Pro Git, el libro oficial de Git

3.1. ¿Qué es una rama?

Para entender realmente cómo ramifica Git, previamente hemos de examinar la forma en que almacena sus datos. Recordando lo citado en el capítulo 1, Git no los almacena de forma incremental (guardando solo diferencias), sino que los almacena como una serie de instantáneas (copias puntuales de los archivos completos, tal y como se encuentran en ese momento).

En cada confirmación de cambios (*commit*), Git almacena un punto de control que conserva: un apuntador a la copia puntual de los contenidos preparados (*staged*), unos metadatos con el autor y el mensaje explicativo, y uno o varios apuntadores a las confirmaciones (*commit*) que sean padres directos de esta (un padre en los casos de confirmación normal, y múltiples padres en los casos de estar confirmando una fusión (*merge*) de dos o mas ramas).

Para ilustrar esto, vamos a suponer, por ejemplo, que tienes una carpeta con tres archivos, que preparas (*stage*) todos ellos y los confirmas (*commit*). Al preparar los archivos, Git realiza una suma de control de cada uno de ellos (un resumen SHA-1, tal y como se mencionaba en el capítulo 1), almacena una copia de cada uno en el repositorio (estas copias se denominan "*blobs*"), y guarda cada suma de control en el área de preparación (*staging area*):

```
$ git add README test.rb LICENSE
$ git commit -m 'initial commit of my project'
```

Cuando creas una confirmación con el comando `git commit`, Git realiza sumas de control de cada subcarpeta (en el ejemplo, solamente tenemos la carpeta principal del proyecto), y guarda en el repositorio Git una copia de cada uno de los archivos contenidos en ella/s. Después, Git crea un objeto de confirmación con los metadatos pertinentes y un apuntador al nodo correspondiente del árbol de proyecto. Esto permitirá poder regenerar posteriormente dicha instantánea cuando sea necesario.

En este momento, el repositorio de Git contendrá cinco objetos: un "*blob*" para cada uno de los tres archivos, un árbol con la lista de contenidos de la carpeta (más sus respectivas relaciones con los "*blobs*"), y una

Indice de contenidos

confirmación de cambios (*commit*) apuntando a la raiz de ese árbol y conteniendo el resto de metadatos pertinentes. Conceptualmente, el contenido del repositorio Git será algo parecido a la Figura 3-1

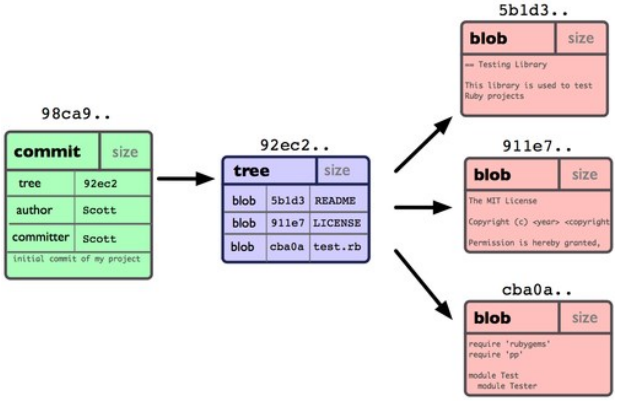

Figura 3.1 Datos en el repositorio tras una confirmación sencilla

Si haces más cambios y vuelves a confirmar, la siguiente confirmación guardará un apuntador a esta su confirmación precedente. Tras un par de confirmaciones más, el registro ha de ser algo parecido a la Figura 3-2.

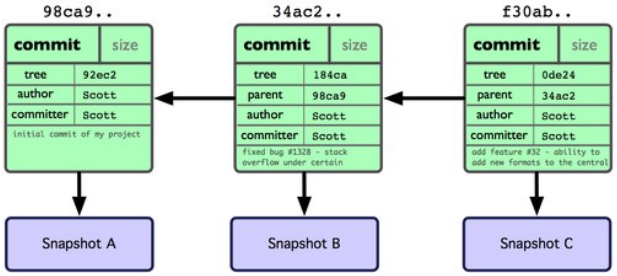

Figura 3.2 Datos en el repositorio tras una serie de confirmaciones

Una rama Git es simplemente un apuntador móvil apuntando a una de esas confirmaciones. La rama por defecto de Git es la rama `master`. Con la primera confirmación de cambios que realicemos, se creará esta rama principal `master` apuntando a dicha confirmación. En cada confirmación de cambios que realicemos, la rama irá avanzando automáticamente. Y la rama `master` apuntará siempre a la última confirmación realizada.

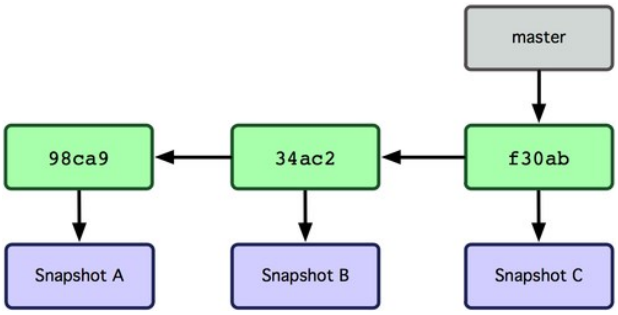

Figura 3.3 Apuntadores en el registro de confirmaciones de una rama

¿Qué sucede cuando creas una nueva rama? Pues que simplemente se crea un nuevo apuntador para que lo puedas mover libremente. Por ejemplo, si quieres crear una nueva rama denominada `testing` usarás el comando `git branch`:

```
$ git branch testing
```

Esto creará un nuevo apuntador apuntando al mismo *commit* donde estés actualmente (ver Figura 3-4).

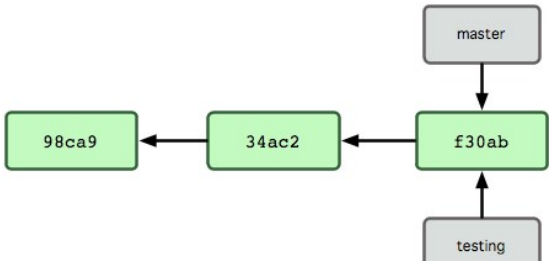

Figura 3.4 Apuntadores de varias ramas en el registro de confirmaciones de cambio

Y, ¿cómo sabe Git en qué rama estás en este momento? Pues mediante un apuntador especial denominado `HEAD`. Aunque es preciso comentar que este `HEAD` es totalmente distinto al concepto de `HEAD` en otros sistemas de control de cambios como Subversion o CVS. En Git, es simplemente el apuntador a la rama local en la que tú estés en ese momento. En este caso, en la rama `master`. Puesto que el comando `git branch` solamente crea una nueva rama, y no salta a dicha rama.

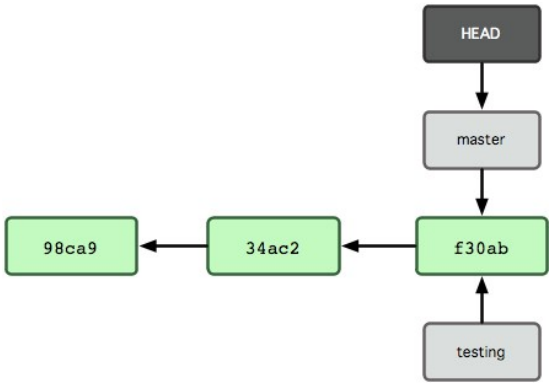

Figura 3.5 Apuntador HEAD a la rama donde estás actualmente

Para saltar de una rama a otra, tienes que utilizar el comando `git checkout`. Hagamos una prueba, saltando a la rama `testing` recién creada:

```
$ git checkout testing
```

Esto mueve el apuntador `HEAD` a la rama `testing` (ver Figura 3-6).

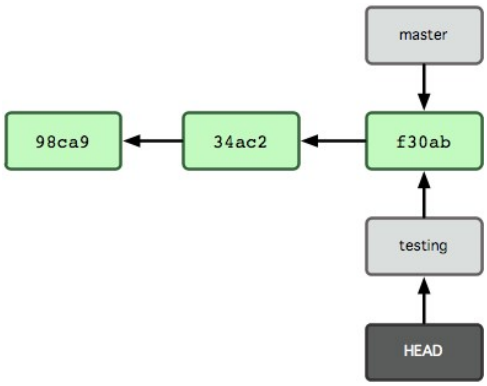

Figura 3.6 Apuntador HEAD apuntando a otra rama cuando saltamos de rama

¿Cuál es el significado de todo esto?. Lo veremos tras realizar otra confirmación de cambios:

```
$ vim test.rb
$ git commit -a -m 'made a change'
```

La Figura 3-7 ilustra el resultado.

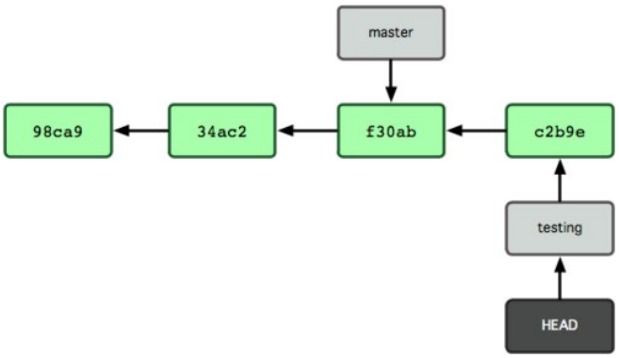

Figura 3.7 La rama apuntada por HEAD avanza con cada confirmación de cambios

Observamos algo interesante: la rama `testing` avanza, mientras que la rama `master` permanece en la confirmación donde estaba cuando lanzaste el comando `git checkout` para saltar. Volvamos ahora a la rama `master`:

```
$ git checkout master
```

La Figura 3-8 muestra el resultado.

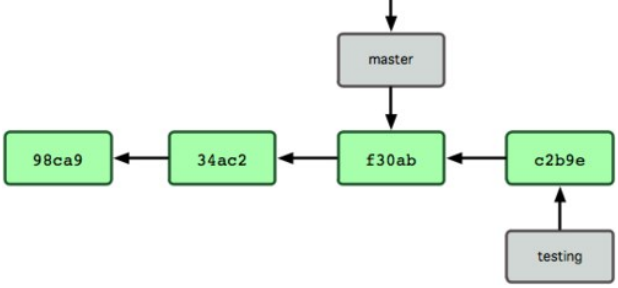

Figura 3.8 HEAD apunta a otra rama cuando hacemos un checkout

Este comando realiza dos acciones: Mueve el apuntador `HEAD` de nuevo a la rama `master`, y revierte los archivos de tu directorio de trabajo; dejandolos tal y como estaban en la última instantánea confirmada en dicha rama `master`. Esto supone que los cambios que hagas desde este momento en adelante divergerán de la antigua versión del proyecto. Básicamente, lo que se está haciendo es rebobinar el trabajo que habias hecho temporalmente en la rama `testing`; de tal forma que puedas avanzar en otra dirección diferente.

Haz algunos cambios más y confirmalos:

```
$ vim test.rb
$ git commit -a -m 'made other changes'
```

Ahora el registro de tu proyecto diverge (ver Figura 3-9). Has creado una rama y saltado a ella, has trabajado sobre ella; has vuelto a la rama original, y has trabajado también sobre ella. Los cambios realizados en ambas sesiones de trabajo están aislados en ramas independientes: puedes saltar libremente de una a otra según estimes oportuno. Y todo ello simplemente con dos comandos: `git branch` y `git checkout`.

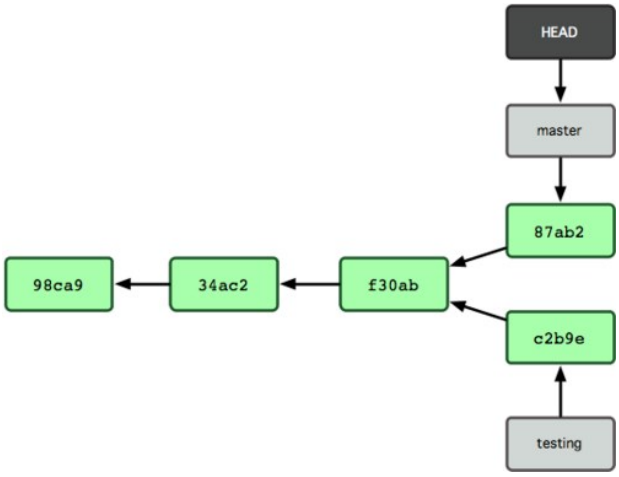

Figura 3.9 Los registros de las ramas divergen

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

Esto contrasta fuertemente con los métodos de ramificación usados por otros sistemas de control de versiones. En los que crear una nueva rama supone el copiar todos los archivos del proyecto a una nueva carpeta adiccional. Lo que puede llevar segundos o incluso minutos, dependiendo del tamaño del proyecto. Mientras que en Git el proceso es siempre instantáneo. Y, además, debido a que se almacenan tambien los nodos padre para cada confirmación, el encontrar las bases adecuadas para realizar una fusión entre ramas es un proceso automático y generalmente sencillo de realizar. Animando así a los desarrolladores a utilizar ramificaciones frecuentemente.

Y vamos a ver el por qué merece la pena hacerlo así.