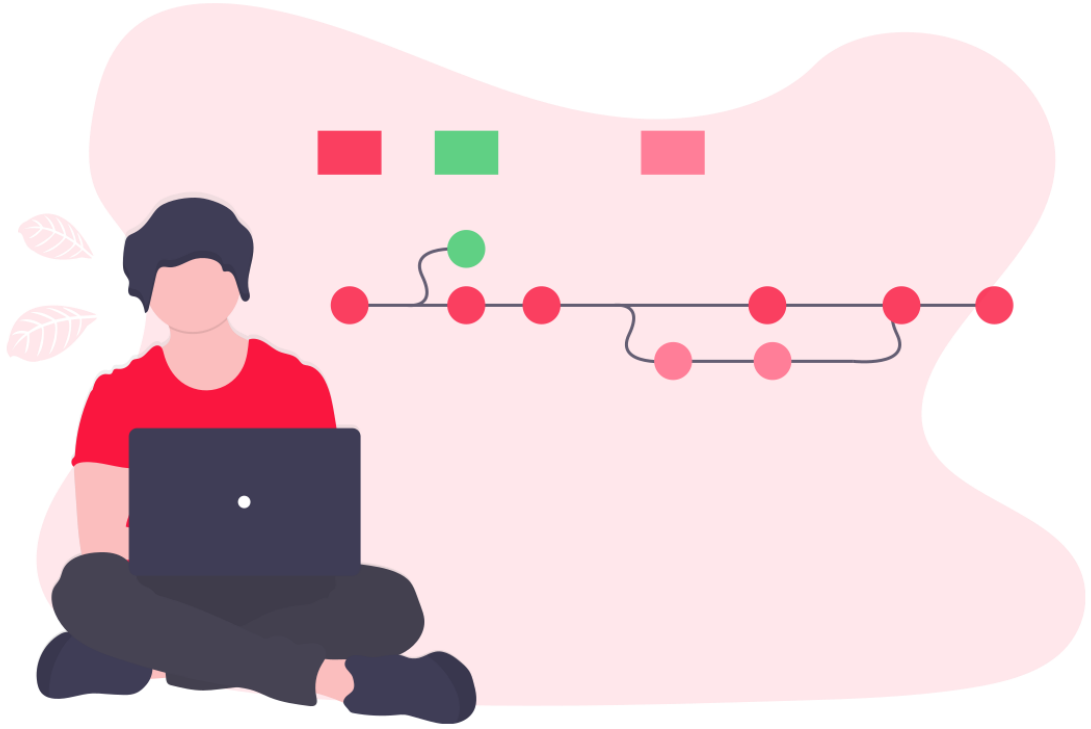
# Git y Github

Control de versiones



# Qué es un sistema de control de versiones?

# Control de versiones

Controla los cambios realizados en un archivo o conjunto de archivos a través del tiempo de modo que puedas recuperar versiones especificas del o de los mismos en cualquier momento.

# Un poco de historia

# Erase una vez en 1991

Linus Torvalds comienza a desarrollar el Kernel de Linux y los cambios en el software que se desarrollaban se hacían a través de parches y descarga de archivos adicionales.

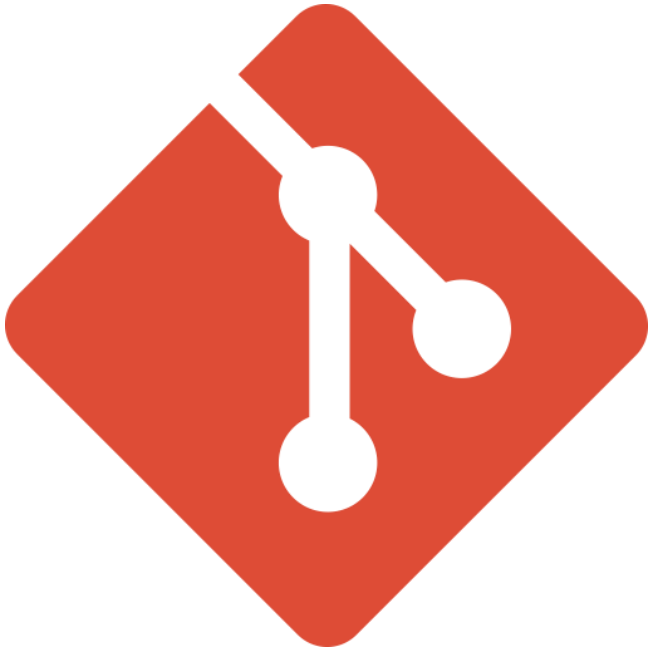
Hasta que en 2002 decide utilizar el sistema de control de versiones *BitKeeper*.
Esta relación duro hasta 2005 cuando *BitKeeper* decide empezar a cobrar por sus servicios.
Y entonces…

Creare mi propio sistema
de Control de versiones

Con juegos de azar y mu jerzuelas
Y será gratis

# Y así nació Git

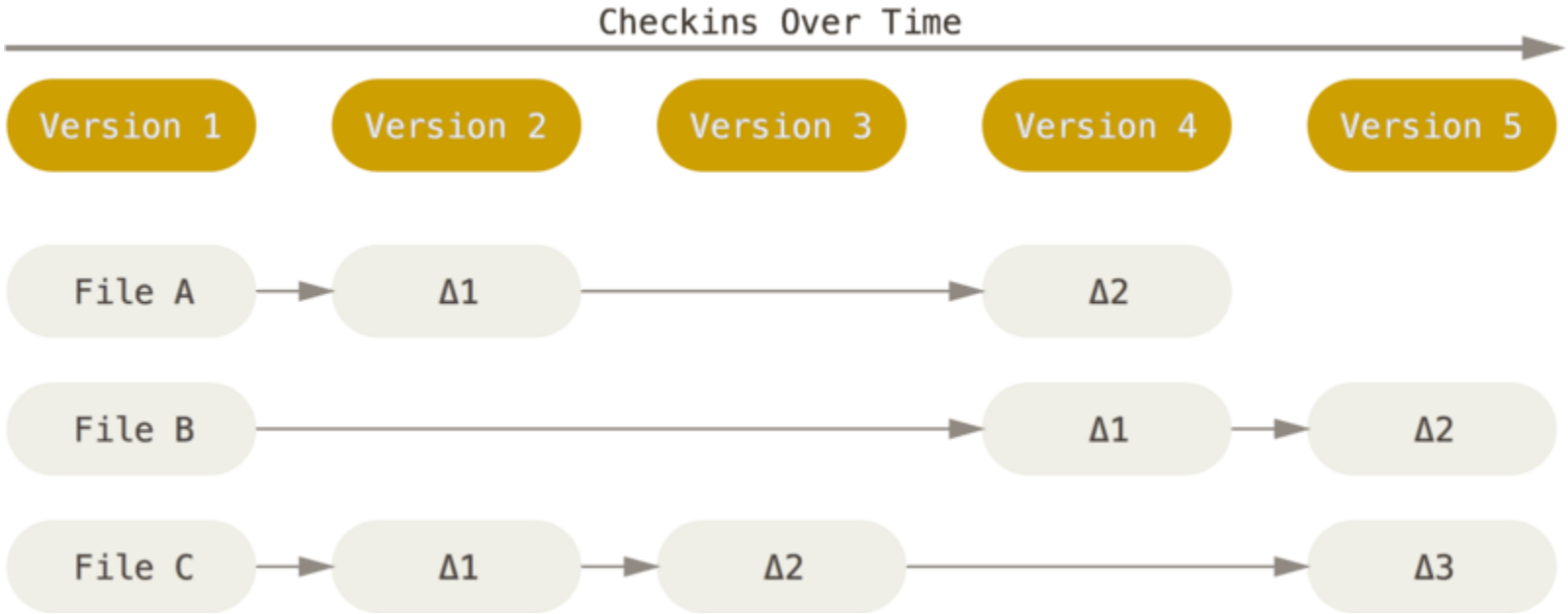
- Velocidad
- Diseño sencillo
- Gran soporte para desarrollo no lineal (miles de ramas paralelas)
- Completamente distribuido
- Capaz de manejar grandes proyectos.

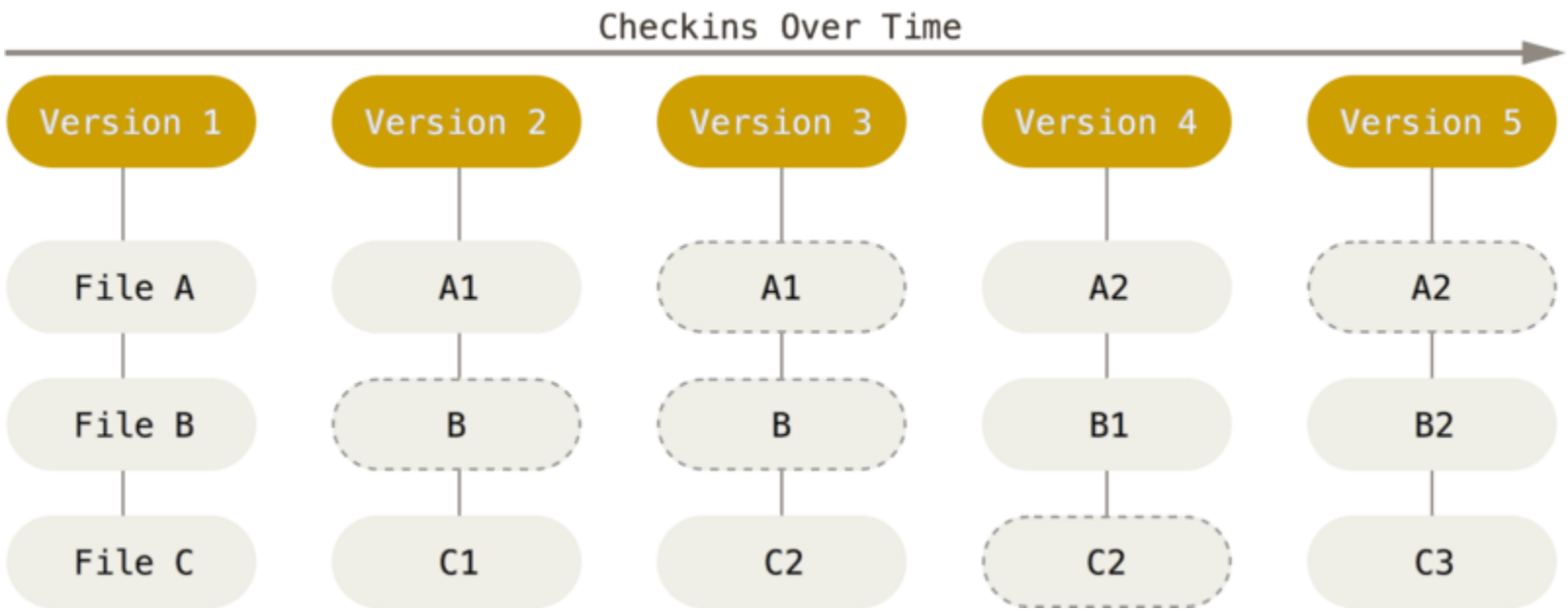


# Cómo Funciona Git ?

# Copias instantáneas no diferenciadas

En cada versión de tu proyecto Git saca una copia instantánea a todos los archivos en miniaturas, aplica un CheckSum para validar el contenido de cada archivo y guarda todo en valores Hash.


Checkins Over Time
Version 1
Version 2
Version 3
Version 4
Version 5
File A
Δ1
Δ2
File B
Δ1
Δ2
File C
Δ1
Δ2
Δ3

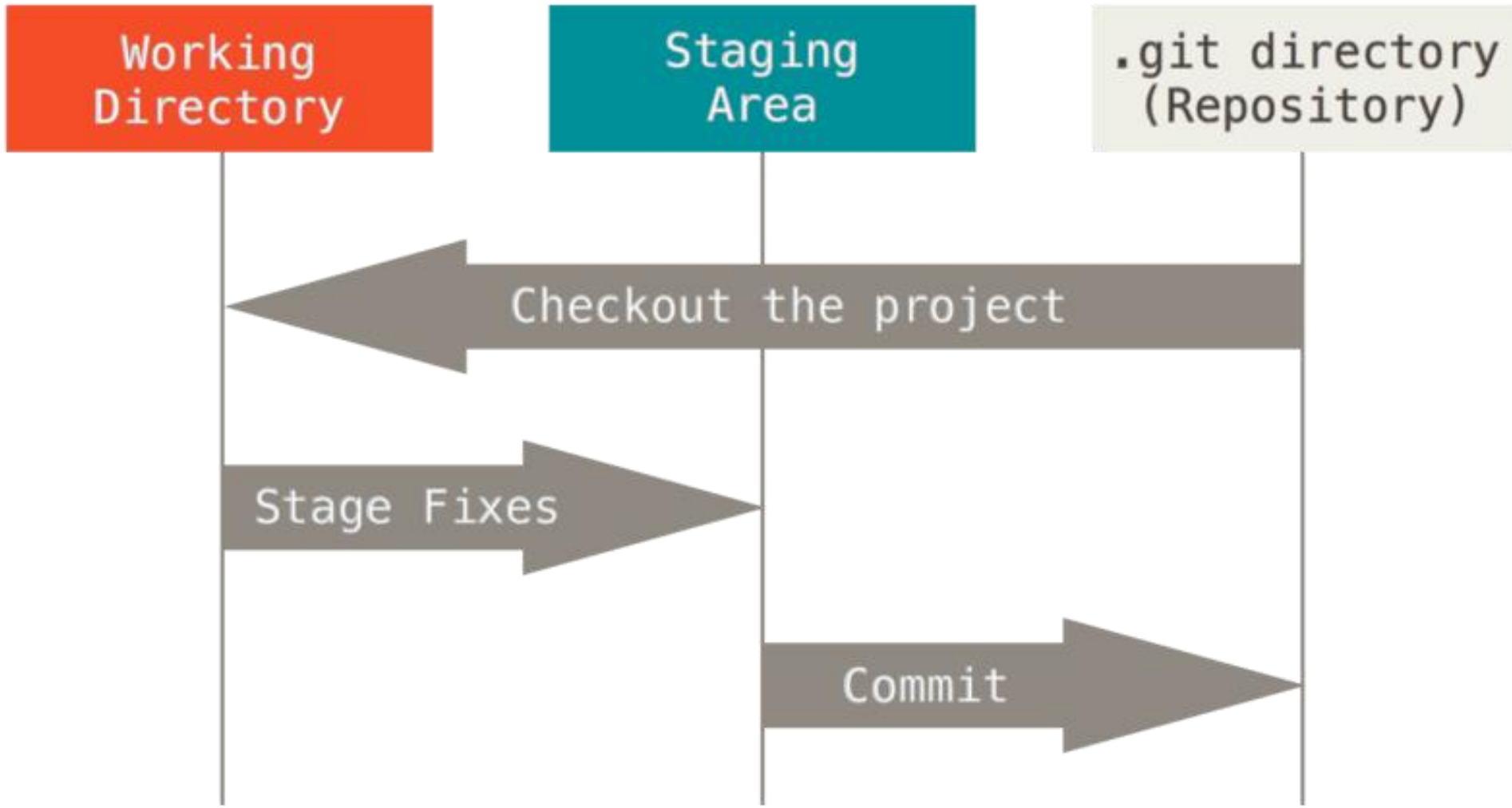

Checkins Over Time
Version 1
Version 2
Version 3
Version 4
Version 5
File A
A1
A1
A2
A2
File B
B
B
B1
B2
File C
C1
C2
C2
C3

# Los tres estados de tus archivos en Git

1. **Confirmado** (Committed): Guardados de manera segura en la base de datos de git.
2. **Modificado** (Modified): Tus archivos se han modificado pero no están confirmados en la base de datos de git.
3. **Preparado** (Staged): Archivos marcados archivos modificados en su versión actual para ser enviados en tu próxima confirmación.

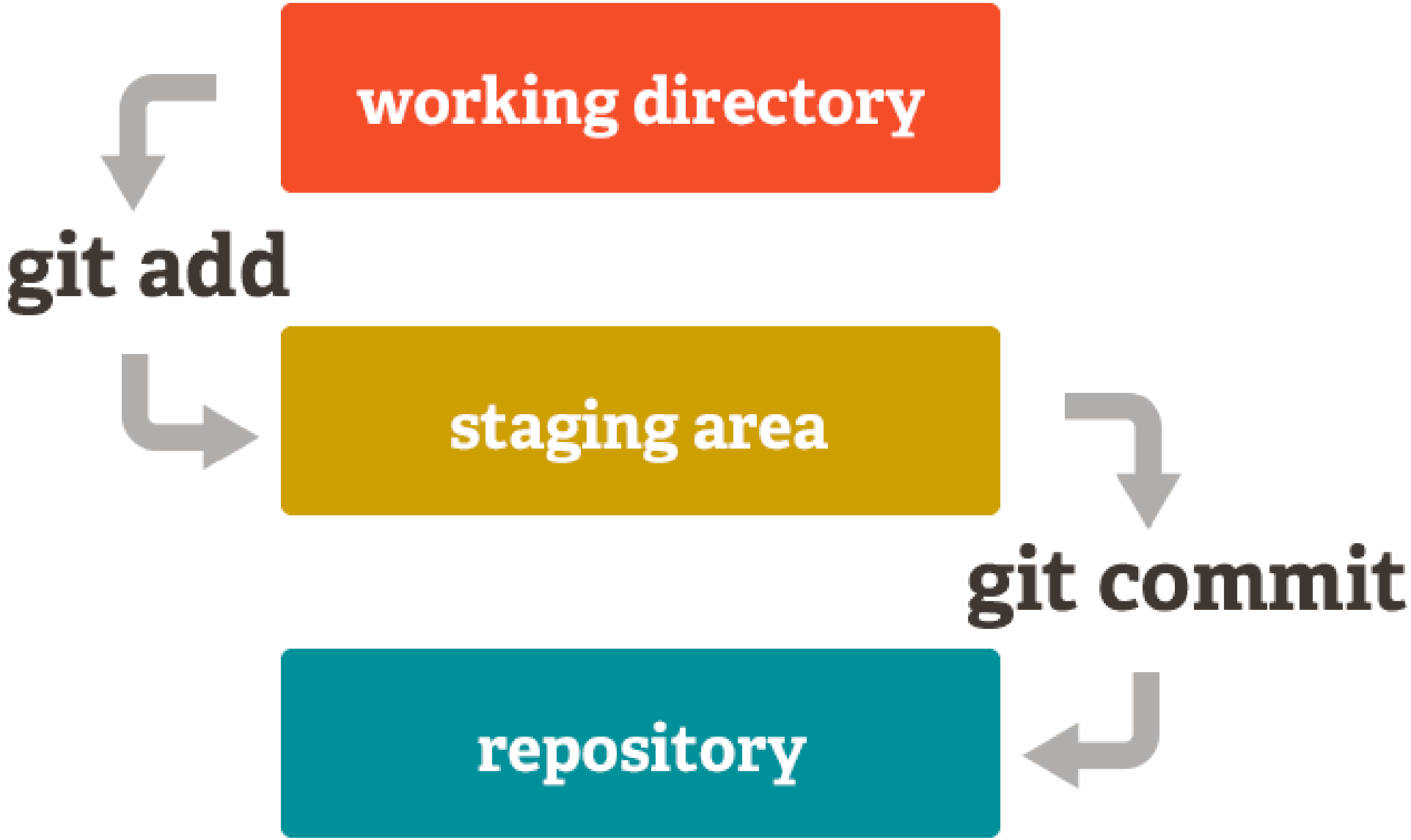
# Los tres estados de un proyecto Git

1. **Directorio de Git** (Git directory): Es donde se almacena tu proyecto, es lo que se copia cuando clonas un repositorio.
2. **Directorio de trabajo** (working directory): es una copia de una versión del proyecto, sus archivos se sacan de la base de datos comprimida de el directorio git que es lo que puedes modificar.
3. **Área de preparación** (staging area): es un archivo, que esta en tu directorio de Git, que almacena información acerca de lo que va a ir en tu próxima confirmación.


Working Directory
Staging Area
.git directory (Repository)
Checkout the project
Stage Fixes
Commit

# Flujo de trabajo en Git

1. Modificas una serie de archivos en tu directorio de trabajo.
2. Preparas los archivos, añadiéndolos a tu área de preparación.
3. Confirmas los cambios, lo que toma los archivos tal y como están en el área de preparación y almacena esa copia instantánea de manera permanente en tu directorio de Git.


working directory
git add
staging area
git commit
repository

# Configuración basica

- **git config --global user.name "John Doe“ :** Establece un nombre de usuario.
- **git config --global user.email johndoe@example.com :** Establece un correo de usuario.
- **git config --list :** Lista todas las configuraciones realizadas.

# Comandos básicos

- **git status** : Muestra el estado del directorio de trabajo.
- **git add <filename>** : Agrega solo un archivo modificado por su nombre al área de preparación.
- **git add *.<extensión>** : Agrega todos los archivos con una misma extensión al área de preparación.
- **git add .** : Agrega todos los archivos modificados al área de preparación.
- **git commit –m “<mensaje>”**: Guarda los cambios del área de preparación en el directorio git.
- **git log ––stat**: Muestra el historial de confirmaciones.

# Deshacer cosas

- **git commit --amend :** Rehacer una confirmación.
- **git restore --staged <file> :** Recupera un archivo agregado al área de preparación.
- **git checkout <cod. commit> <file>:** Deshace los cambios de un archivo.
- **git reset --hard HEAD~1:** Elimina el ultimo commit.

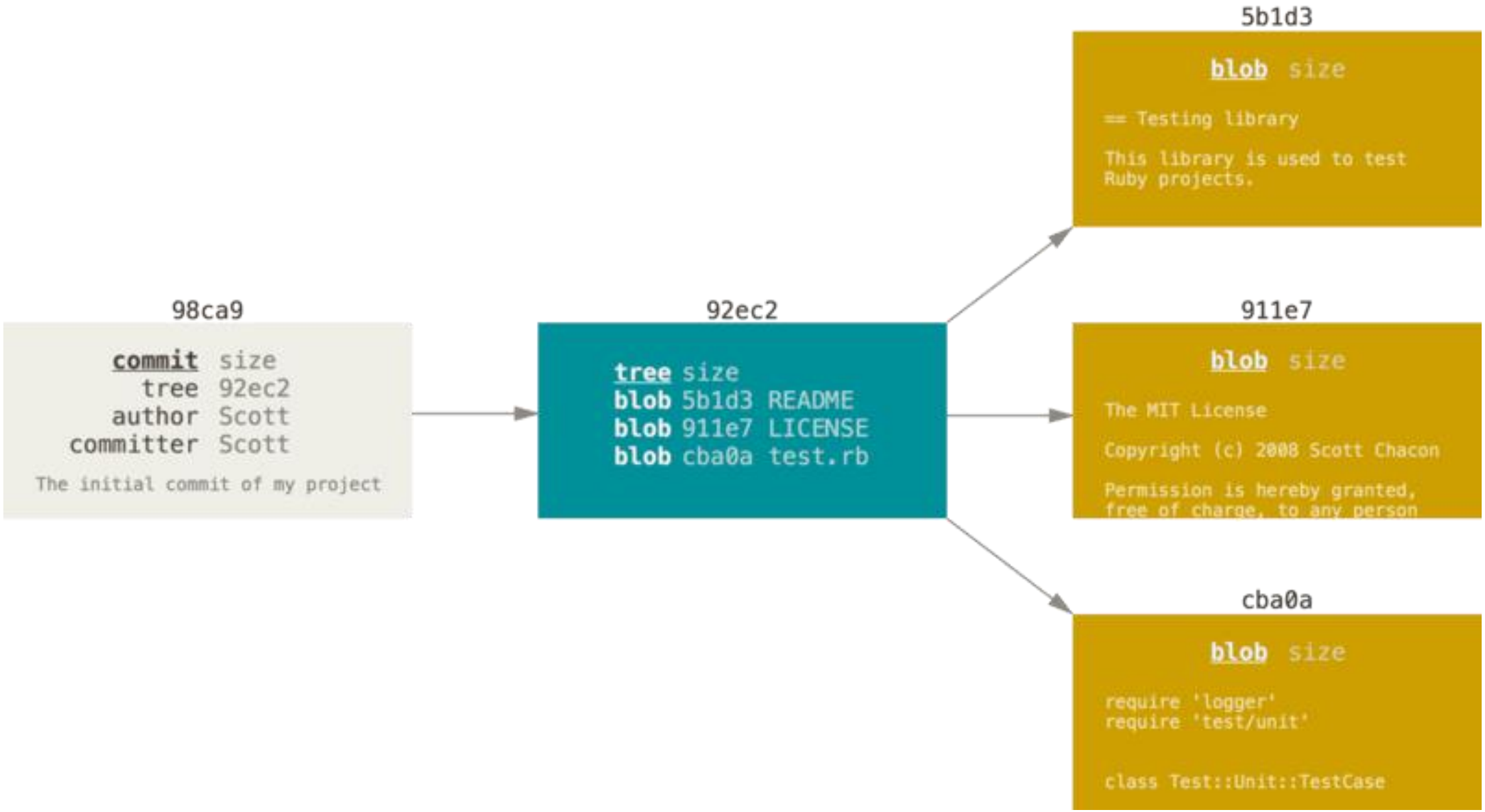
# .gitignore

Dentro de un proyecto Git podemos crear un archivo con el nombre de “.gitignore” en el cual podremos especificar archivos o carpetas que no queremos que entren dentro del flujo de trabajo de git.

# Ramas

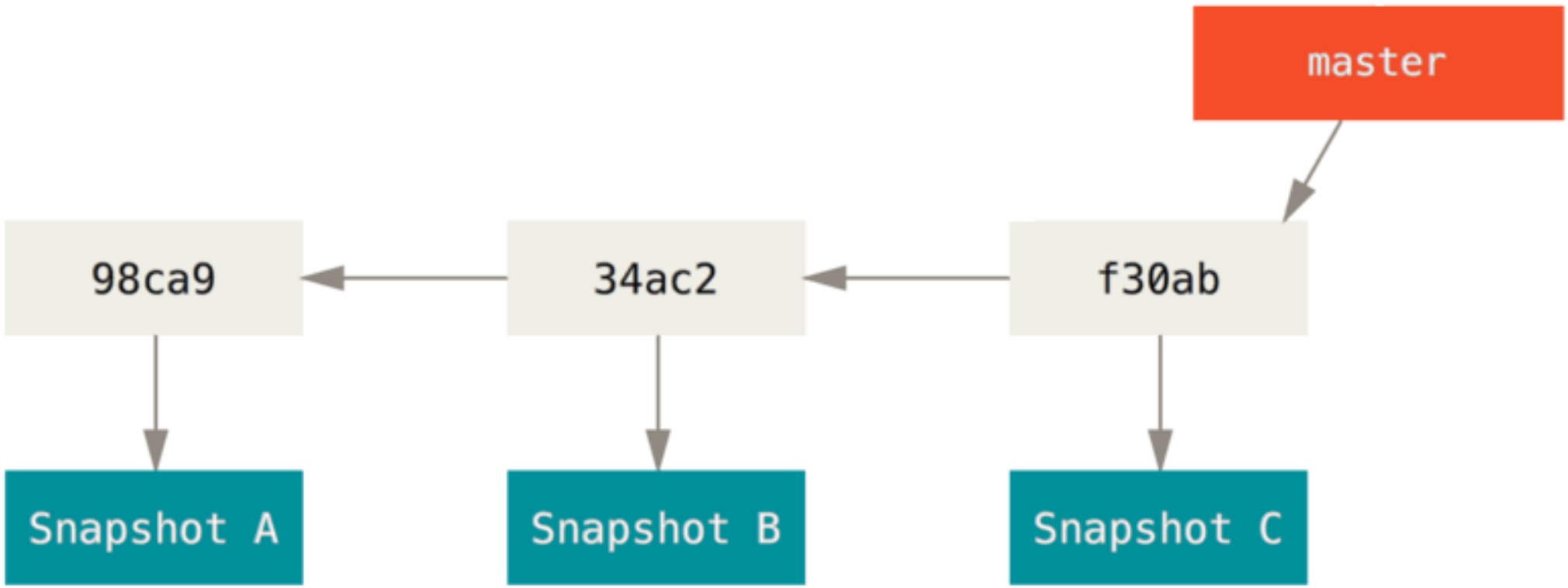
# Bloobs, Raíces y apuntadores

Cuando preparamos los cambios realizados en uno o varios archivos de nuestro proyecto git, se realiza una suma de control para validarlos y se generar sus direcciones hash-1.
Al realizar la confirmación los archivos se almacenan en el directorio git como copias instantáneas que se denominan Bloobs, luego se apunta las direcciones hash de cada archivo en un archivo raíz y finalmente se crea un apuntador que contiene los metadatos de la confirmación y que apunta hacia el archivo raíz.


98ca9
commit size
tree 92ec2
author Scott
committer Scott
The initial commit of my project
92ec2
tree size
blob 5b1d3 README
blob 911e7 LICENSE
blob cba0a test.rb
5b1d3
blob size
== Testing library
This library is used to test
Ruby projects.
911e7
blob size
The MIT License
Copyright (c) 2008 Scott Chacon
Permission is hereby granted,
free of charge, to any person
cba0a
blob size
require 'logger'
require 'test/unit'
class Test::Unit::TestCase

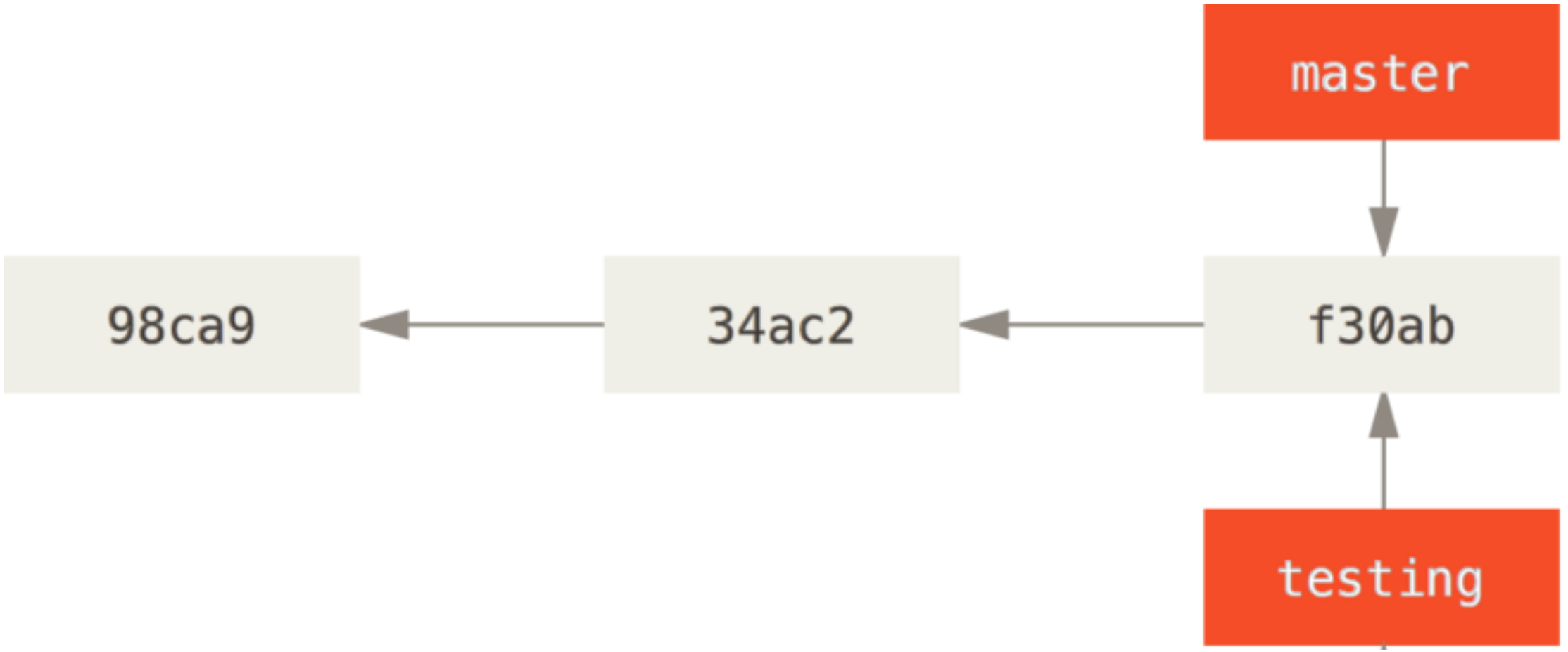
# Qué es una rama?

Es un apuntador móvil que ira avanzando que apunta a cada confirmación que vayamos haciendo; con la primera confinación que realicemos se creara la rama “master“ por defecto.


master
98ca9
34ac2
f30ab
Snapshot A
Snapshot B
Snapshot C

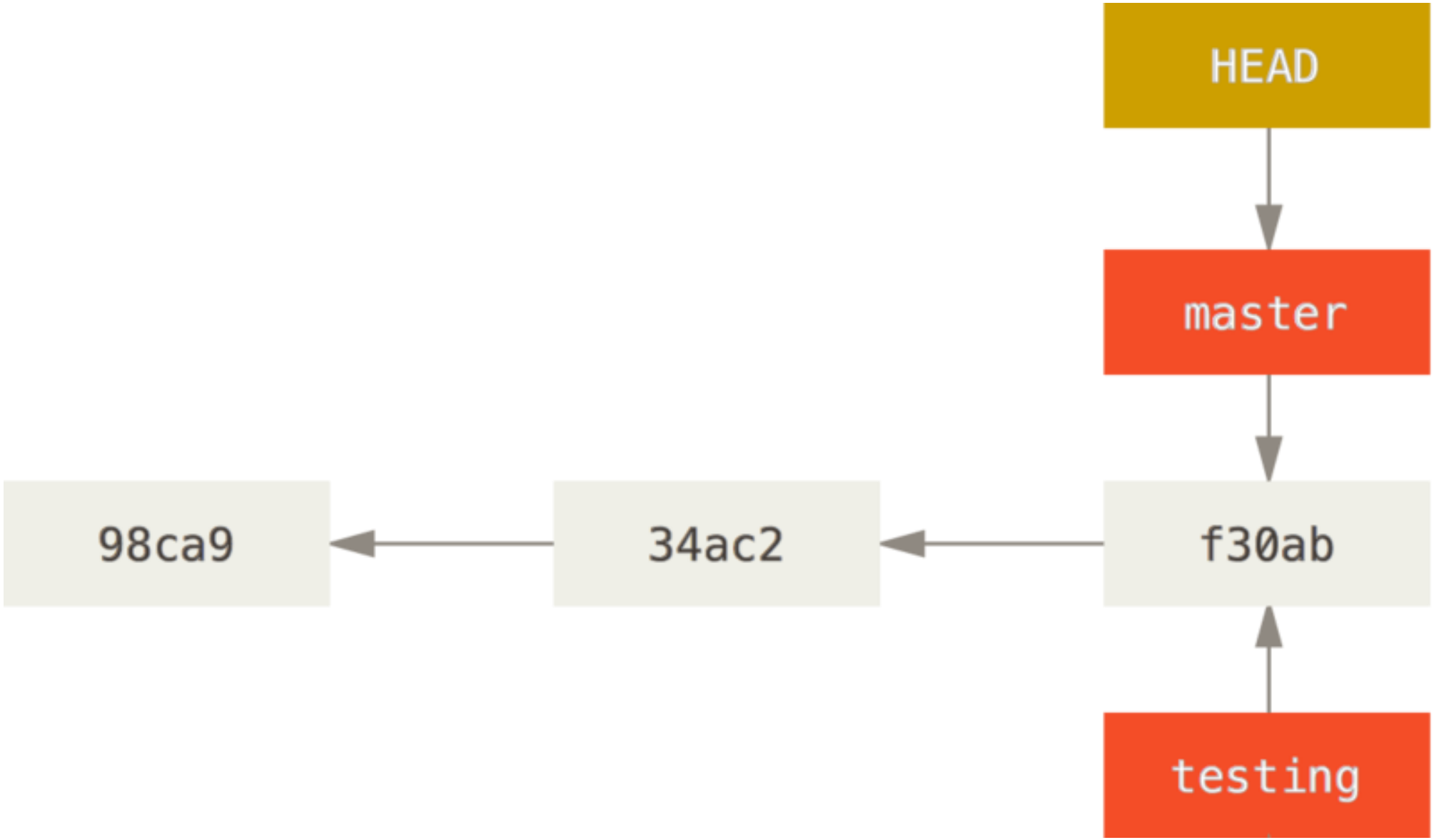
# Qué pasa cuando creamos una nueva rama?

Simplemente se crea un apuntador que apunte a la ultima confirmación que realizamos


master
98ca9
34ac2
f30ab
testing

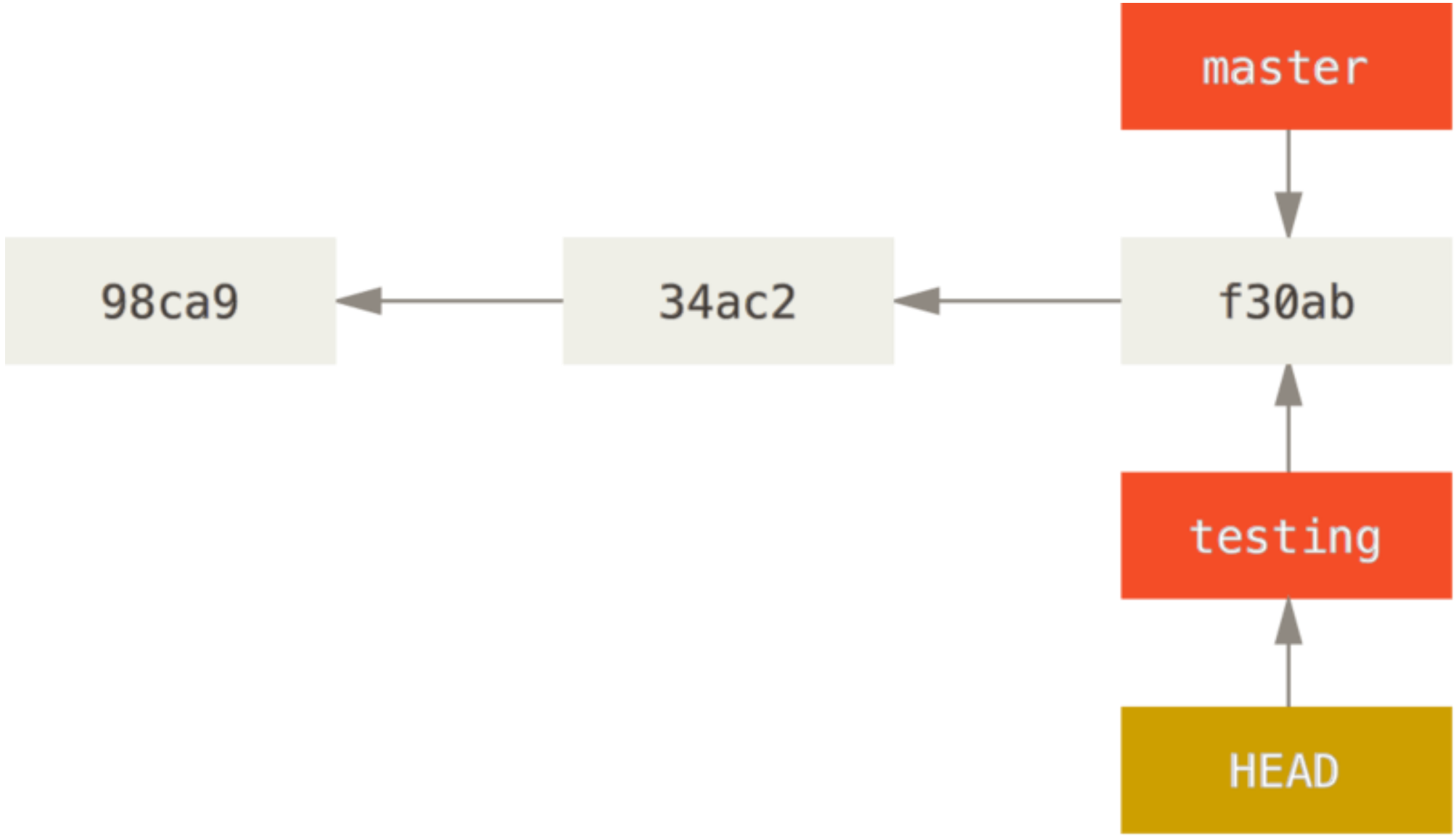
# Cómo sabe git en que rama esta trabajando?

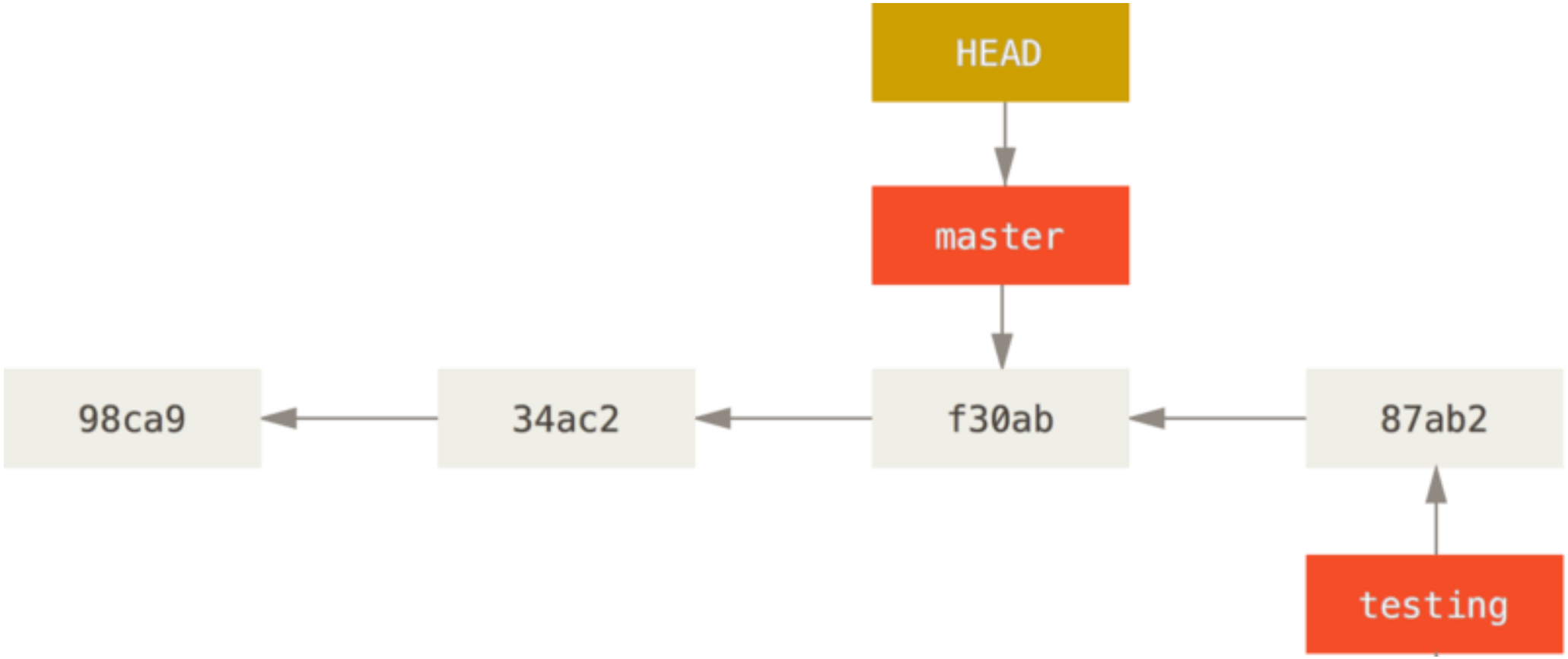
Con un apuntador especial denominada HEAD, este apuntador apunta a la rama local en la que se este trabajando.


HEAD
master
98ca9
34ac2
f30ab
testing

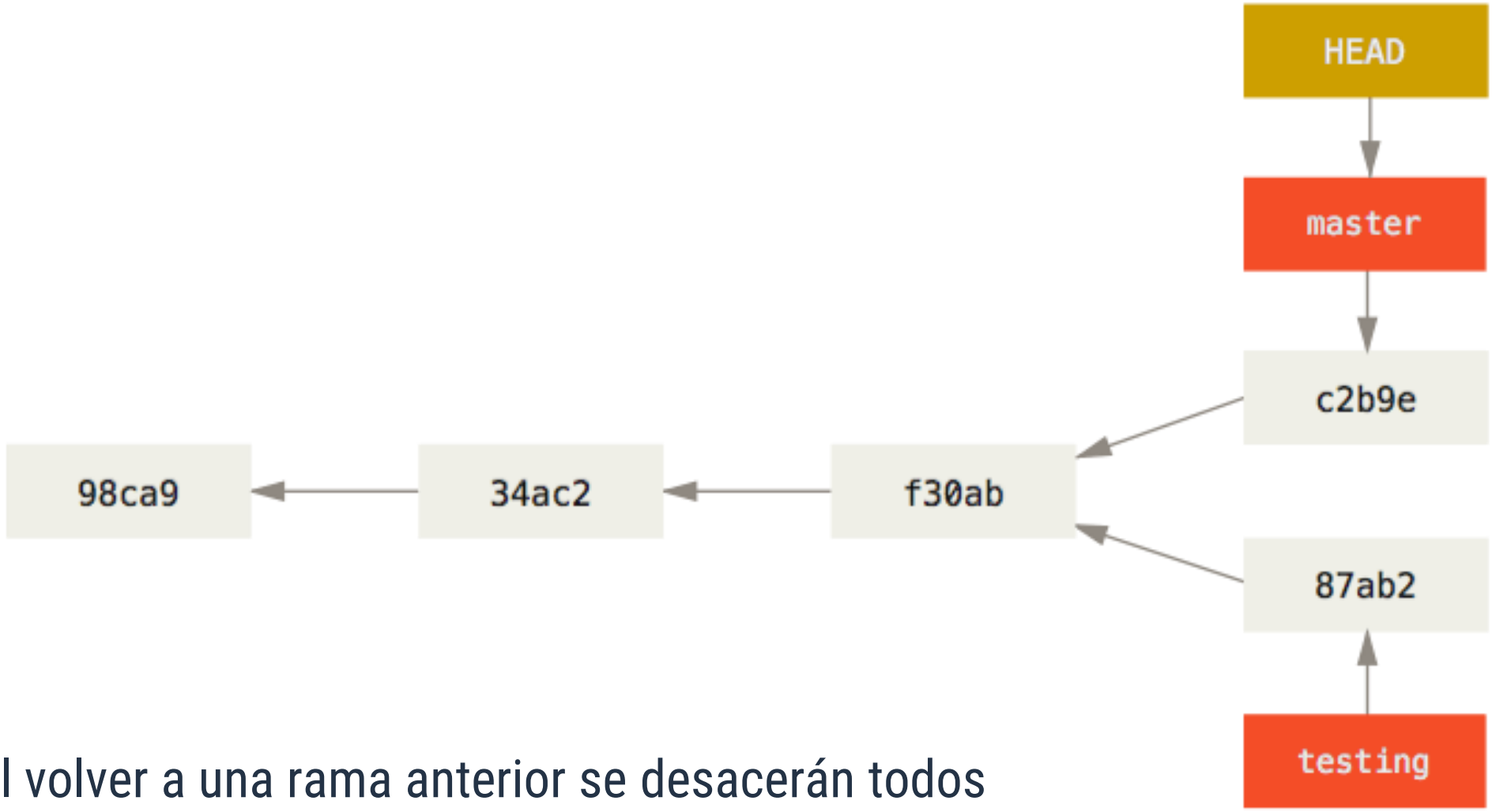
# Qué pasa cuando nos movemos entre ramas?

Simplemente el apuntador especial HEAD apunta a otra rama.

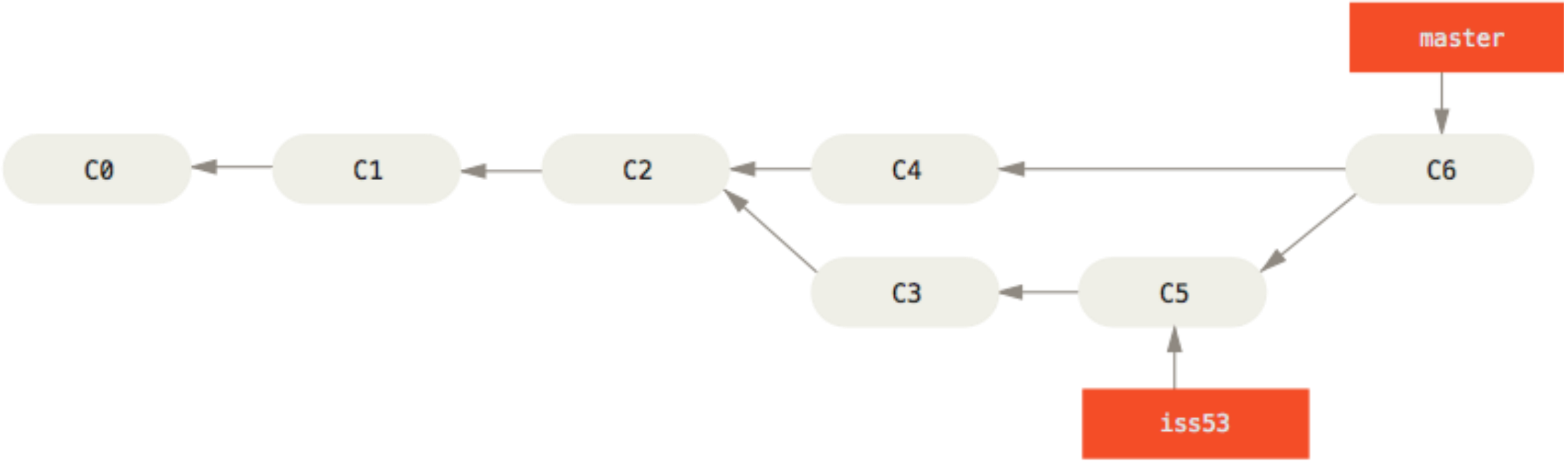

master
98ca9
34ac2
f30ab
testing
HEAD




La nueva rama avanzara con las nuevas confirmaciones que hagamos.




Al volver a una rama anterior se desacerán todos los cambios confirmados en la otra rama.




Fusión: al fusionar las ramas concatenamos el contenido de las confinaciones de la rama que queremos fusionar a la rama fusionada y el puntero HEAD termina apuntado a la rama fusionada.


Flujo de trabajo para proyectos en Git.

# Comandos para trabajar con ramas

- **git branch <nombre>** : Crea una nueva rama.
- **git checkout <nombre>**: Cambia de rama.
- **git merge <nombre>** : Concatena la rama <nombre> con la rama actual.
- **git log --oneline --decorate --graph –all** : Muestra el detalle de las confirmaciones por rama.

GitHub

# Que es GitHub?

GitHub es el mayor proveedor de alojamiento de repositorios Git, y es el punto de encuentro para que millones de desarrolladores colaboren en el desarrollo de sus proyectos

# Comandos para trabajar con Git y Github

- **git remote add origin <url>** : Enlaza un repositorio de github con tu directorio de trabajo git local.
- **git push origin master** : Envía todas tus confirmaciones y archivos locales a tu repositorio en github.
- **git clone <url>** : Clona un repositorio en Github a una carpeta de trabajo local.
- **git fetch** : Sincroniza tu repositorio de Github con tu directorio Git.
- **git pull origin master** : Concatena todas las confirmaciones de tu repositorio sincronizado con tu rama actual.