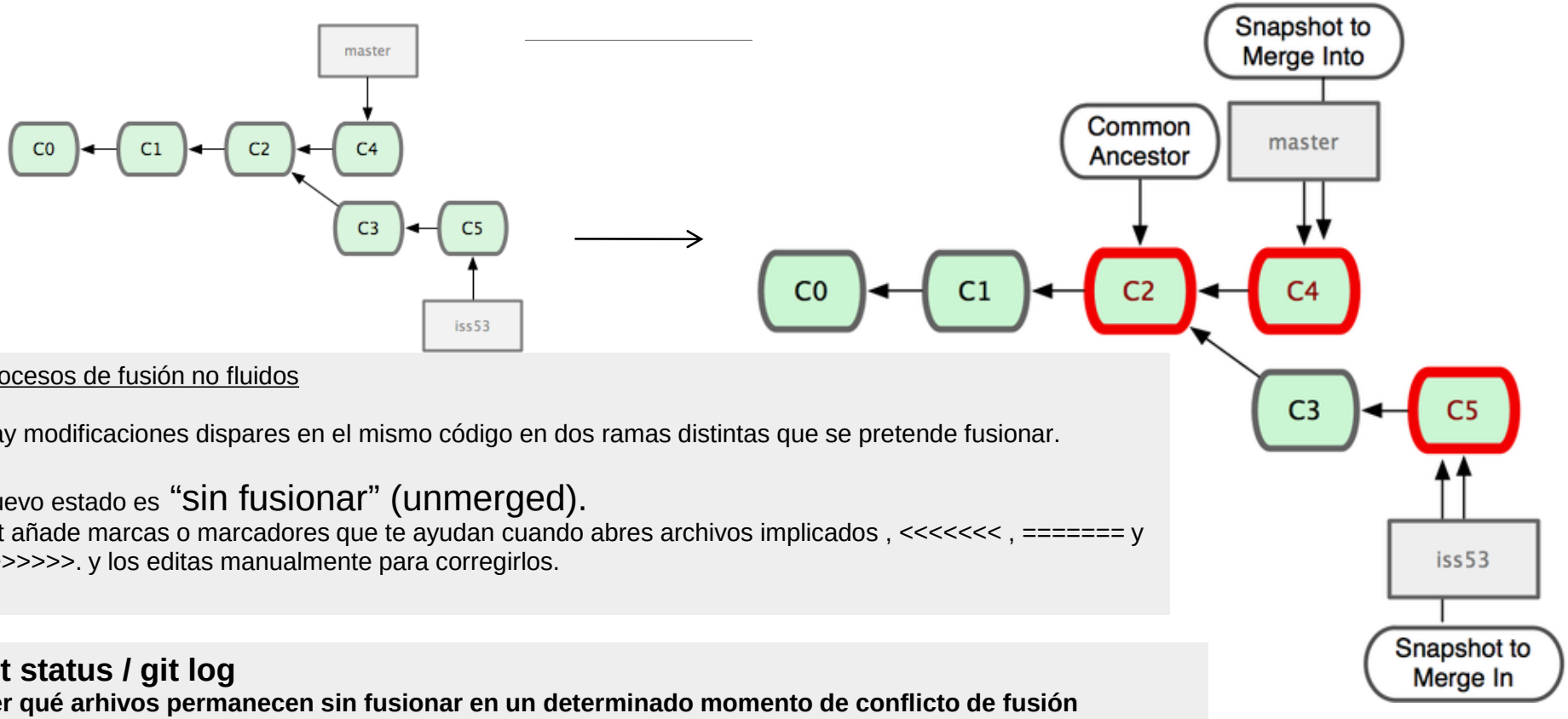
Llevar cambios de rama master iss53. Si hay que llevarlos con:
git checkout master
git merge iss53





PROCESO DE FUSIÓN CONFIRMADA *(merge commit)*
Git crea una nueva instantánea, C6, resultante de tres bandas: crea una nueva confirmación (commit) que apunta a ella.
Su principal características es que tiene más de un padre.
Siguiente paso, borrar la rama iss53.

Conflictos en las fusiones



Procesos de fusión no fluidos

Hay modificaciones dispares en el mismo código en dos ramas distintas que se pretende fusionar.

Nuevo estado es “sin fusionar” (unmerged).
Git añade marcas o marcadores que te ayudan cuando abres archivos implicados , <<<<<<< , ======= y >>>>>>>. y los editas manualmente para corregirlos.

git status / git log
Ver qué arhivos permanecen sin fusionar en un determinado momento de conflicto de fusión

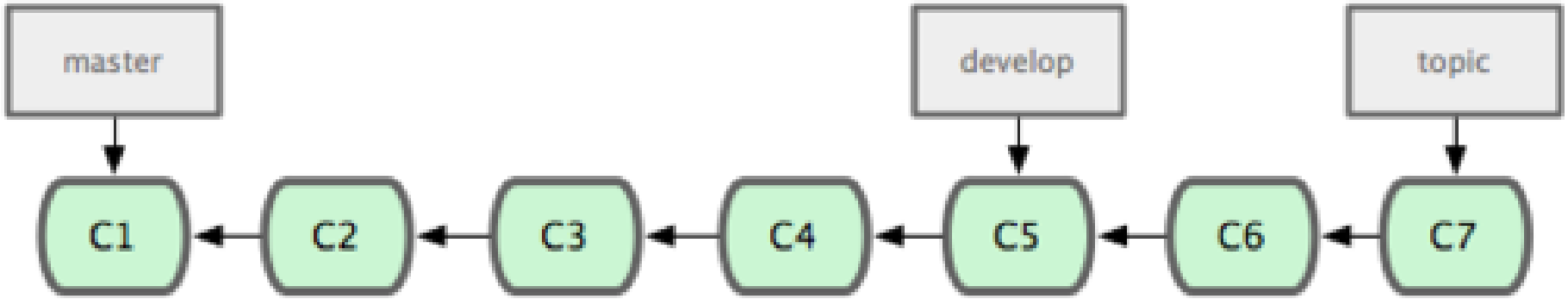
Flujo de trabajo por ramas

RAMAS DE LARGO RECORRIDO:

A) **master** - código estable

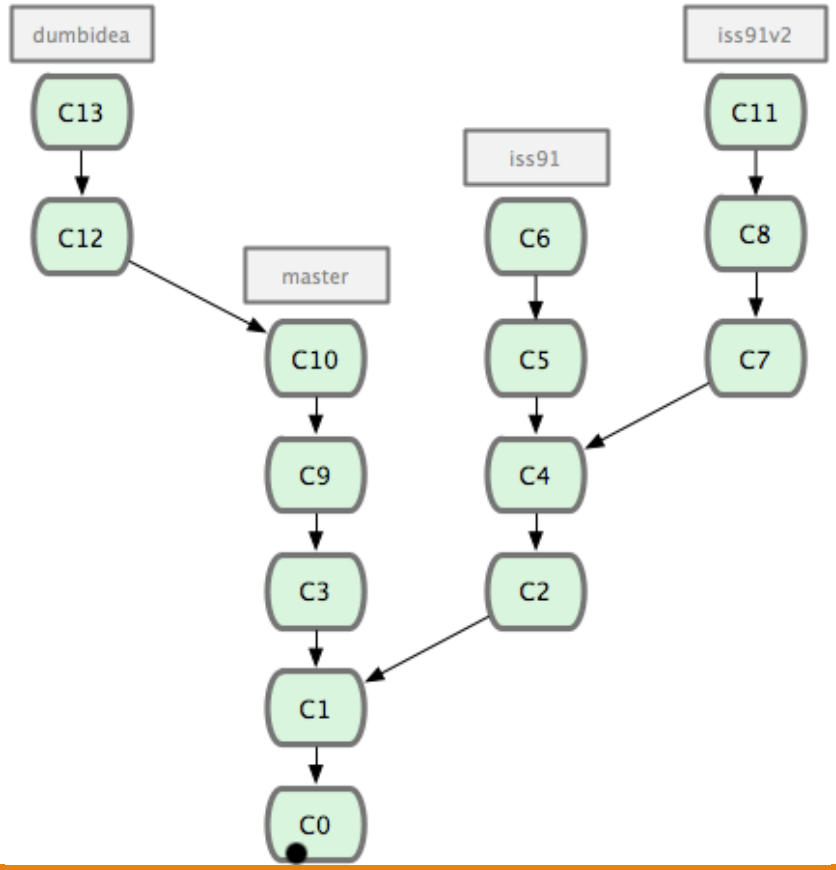
B) **develop** - ramas en las que se trabaja; cada vez que son estables se fusionan con la ma

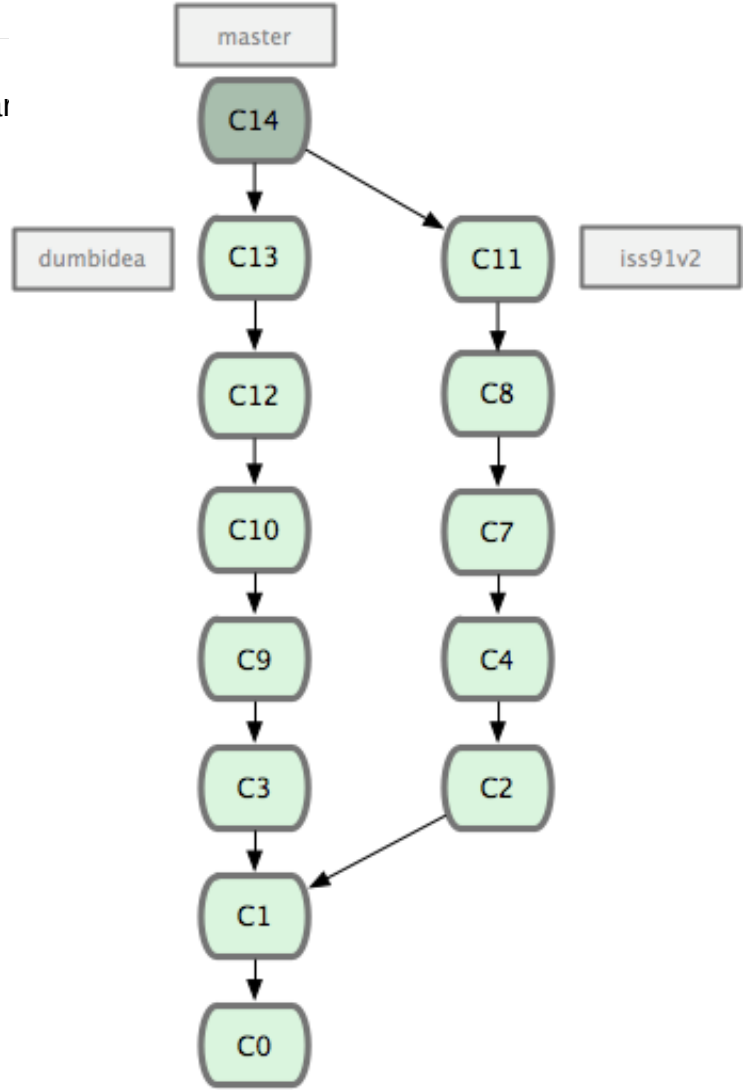
Este sistema de trabajo se puede ampliar para diversos grados de estabilidad. Algunos pro



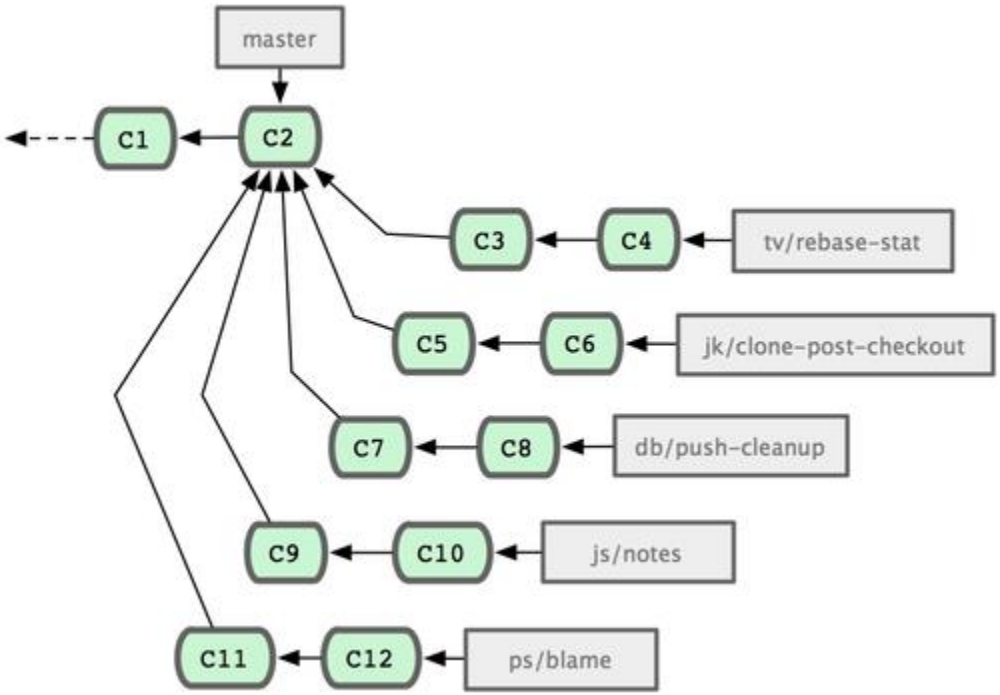
Flujo de trabajo por ramas

RAMAS PUNTUALES:
Una rama puntual es aquella de corta duración que abres para un tema o par
una funcionalidad muy concreto





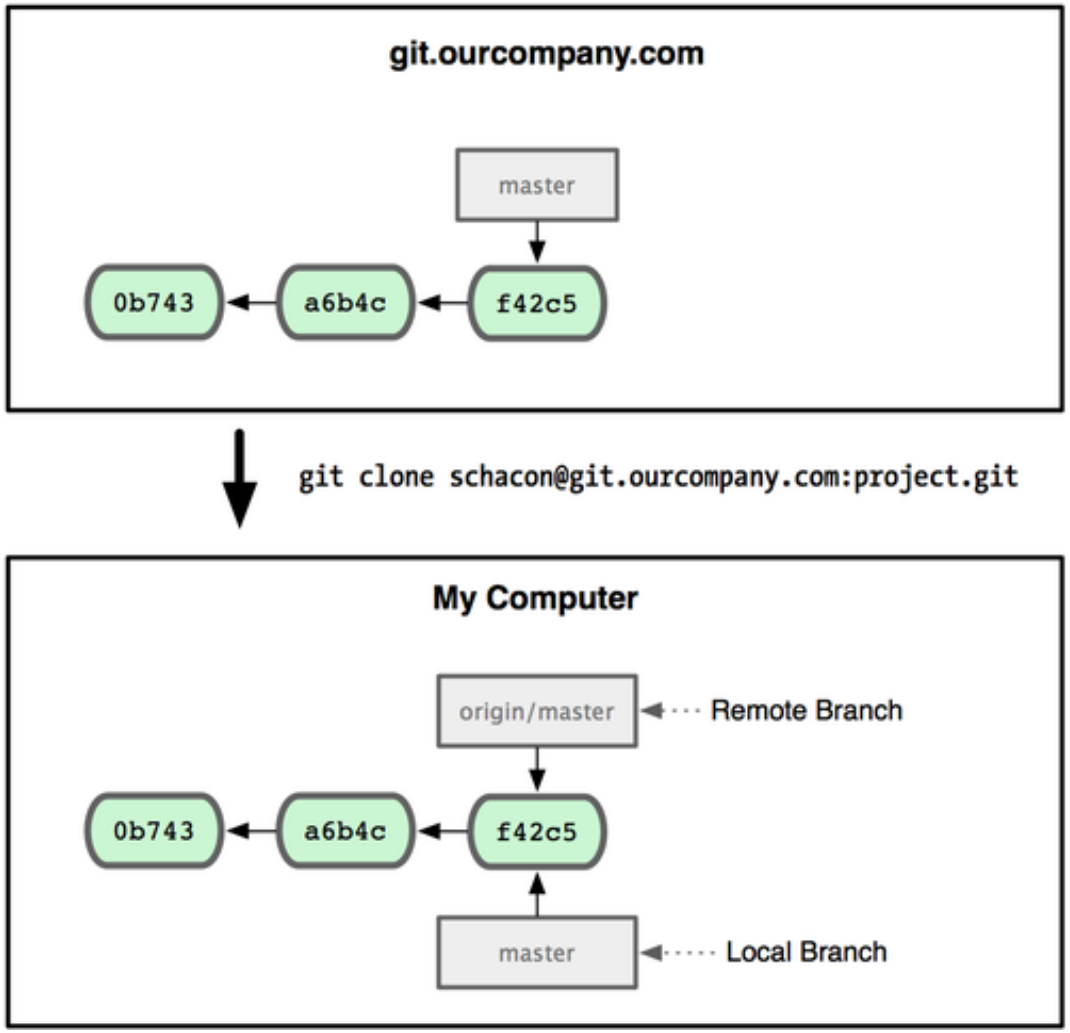
RAMAS REMOTAS



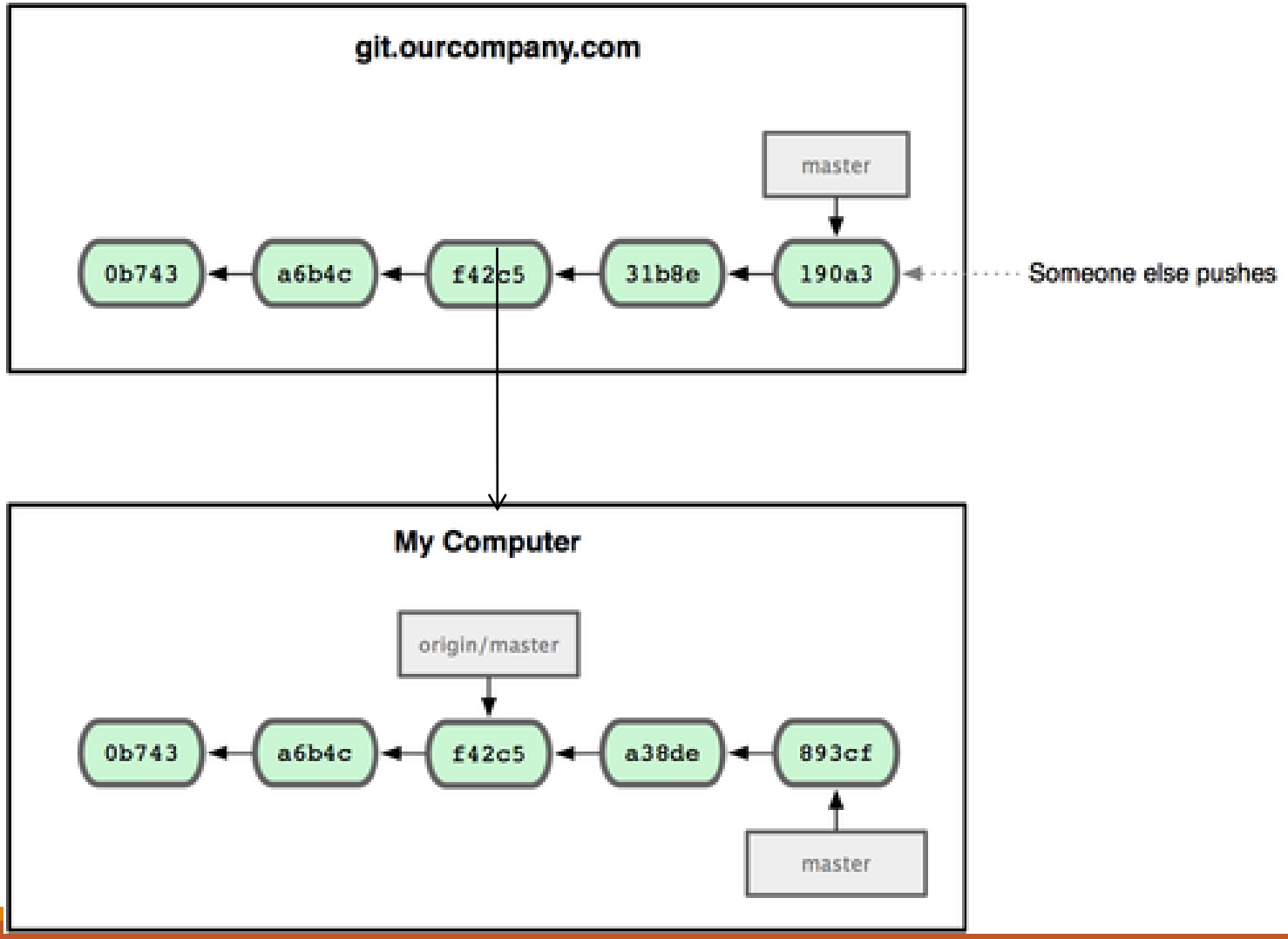
RAMAS REMOTAS

- Las ramas remotas son referencias al estado de las ramas en tus repositorios remotos.
- Son ramas que no se pueden mover, se mueven automáticamente cuando estableces comunicaciones en la red.
- Las ramas remotas funcionan como marcadores, para recordarte en qué estado se encontraban tus repositorios remotos la última vez que contectaste con ellos.

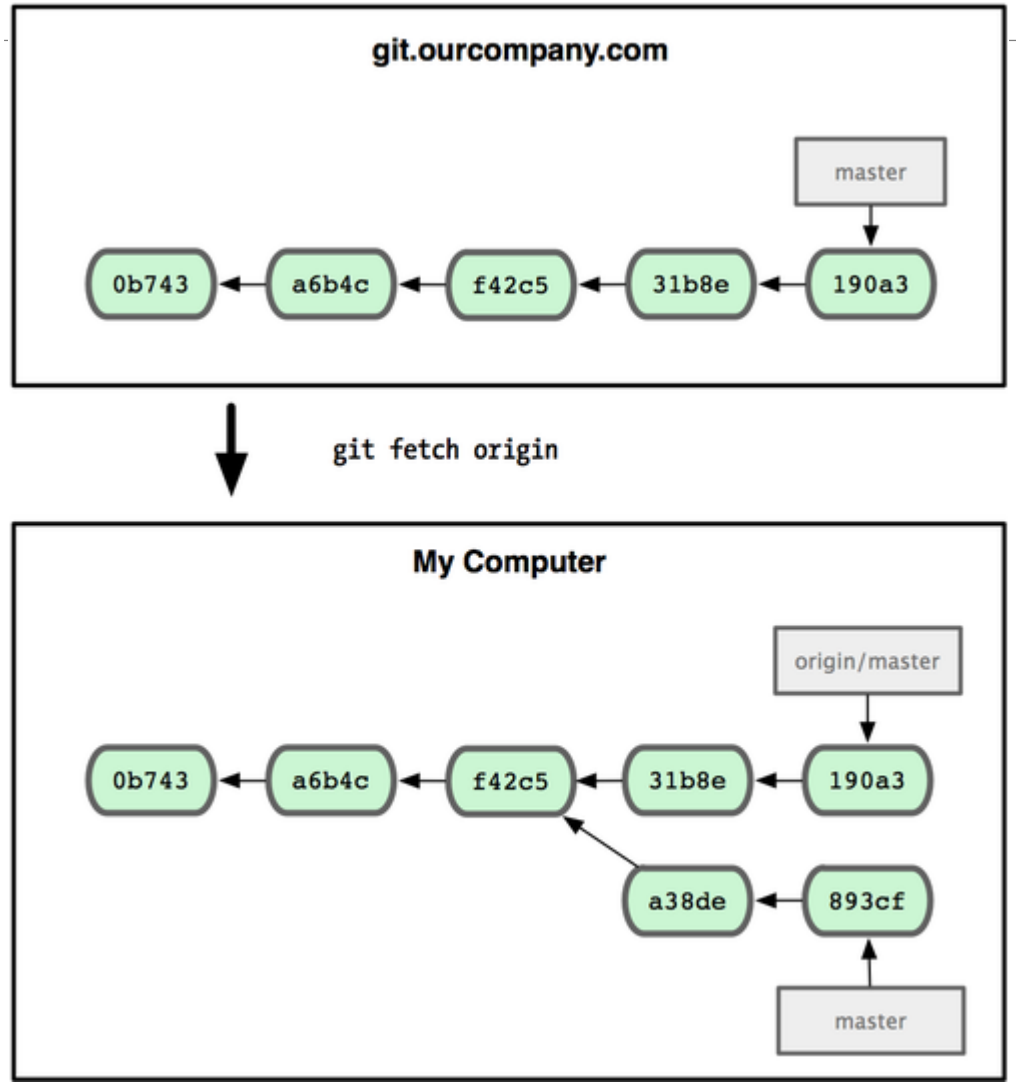
Ramas remotas: clonar



Ramas remotas: distintos avances



Ramas remotas: sincronizar



El comando
git fetch
actualiza tus referencias remotas.

Ramas remotas: publicar

- El comando git push (remoto) (rama) nos permite compartir ramas en un servidor remoto

- El comando git push (remoto) (ramalocal):(ramaremota) es lo mismo pero en el servidor local la rama se llama de forma diferente que el servidor remoto.

Ramas remotas: recuperar (fetch)

- El comando git fetch remoto obtiene una copia del repositorio con las nuevas ramas remotas.
- No obtiene una copia editable local de las mismas. Solamente un puntero no editable.
- Para integrar (merge) esto en tu actual rama de trabajo, puedes usar el comando git merge (remoto)/(rama).

Ramas remotas: SEGUIMIENTO DE RAMAS

- Si activas checkout en una rama local a partir de una rama remota, se crea “una rama de seguimiento” (tracking branch).

git checkout --track origin/rama

- Si estas en una rama de seguimiento:

git push → sabe a que servidor y rama ha de llevar los contenidos.

git pull → si estamos en una rama de seguimiento recupera (fetch) todas las referencias remotas y las consolida (merge) automáticamente en la correspondiente rama remota.

Ramas remotas: Publicar

```
git    push    [nombreremoto]      :[rama]
```