Pro Git, el libro oficial de Git

# 3.2. Procedimientos básicos para ramificar y fusionar

Vamos a presentar un ejemplo simple de ramificar y de fusionar, con un flujo de trabajo que se podría presentar en la realidad. Imagina que sigues los siquientes pasos:

1. Trabajas en un sitio web.
2. Creas una rama para un nuevo tema sobre el que quieres trabajar.
3. Realizas algo de trabajo en esa rama.

En este momento, recibes una llamada avisandote de un problema crítico que has de resolver. Y sigues los siguientes pasos:

1. Vuelves a la rama de producción original.
2. Creas una nueva rama para el problema crítico y lo resuelves trabajando en ella.
3. Tras las pertinentes pruebas, fusionas (*merge*) esa rama y la envias (*push*) a la rama de producción.
4. Vuelves a la rama del tema en que andabas antes de la llamada y continuas tu trabajo.

## 3.2.1. Procedimientos básicos de ramificación

Imagina que estas trabajando en un proyecto, y tienes un par de confirmaciones (*commit*) ya realizadas. (ver Figura 3-10)

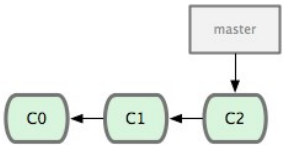



**Figura 3.10** Un registro de confirmaciones simple y corto

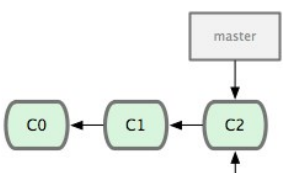
Decides trabajar el problema o error `#53` (en inglés, *issue #53*), del sistema que tu empresa utiliza para llevar seguimiento de los problemas. Aunque, por supuesto, Git no está ligado a ningún sistema de seguimiento de problemas concreto. Como el problema `#53` es un tema concreto y puntual en el que vas a trabajar, creas una nueva rama para él. Para crear una nueva rama y saltar a ella, en un solo paso, puedes utilizar el comando `git checkout` con la opción `-b`:

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

Esto es un atajo a:

```
$ git branch iss53
$ git checkout iss53
```

Figura 3-11 muestra el resultado.




## Indice de contenidos

1. Empezando
2. Fundamentos de Git
Capítulo 3. Trabajando con ramas en Git
3.1. ¿Qué es una rama?
**3.2. Procedimientos básicos para ramificar y fusionar**
3.3. Gestión de ramificaciones
3.4. Flujos de trabajo ramificados
3.5. Ramas Remotas
3.6. Reorganizando el trabajo realizado
3.7. Resumen
4. Git en un servidor
5. Git en entornos distribuidos
6. Las herramientas de Git
7. Personalizando Git
8. Git y otros sistemas
9. Funcionamiento interno de Git


**Figura 3.11** Creación de un apuntador a la nueva rama

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 (es decir, a la que apunta `HEAD`; ver Figura 3-12):

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

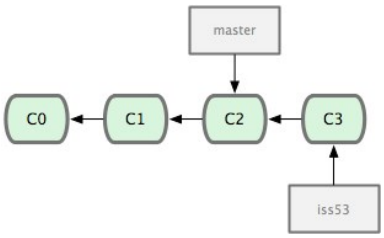



**Figura 3.12** La rama 'iss53' ha avanzado con tu trabajo

Entonces, recibes una llamada avisándote de otro problema urgente en el sitio web. Problema que has de resolver inmediatamente. Usando Git, no necesitas mezclar el nuevo problema con los cambios que ya habias realizado sobre el problema `#53`; ni tampoco perder tiempo revirtiendo esos cambios para poder trabajar sobre el contenido que está en producción. Basta con saltar de nuevo a la rama `master` y continuar trabajando a partir de ella.

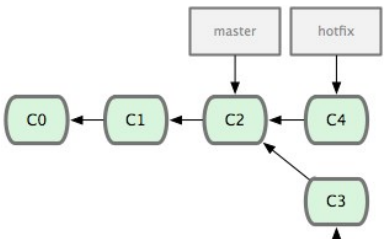
Pero, antes de poder hacer eso, hemos de tener en cuenta que teniendo cambios aún no confirmados en la carpeta de trabajo o en el área de preparación, Git no nos permitirá saltar a otra rama con la que podríamos tener conflictos. Lo mejor es tener siempre un estado de trabajo limpio y despejado antes de saltar entre ramas. Y, para ello, tenemos algunos procedimientos (*stash* y *commit ammend*), que vamos a ver más adelante. Por ahora, como tenemos confirmados todos los cambios, podemos saltar a la rama `master` sin problemas:

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

Tras esto, tendrás la carpeta de trabajo exactamente igual a como estaba antes de comenzar a trabajar sobre el problema `#53`. Y podrás concentrarte en el nuevo problema urgente. Es importante recordar que Git revierte la carpeta de trabajo exactamente al estado en que estaba en la confirmación (*commit*) apuntada por la rama que activamos (*checkout*) en cada momento. Git añade, quita y modifica archivos automáticamente. Para asegurarte que tu copia de trabajo es exactamente tal y como era la rama en la última confirmación de cambios realizada sobre ella.

Volviendo al problema urgente. Vamos a crear una nueva rama `hotfix`, sobre la que trabajar hasta resolverlo (ver Figura 3-13):

```
$ 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(-)
```



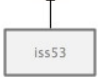



**Figura 3.13** Rama 'hotfix' basada en la rama `master` original

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(-)
```

Merece destacar la frase "*Avance rápido*" ("*Fast forward*") que aparece en la respuesta al comando. Git ha movido el apuntador hacia adelante, ya que la confirmación apuntada en la rama donde has fusionado estaba directamente por delante respecto de la confirmación actual. Dicho de otro modo: cuando intentas fusionar una confirmación con otra confirmación accesible siguiendo directamente el registro de la primera; Git simplifica las cosas avanzando el puntero, ya que no hay ningún otro trabajo divergente a fusionar. Esto es lo que se denomina "*avance rápido*" ("*fast forward*").

Ahora, los cambios realizados están ya en la instantánea (*snapshot*) de la confirmación (*commit*) apuntada por la rama `master`. Y puedes desplegarlos (ver Figura 3-14)

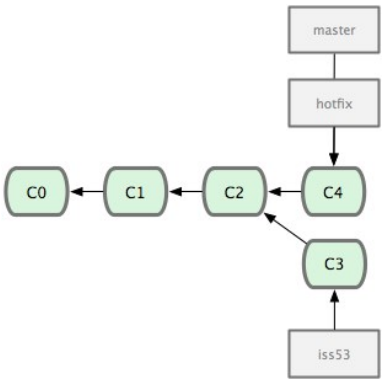



**Figura 3.14** Tras la fusión (*merge*), la rama `master` apunta al mismo sitio que la rama `hotfix`

Tras haber resuelto el problema urgente que te habia interrumpido tu trabajo, puedes volver a donde estabas. 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 estas dispuesto para regresar al trabajo sobre el problema `#53` (ver Figura 3-15):

```
$ 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(-)
```

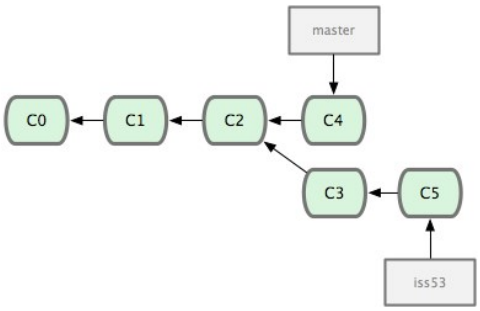



**Figura 3.15** La rama 'iss53' puede avanzar independientemente

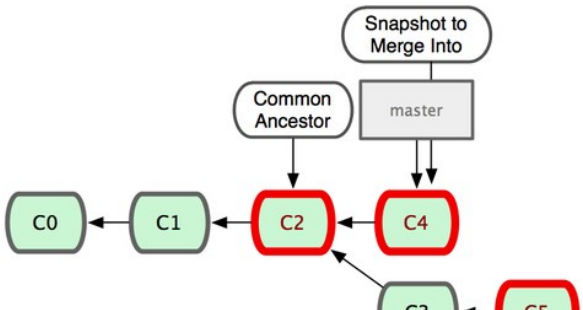
Cabe indicar que todo el trabajo realizado en la rama `hotfix` no está en los archivos de la rama `iss53`. Si fuera necesario agregarlos, puedes fusionar (*merge*) la rama `master` sobre la rama `iss53` utilizando el comando `git merge master`. O puedes esperar hasta que decidas llevar (*pull*) la rama `iss53` a la rama `master`.

## 3.2.2. Procedimientos básicos de fusión

Supongamos que tu trabajo con el problema `#53` está ya completo y listo para fusionarlo (*merge*) con la rama `master`. Para ello, de forma similar a como antes has hecho con la rama `hotfix`, vas a 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(-)
```

Es algo diferente de la fusión realizada anteriormente con `hotfix`. En este caso, el registro de desarrollo habia divergido en un punto anterior. Debido a que la confirmación en la rama actual no es ancestro directo de la rama que pretendes fusionar, Git tiene cierto trabajo extra que hacer. Git realizará una fusión a tres bandas, utilizando las dos instantáneas apuntadas por el extremo de cada una de las ramas y por el ancestro común a ambas dos. La figura 3-16 ilustra las tres instantáneas que Git utiliza para realizar la fusión en este caso.



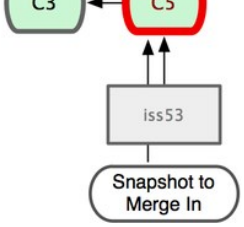



**Figura 3.16** Git identifica automáticamente el mejor ancestro común para realizar la fusión de las ramas

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. Nos referimos a este proceso como "*fusión confirmada*". Y se diferencia en que tiene más de un padre.

Merece la pena destacar el hecho de que es el propio Git quien determina automáticamente el mejor ancestro común para realizar la fusión. Diferenciandose de otros sistemas tales como CVS o Subversion, donde es el desarrollador quien ha de imaginarse cuál puede ser dicho mejor ancestro común. Esto hace que en Git sea mucho más facil el realizar fusiones.

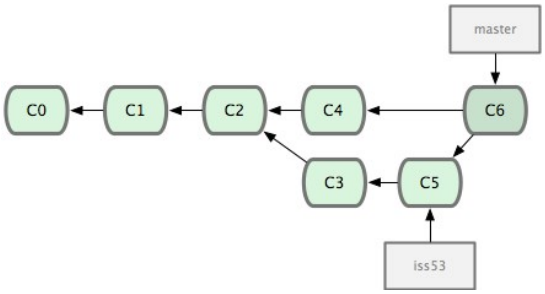



**Figura 3.17** Git crea automáticamente una nueva confirmación para la fusión

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. Y cerrar manualmente el problema en el sistema de seguimiento de problemas de tu empresa.

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

## 3.2.3. Principales conflictos que pueden surgir en las fusiones

En algunas ocasiones, los procesos de fusión no suelen ser fluidos. Si hay modificaciones dispares en una misma porción de un mismo archivo en las dos ramas distintas que pretendes fusionar, Git no será capaz de fusionarlas directamente. Por ejemplo, si en tu trabajo del problema `#53` has modificado una misma porción que también ha sido modificada en el problema `hotfix`. Puedes obtener un conflicto de fusión tal que:

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

Git no crea automáticamente una nueva fusión confirmada (*merge commit*). Sino que hace una pausa en el proceso, esperando a que tu resuelvas el conflicto. Para ver qué archivos permanecen sin fusionar en un determinado momento conflictivo de una fusión, puedes usar el comando `git status`:

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

```
#
```

Todo aquello que sea conflictivo y no se haya podido resolver, se marca como "*sin fusionar*" (*unmerged*). Git añade a los archivos conflictivos unas marcas especiales de resolución de conflictos. Estas marcas o marcadores te ayudan cuando abres los archivos implicados y los editas manualmente para corregirlos. El archivo conflictivo contendrá algo como:

```
<<<<<<< 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
```

Donde nos dice que la versión en `HEAD` (la rama `master`, la que habias activado antes de lanzar el comando de fusión), contiene lo indicado en la parte superior del bloque (todo lo que está encima de `=======`). Y que la versión en `iss53` contiene el resto, lo indicado en la parte inferior del bloque. Para resolver el conflicto, has de elegir manualmente contenido de uno o de otro lado. Por ejemplo, puedes optar por cambiar el bloque, dejandolo tal que:

```
<div id="footer">
please contact us at email.support@github.com
</div>
```

Esta corrección contiene un poco de ambas partes. Y se han eliminado completamente las líneas `<<<<<<<` , `=======` y `>>>>>>>` Tras resolver todos los bloques conflictivos, has de lanzar comandos `git add` para marcar cada archivo modificado. Marcar archivos como preparados (*staging*), indica a Git que sus conflictos han sido resueltos.

Si, en lugar de resolver directamente, prefieres utilizar una herramienta gráfica. Puedes usar el comando `git mergetool`. Esto arrancará la correspondiente herramienta de visualización y te permirá ir resolviendo conflictos con ella.

```
$ git mergetool
merge tool candidates: kdiff3 tkdiff xxdiff meld gvimdiff opendiff emerge vimdiff
Merging the files: index.html


Normal merge conflict for 'index.html':
  {local}: modified
  {remote}: modified
Hit return to start merge resolution tool (opendiff):
```

Si deseas usar una herramienta distinta de la escogida por defecto (en mi caso `opendiff`, porque estoy lanzando el comando en un Mac), puedes escogerla entre la lista de herramientas soportadas mostradas al principio ("*merge tool candidates*"). Tecleando el nombre de dicha herramienta. En el capítulo 7 se verá cómo cambiar este valor por defecto de tu entorno de trabajo.

Tras salir de la herramienta de fusionado, Git preguntará a ver si hemos resuelto todos los conflictos y la fusión ha sido satisfactoria. Si le indicas que así ha sido, Git marca como preparado (*staged*) el archivo que acabamos de modificar.

En cualquier momento, puedes lanzar el comando `git status` para ver si ya has resuelto todos los conflictos:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   index.html
```

```
#
```

Si todo ha ido correctamente, y ves que todos los archivos conflictivos están marcados como preparados, puedes lanzar el comando `git commit` para terminar de confirmar la fusión. El mensaje de confirmación por defecto será algo parecido a:

```
Merge branch 'iss53'


Conflicts:
  index.html
#
# It looks like you may be committing a MERGE.
# If this is not correct, please remove the file
# .git/MERGE_HEAD
# and try again.
#
```

Puedes modificar este mensaje añadiendo detalles sobre cómo has resuelto la fusión, si lo consideras útil para que otros entiendan esta fusión en un futuro. Se trata de indicar porqué has hecho lo que has hecho; a no ser que resulte obvio, claro está.


Anterior
3.1. ¿Qué es una rama?

Siguiente
3.3. Gestión de ramificaciones