Chapter 3

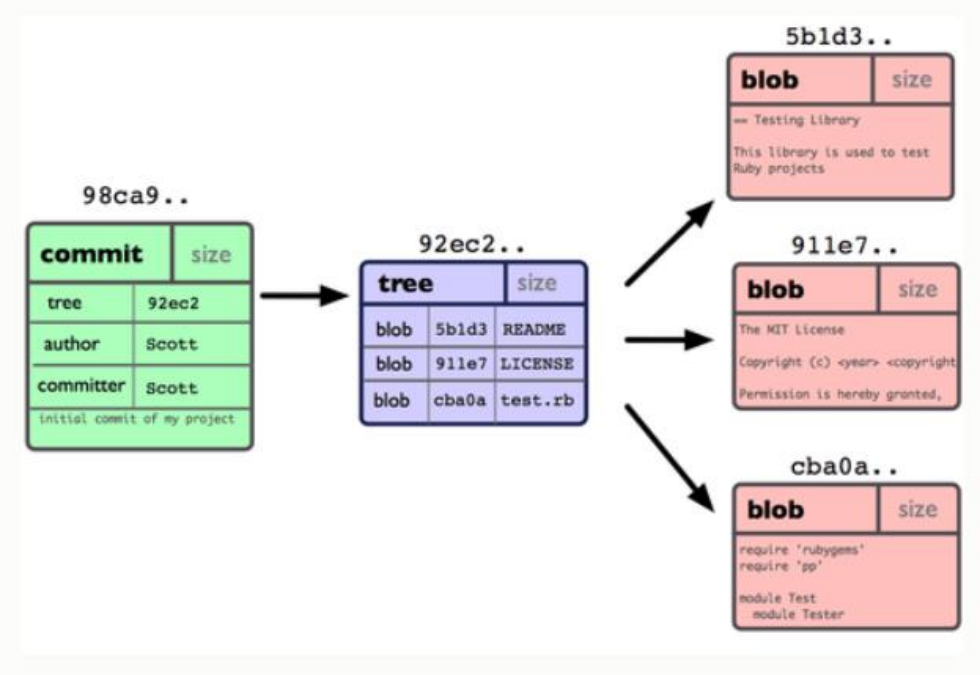
Ramificaciones en Git

Cuando hablamos de ramificaciones, significa que tu has tomado la rama principal de desarrollo (master) y a partir de ahí has continuado trabajando sin seguir la rama principal de desarrollo.

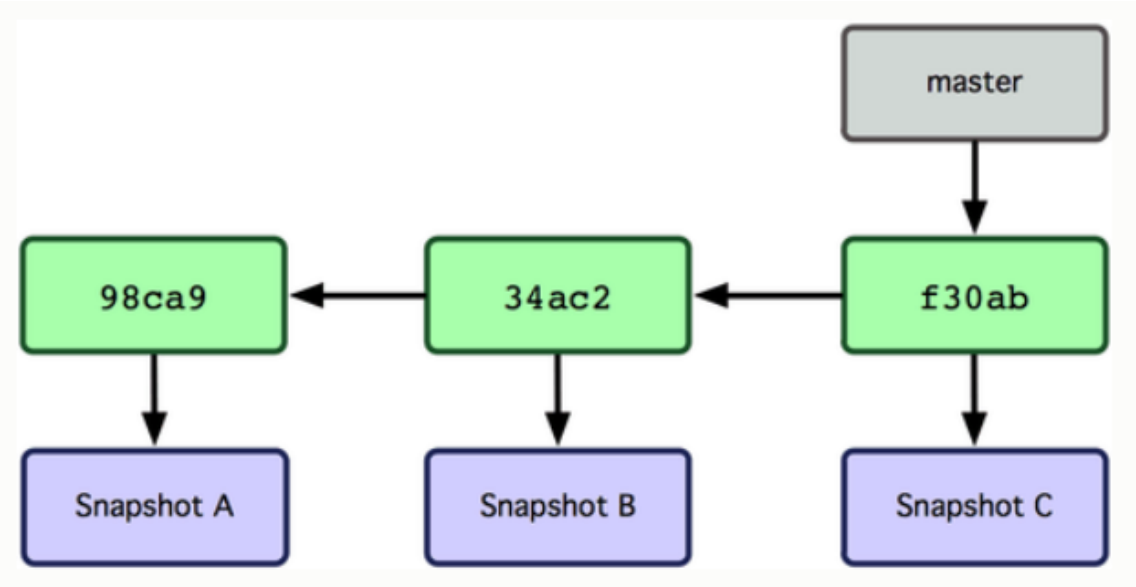
A diferencia de otros sistemas de control de versiones, Git promueve un ciclo de desarrollo donde las ramas se crean y se unen ramas entre sí, incluso varias veces en el mismo día.

3.1 Ramificaciones en Git - ¿Qué es una rama?

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.



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 queda así:



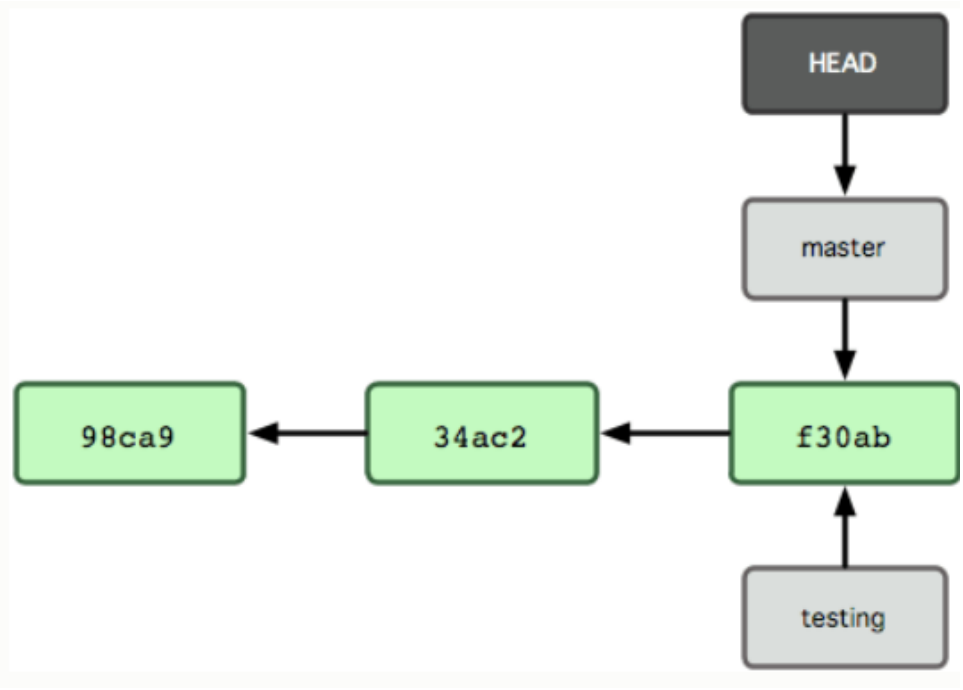
¿Qué sucede cuando creas una nueva rama?

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 a la misma confirmación donde estés actualmente:



Y, ¿cómo sabe Git en qué rama estás en este momento?

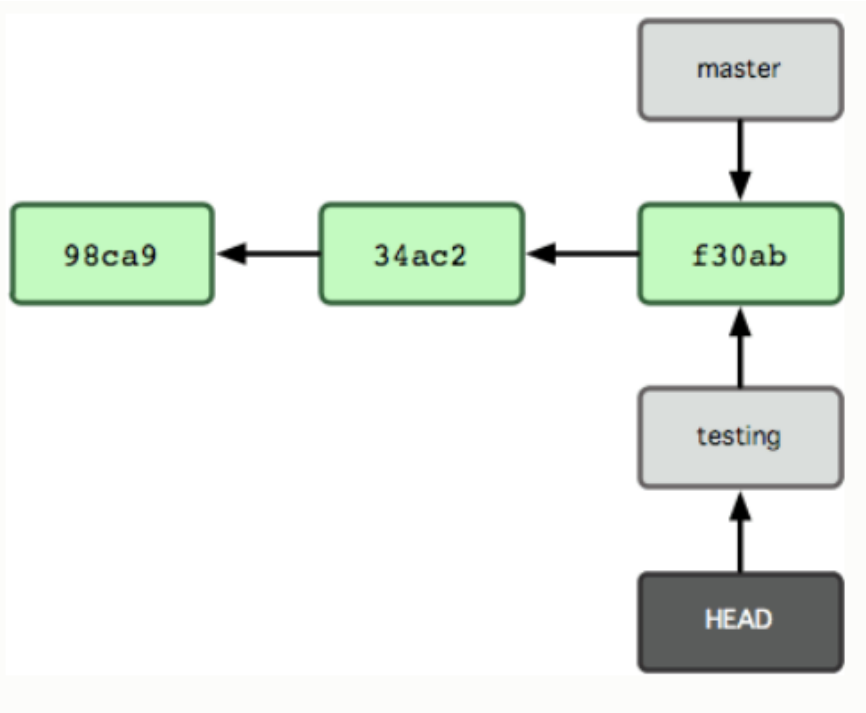
Pues, mediante un apuntador especial denominado HEAD.

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.**

Para saltar de una rama a otra, tienes que utilizar el comando:

```
$ git checkout testing
```

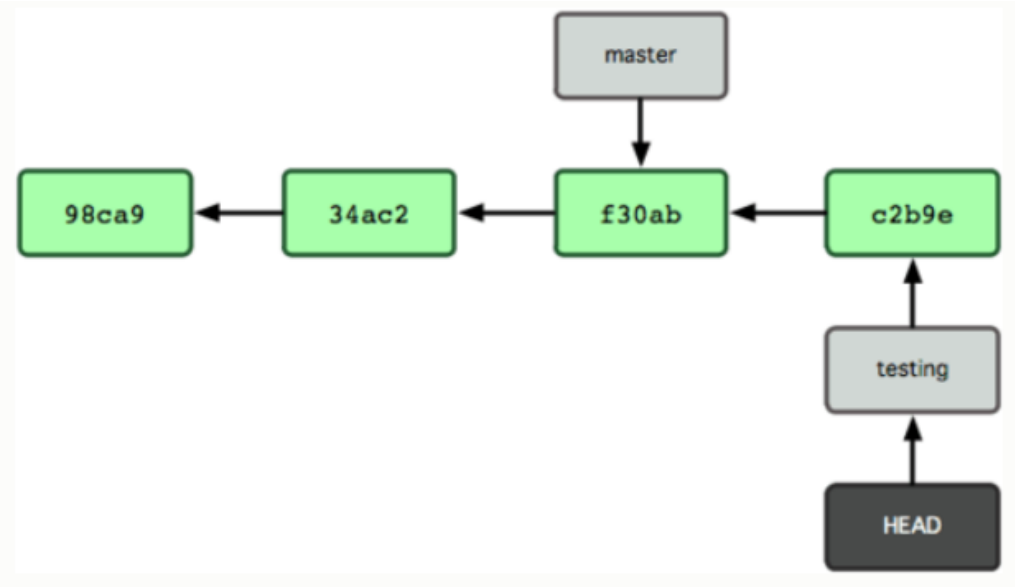
Esto mueve el apuntador HEAD a la rama `testing`



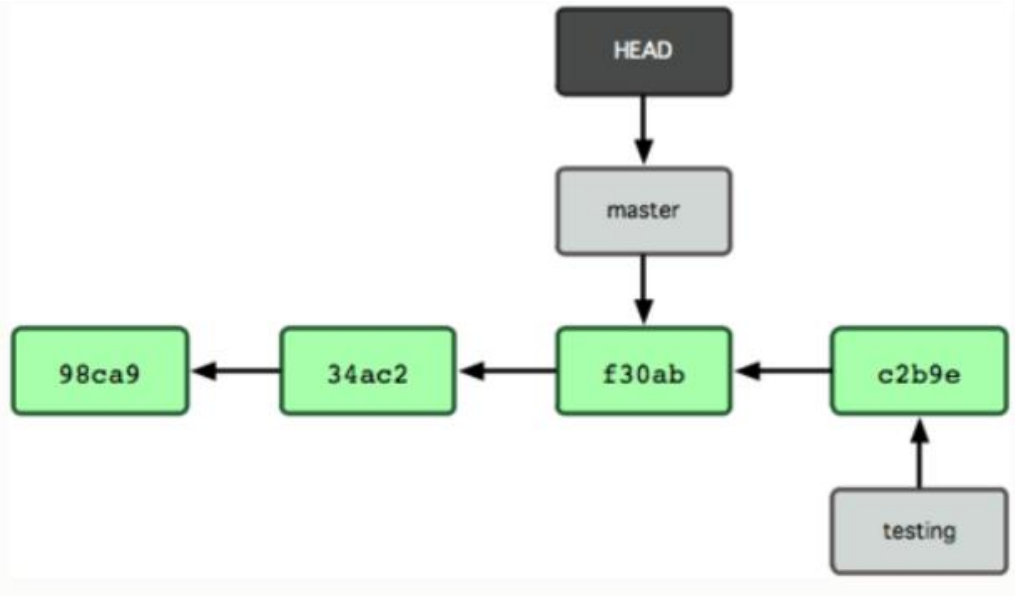
Si ahora editamos un archivo dentro de nuestro proyecto i realizamos un commit:

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

La rama `testing` avanza, mientras que la rama `master` permanece en la confirmación donde estaba cuando lanzaste el comando `git checkout` para saltar.



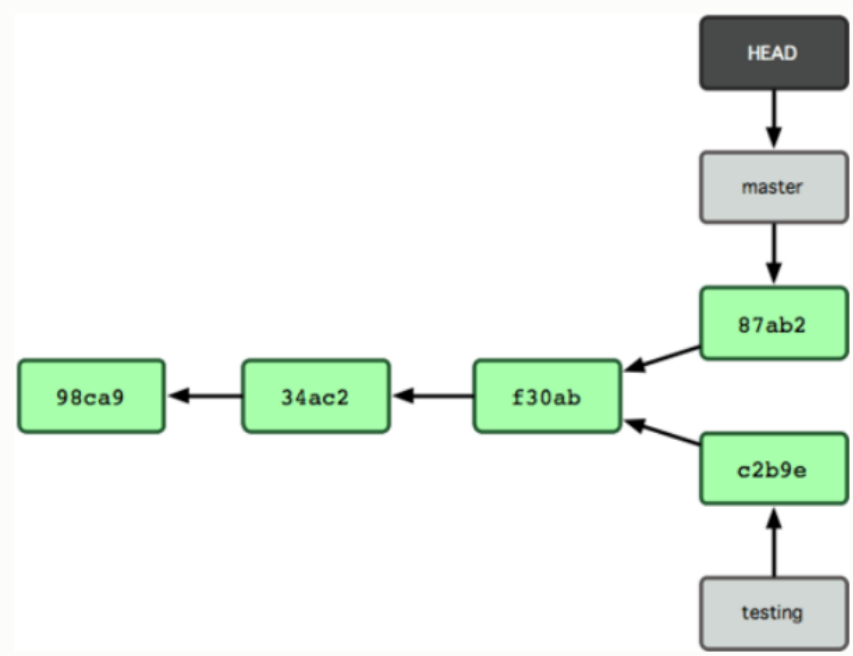
Si ahora volvemos a la rama Master:



Si ahora hacemos una nueva confirmación en la rama master:

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

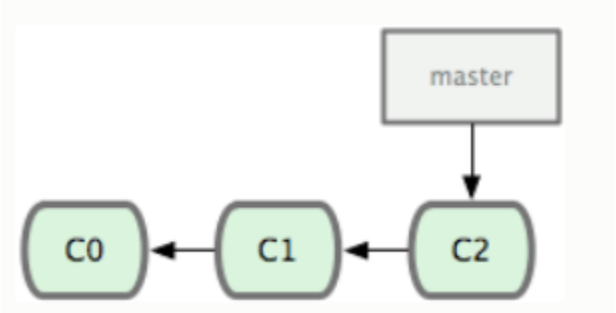
Ahora el registro de tu proyecto diverge:



Y todo ello simplemente con dos comandos: `git branch` y `git checkout`.

3.2 Ramificaciones en Git - Procedimientos básicos para ramificar y fusionar

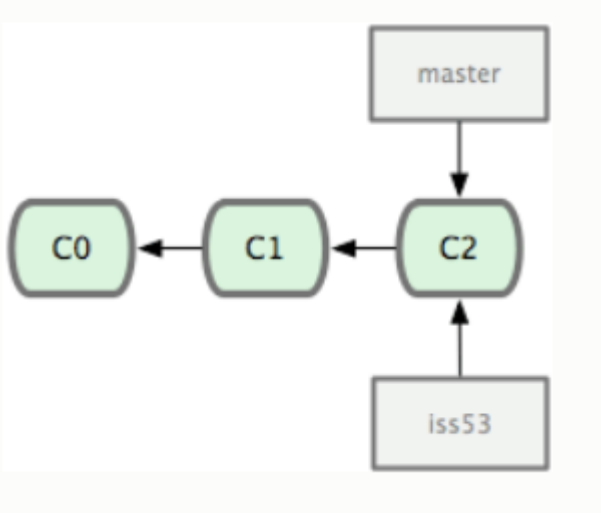
Imagina que estas trabajando en un proyecto, y tienes un par de confirmaciones (commit) ya realizadas:



Decides trabajar el problema #53, del sistema que tu compañía utiliza para llevar seguimiento de los problemas.

Como el problema #53 es un tema concreto y puntual en el que vas a trabajar, **creas una nueva rama para él**.

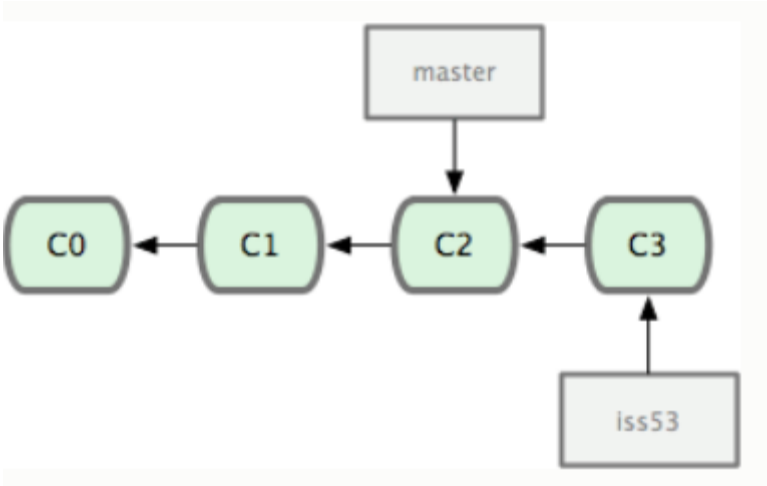
```
$ git checkout -b iss53
Switched to a new branch "iss53"
```



Creación de un apuntador a la nueva rama iss53

Trabajas en el sitio web y haces algunas confirmaciones de cambios (commits). **Con ello avanzas la rama `iss53`, que es la que tienes activada** (checked out) en este momento.

```
$ vim index.html
$ git commit -a -m 'added a new footer [issue 53]'
```



La rama `iss53` ha avanzado con tu trabajo.

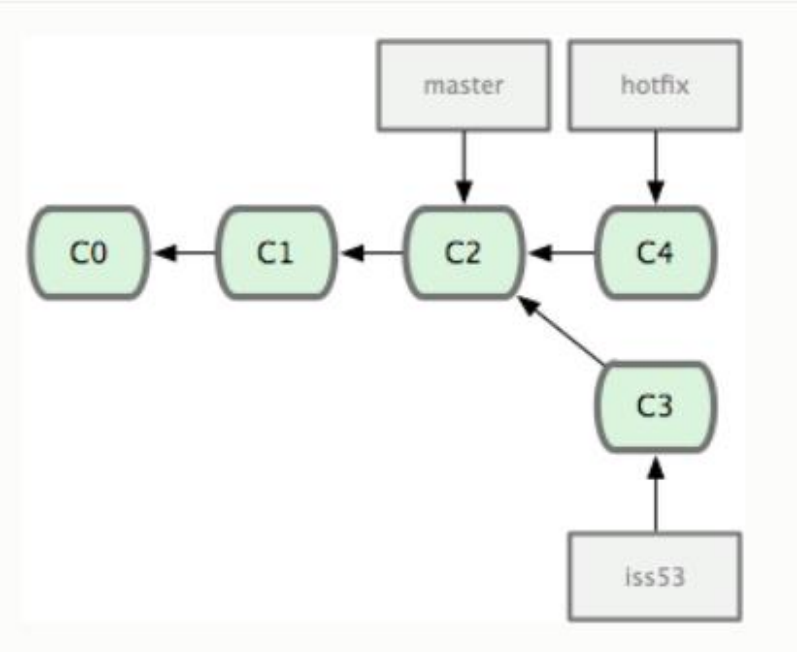
Entonces, recibes una llamada avisándote de otro problema urgente en el sitio web. Problema que has de resolver inmediatamente en la rama master o de producción:

Basta con saltar de nuevo a la rama `master` y continuar trabajando a partir de ella.

```
$ git checkout master
Switched to branch "master"
```

Volviendo al problema urgente. Vamos a crear una nueva rama `hotfix`, sobre la que trabajar hasta resolverlo:

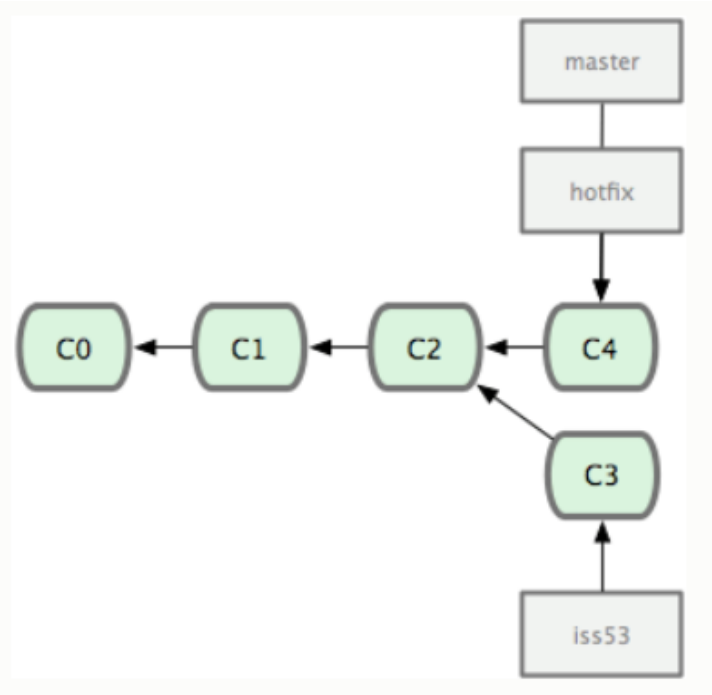
```
$ git checkout -b 'hotfix'
Switched to a new branch "hotfix"
$ vim index.html
$ git commit -a -m 'fixed the broken email address'
[hotfix]: created 3a0874c: "fixed the broken email address"
 1 files changed, 0 insertions(+), 1 deletions(-)
```



Puedes realizar las pruebas oportunas, asegurarte que la solución es correcta, e **incorporar los cambios a la rama `master` para ponerlos en producción**. Esto se hace con el comando `git merge`:

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast forward
 README |    1 -
 1 files changed, 0 insertions(+), 1 deletions(-)
```

Ahora, los cambios realizados están ya en la instantánea (snapshot) de la confirmación (commit) apuntada por la rama `master`. Y puedes desplegarlos

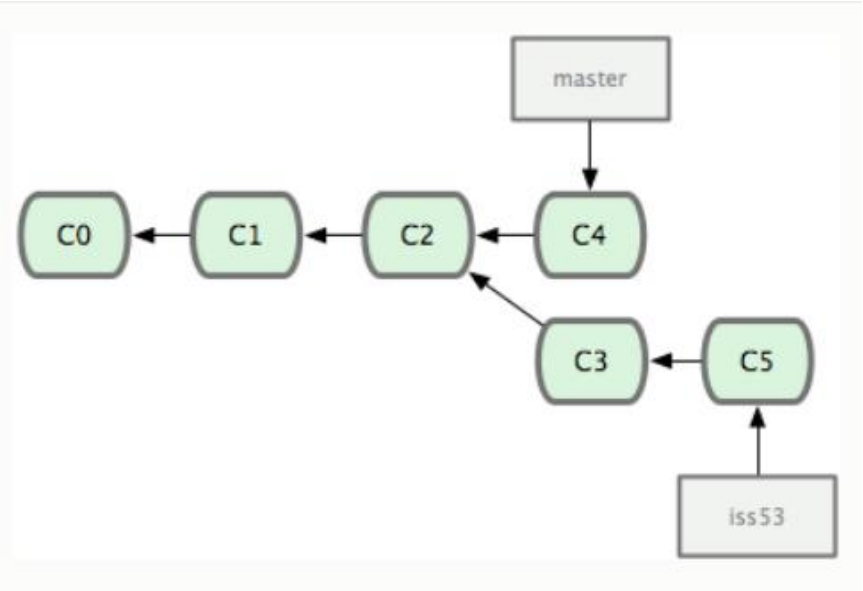


Pero antes, **es interesante borrar la rama `hotfix`. Ya que no la vamos a necesitar más**, puesto que apunta exactamente al mismo sitio que la rama `master`. Esto lo puedes hacer con la opción `-d` del comando `git branch`:

```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).
```

Y, con esto, ya estás dispuesto para regresar al trabajo sobre el problema #53

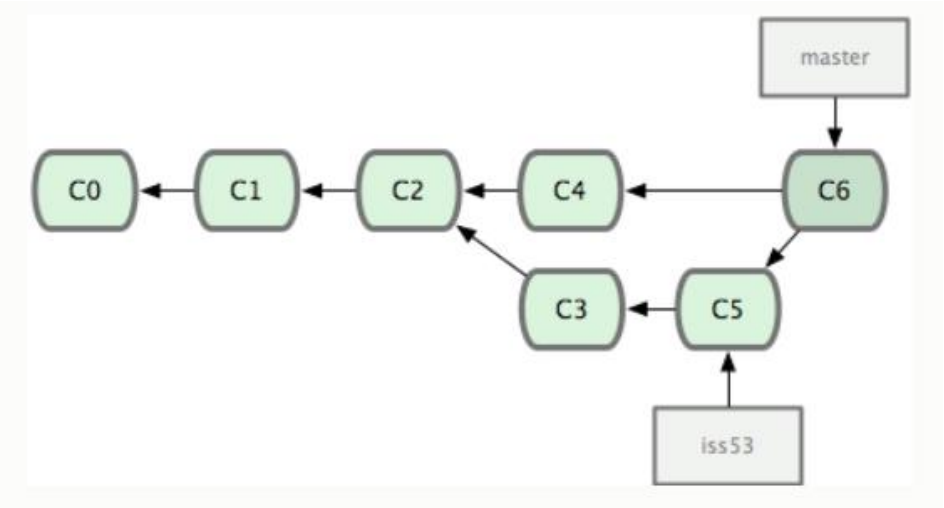
```
$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'finished the new footer [issue 53]'
[iss53]: created ad82d7a: "finished the new footer [issue 53]"
 1 files changed, 1 insertions(+), 0 deletions(-)
```



Supongamos que tu trabajo con el problema #53 está ya completo y listo para fusionarlo (merge) con la rama `master`. Entonces decidimos fusionar la rama `iss53`. Simplemente, activando (checkout) la rama donde deseas fusionar y lanzando el comando `git merge`:

```
$ git checkout master
$ git merge iss53
Merge made by recursive.
 README |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
```

En lugar de simplemente avanzar el apuntador de la rama, Git crea una nueva instantánea (snapshot) resultante de la fusión a tres bandas; y crea automáticamente una nueva confirmación de cambios (commit) que apunta a ella.



Ahora que todo tu trabajo está ya fusionado con la rama principal, ya no tienes necesidad de la rama `iss53`. Por lo que puedes borrarla:

```
$ git branch -d iss53
```

Principales conflictos que pueden surgir en las fusiones

Puedes obtener un conflicto de fusión si por ejemplo en la rama hotfix añadiste alguna linea dispar a la rama iss53:

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Con la orden git status nos indica el archivo confictivo, **con la palabra clave umerged**

```
[master*]$ git status
index.html: needs merge
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directo
#
#   unmerged:   index.html
#
```

Git añade a los archivos conflictivos unos marcadores especiales de resolución de conflictos. Marcadores que te guiarán cuando abras manualmente los archivos implicados y los edites para corregirlos:

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

3.3 Ramificaciones en Git - Gestión de ramificaciones

El comando `git branch` tiene más funciones que las de crear y borrar ramas. Si lo lanzas sin argumentos, obtienes una lista de las ramas presentes en tu proyecto:

```
$ git branch
  iss53
* master
  testing
```

`*` delante de la rama `master`: nos indica la rama activa en este momento.

Si hacemos una confirmación de cambios (commit), esa será la rama que avance.

Para ver la última confirmación de cambios en cada rama, puedes usar el comando `git branch -v`:

```
$ git branch -v
  iss53   93b412c fix javascript issue
* master  7a98805 Merge branch 'iss53'
  testing 782fd34 add scott to the author list in the
readmes
```

Si deseas ver las ramas que han sido fusionadas en la rama activa, puedes lanzar el comando `git branch --merged`

Aparece la rama `iss53` porque ya ha sido fusionada. Y no lleva por delante el caracter `*` porque todo su contenido ya ha sido incorporado a otras ramas. Podemos borrarla tranquilamente con `git branch -d`, sin miedo a perder nada.

```
$ git branch --merged
  iss53
* master
```

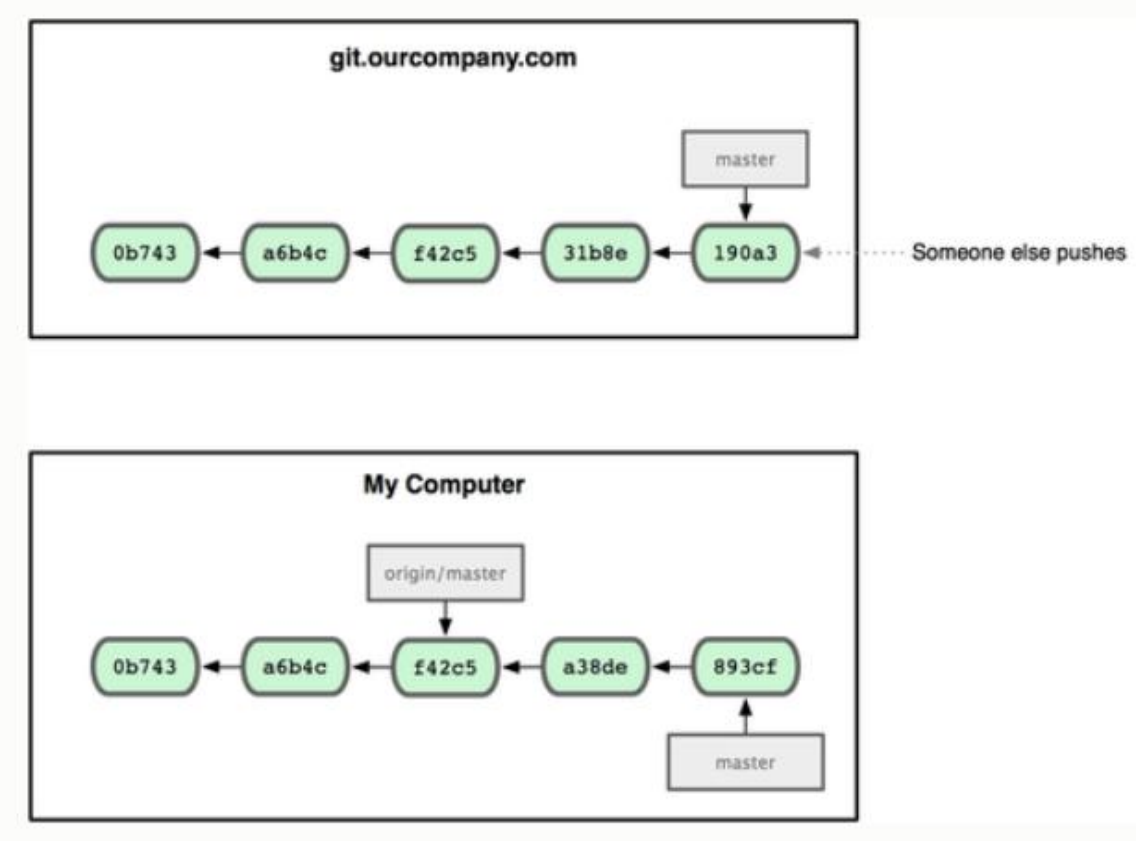
Para mostrar todas las ramas que contienen trabajos sin fusionar aún, puedes utilizar el comando:

```
$ git branch --no-merged
  testing
```

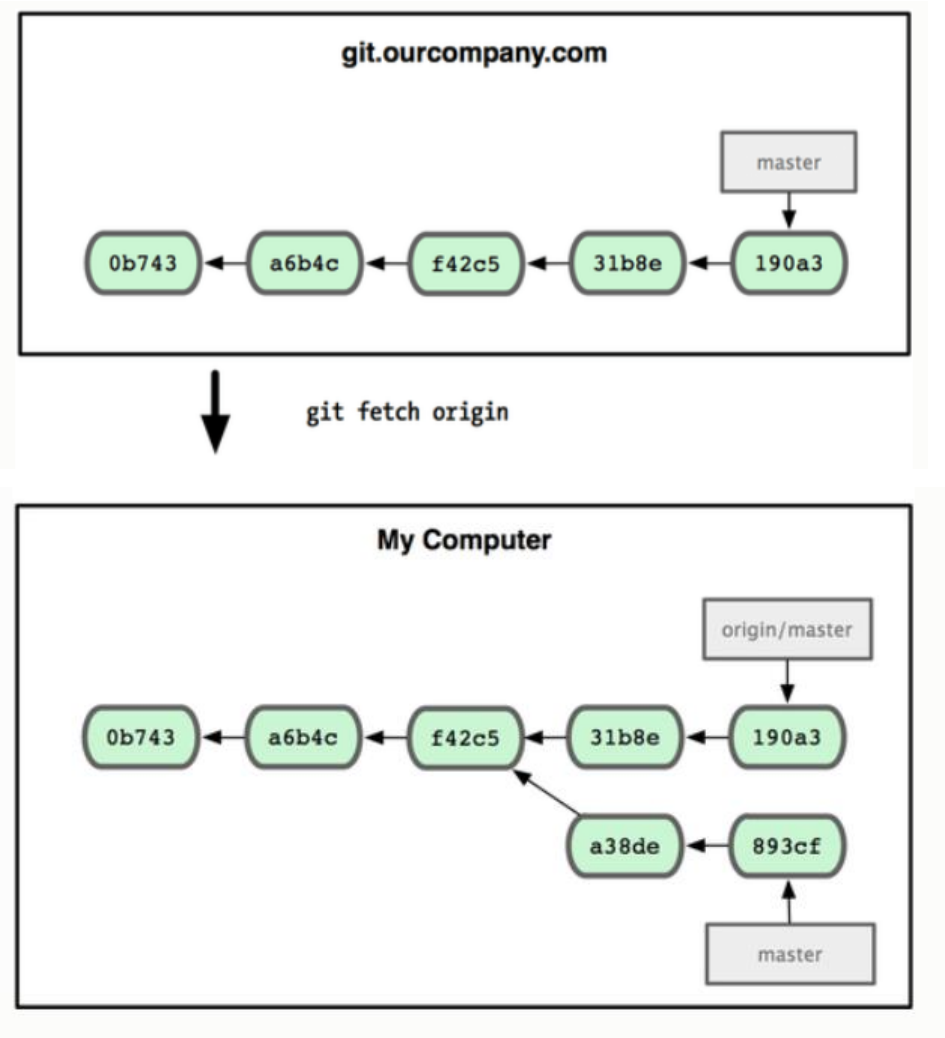
3.5 Ramificaciones en Git - Ramas Remotas

Supongamos que tienes un sevidor Git en tu red, en `git.ourcompany.com`. Si haces un clón desde ahí, el clon Git te proporciona tu propia rama `master` y otra rama `origin/master` apuntando a la rama `master` original:

Si haces algún trabajo en tu rama `master` local, y al mismo tiempo, alguna otra persona lleva (push) su trabajo al servidor `git.ourcompany.com`, actualizando la rama `master` de allí, te encontrarás con que ambos registros avanzan de forma diferente. Además, mientras no tengas contacto con el servidor, tu apuntador a tu rama `origin/master` no se moverá.



Para sincronizarte, puedes utilizar el comando `git fetch origin`. Este comando localiza en qué servidor está el origen (en este caso `git.ourcompany.com`), recupera cualquier dato presente allí que tu no tengas, y actualiza tu base de datos local, moviendo tu rama `origin/master` para que apunte a esta nueva y más reciente posición.



Publicando

Cuando quieres compartir una rama con el resto del mundo, has de llevarla (push) a un remoto donde tengas permisos de escritura. Tus ramas locales no se sincronizan automáticamente con los remotos en los que escribes.

Si tienes una rama llamada `serverfix`, con la que vas a trabajar en colaboración; puedes llevarla al remoto de la misma forma que llevaste tu primera rama. Con el comando `git push (remoto) (rama)`:

```
$ git push origin serverfix
Counting objects: 20, done.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (15/15), 1.74 KiB, done.
Total 15 (delta 5), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new branch]      serverfix -> serverfix
```

Esto viene a decir coge mi rama local `serverfix` y actualiza con ella la rama `serverfix` del remoto"

Es importante destacar que cuando recuperas (fetch) nuevas ramas remotas, no obtienes automáticamente una copia editable local de las mismas. En otras palabras, en este caso, no tienes una nueva rama `serverfix`

Para integrar (merge) esto en tu actual rama de trabajo, puedes usar el comando `git merge origin/serverfix`. Y si quieres tener tu propia rama `serverfix`, donde puedas trabajar, puedes crearla directamente basaádote en rama remota:

```
$ git checkout -b serverfix origin/serverfix
Branch serverfix set up to track remote branch refs/remotes/origin/serverf
Switched to a new branch "serverfix"Switched to a new branch "serverfix"
```

Esto sí te da una rama local donde puedes trabajar, comenzando donde `origin/serverfix` estaba en ese momento.