
This book is available in English.

Full translation available in

azərbaycan dili,
български език,
Deutsch,
Español,
Français,
Ελληνικά,
日本語,
한국어,
Nederlands,
Русский,
Slovenščina,
Tagalog,
Українська
简体中文,

Partial translations available in

Čeština,
Македонски,
Polski,
Српски,
Ўзбекча,
繁體中文,

Translations started for

Беларуская,
فارسی,
Indonesian,
Italiano,
Bahasa Melayu,
Português (Brasil),
Português (Portugal),
Svenska,
Türkçe.

The source of this book is hosted on GitHub.
Patches, suggestions and comments are welcome.

2nd Edition

3.2 Ramificaciones en Git - Procedimientos Básicos para Ramificar y Fusionar

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 siguientes 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 avisándote 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 envías (push) a la rama de producción.
4. Vuelves a la rama del tema en que andabas antes de la llamada y continúas tu trabajo.

Procedimientos Básicos de Ramificación

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

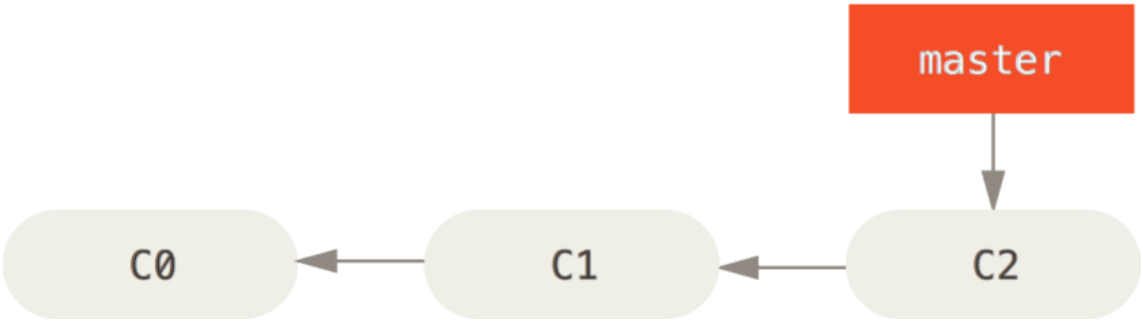

Figura 18. Un registro de confirmaciones corto y sencillo

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

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

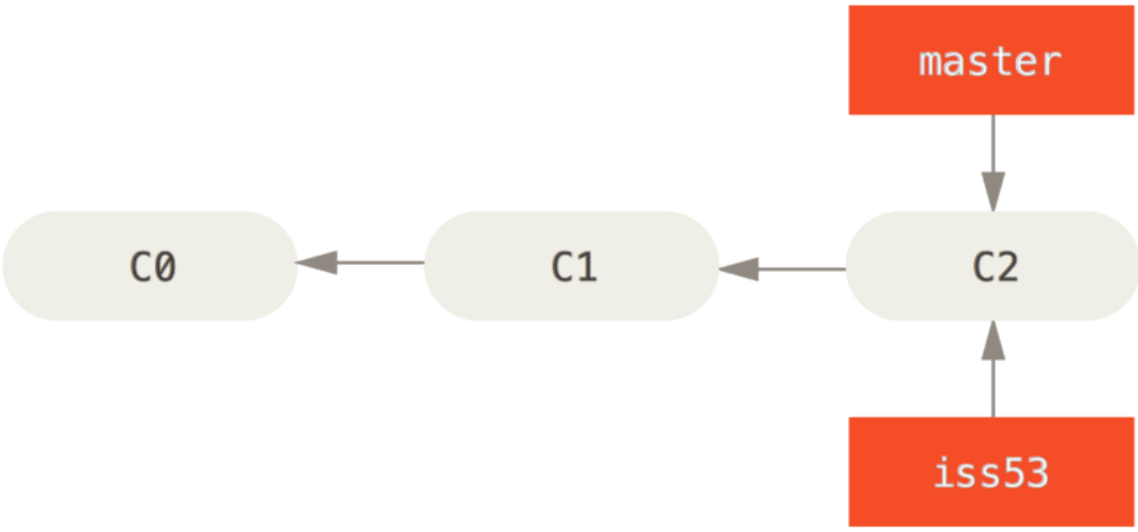

Figura 19. Crear un apuntador a la rama nueva

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):

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

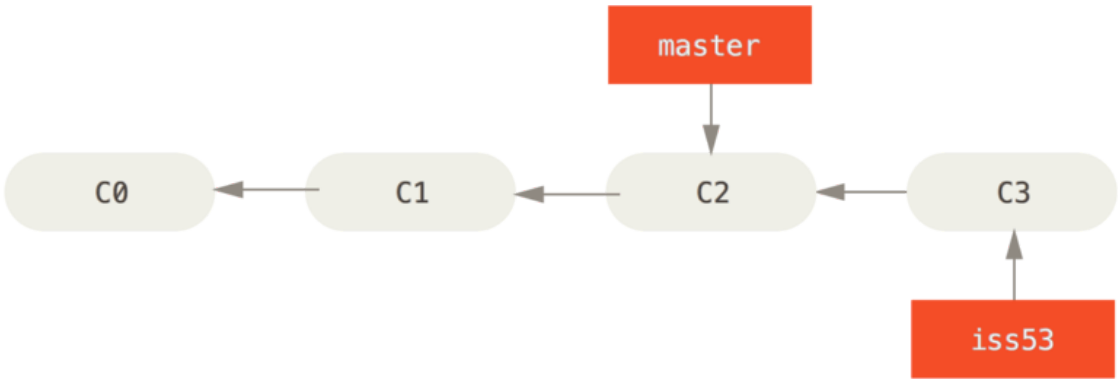

Figura 20. La rama `iss53` ha avanzado con tu trabajo

Entonces, recibes una llamada avisándote de otro problema urgente en el sitio web y debes resolverlo inmediatamente. Al usar Git, no necesitas mezclar el nuevo problema con los cambios que ya habías 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 allí.

Pero, antes de poder hacer eso, hemos de tomar en cuenta que si tenemos cambios aún no confirmados en el directorio 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 corregir confirmaciones), que vamos a ver más adelante en Guardado rápido y Limpieza. 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 el directorio 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 el directorio 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 asegurar que tu copia de trabajo luce exactamente como lucía la rama en la última confirmación de cambios realizada sobre ella.

A continuación, es momento de resolver el 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 1fb7853] fixed the broken email address
 1 file changed, 2 insertions(+)
```

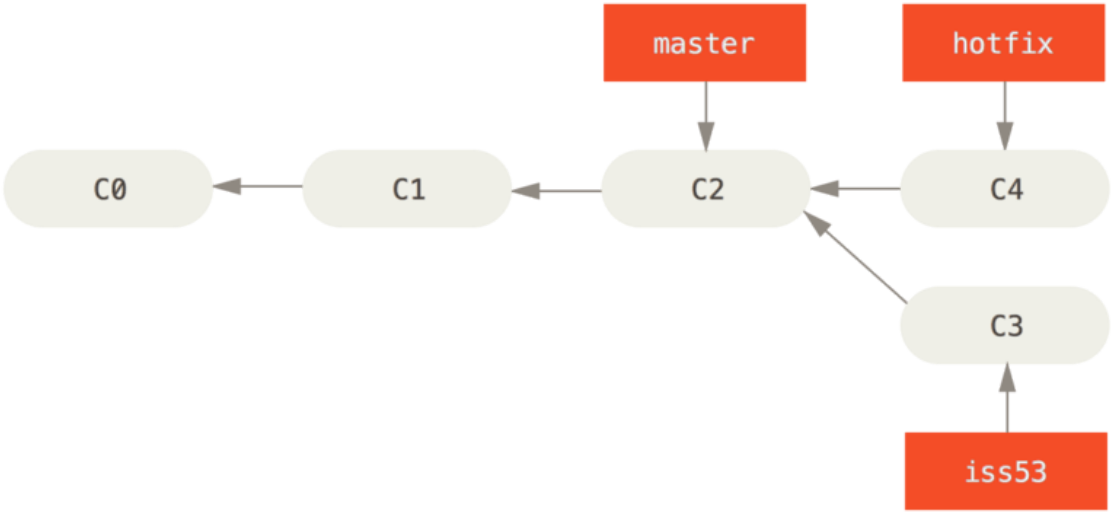

Figura 21. Rama `hotfix` basada en la rama `master` original

Puedes realizar las pruebas oportunas, asegurarte de 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
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```

Notarás la frase “Fast forward” (“Avance rápido”, en inglés) que aparece en la salida del comando. Git ha movido el apuntador hacia adelante, ya que la confirmación apuntada en la rama donde has fusionado estaba directamente arriba respecto a la confirmación actual. Dicho de otro modo: cuando intentas fusionar una confirmación con otra confirmación accesible siguiendo directamente el historial 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.

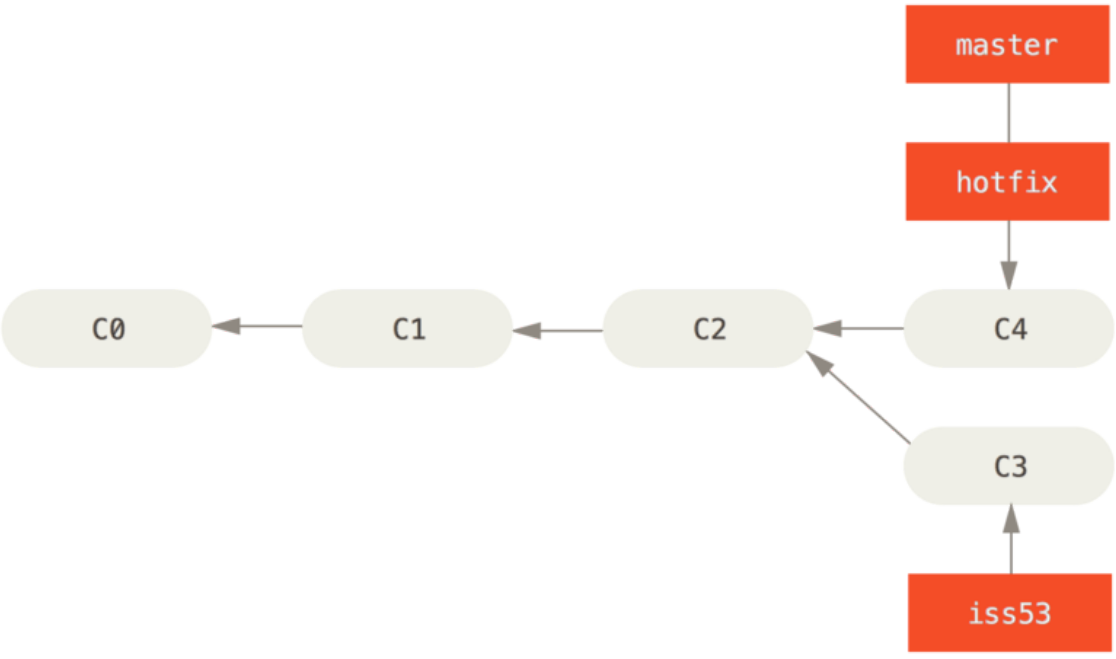

Figura 22. Tras la fusión (merge), la rama `master` apunta al mismo sitio que la rama `hotfix`.

Tras haber resuelto el problema urgente que había interrumpido tu trabajo, puedes volver a donde estabas. Pero antes, es importante 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 listo 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 ad82d7a] finished the new footer [issue 53]
1 file changed, 1 insertion(+)
```

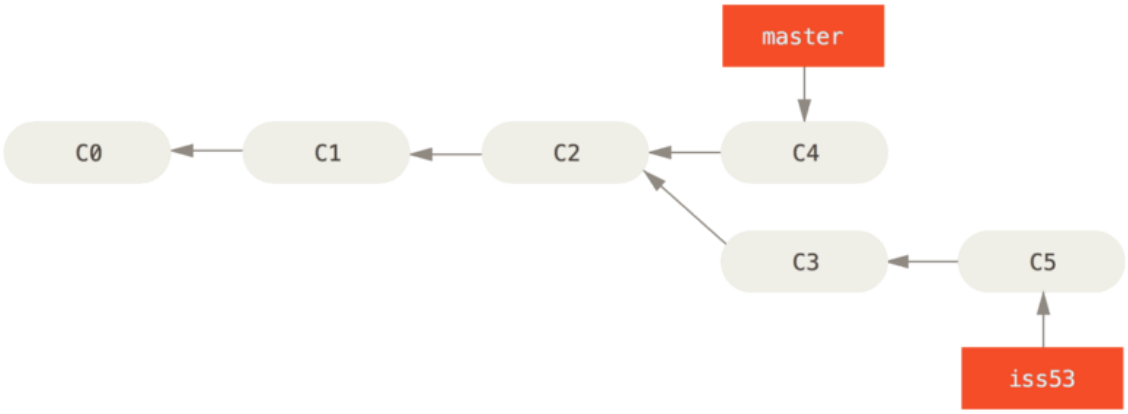

Figura 23. La rama `iss53` puede avanzar independientemente

Cabe destacar 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 fusionar (merge) la rama `iss53` a la rama `master`.

Procedimientos Básicos de Fusión

Supongamos que tu trabajo con el problema #53 ya está 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, activa (checkout) la rama donde deseas fusionar y lanza el comando `git merge`:

```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html |    1 +
1 file changed, 1 insertion(+)
```

Es algo diferente de la fusión realizada anteriormente con `hotfix`. En este caso, el registro de desarrollo había 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.

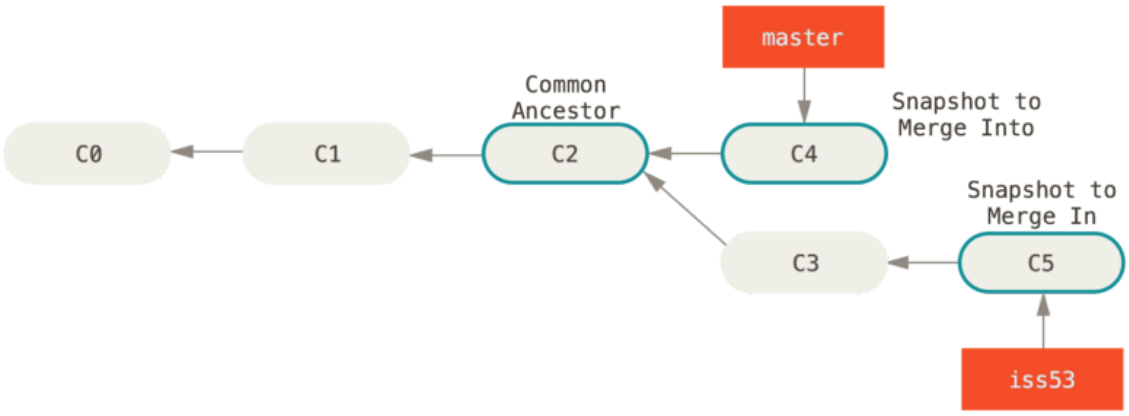

Figura 24. Git identifica automáticamente el mejor ancestro común para realizar la fusión de las ramas

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 su particularidad es que tiene más de un padre.

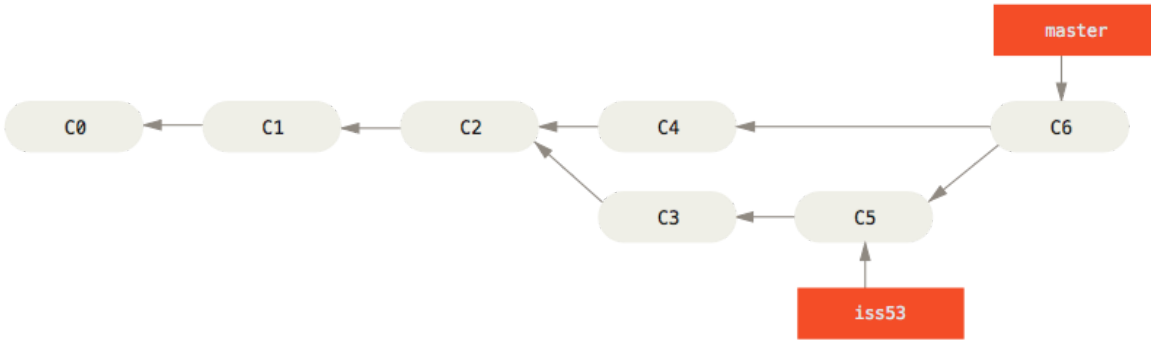

Figura 25. Git crea automáticamente una nueva confirmación para la fusión

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

Ahora que todo tu trabajo ya está fusionado con la rama principal, 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
```

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`, verás un conflicto como este:

```
$ 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 tú resuelvas el conflicto. Para ver qué archivos permanecen sin fusionar en un determinado momento conflictivo de una fusión, puedes usar el comando `git status`:

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Todo aquello que sea conflictivo y no se haya podido resolver, se marca como "sin fusionar" (unmerged). Git añade a los archivos conflictivos unos marcadores especiales de resolución de conflictos que te guiarán cuando abras manualmente los archivos implicados y los edites 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 habías 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 el contenido de uno o de otro lado. Por ejemplo, puedes optar por cambiar el bloque, dejándolo así:

```
<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 (staged) 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`, el cual arrancará la correspondiente herramienta de visualización y te permitirá ir resolviendo conflictos con ella:

```
$ git mergetool

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecmerge p4merge araxis bc3 codecompare vimdiff emerge
Merging:
index.html

Normal merge conflict for 'index.html':
  {local}: modified file
  {remote}: modified file
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 Mac), puedes escogerla entre la lista de herramientas soportadas mostradas al principio ("merge tool candidates") tecleando el nombre de dicha herramienta.

Nota Si necesitas herramientas más avanzadas para resolver conflictos de fusión más complicados, revisa la sección de fusionado en Fusión Avanzada.

Tras salir de la herramienta de fusionado, Git preguntará 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
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

    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.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       modified:   index.html
#
```

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 por qué has hecho lo que has hecho; a no ser que resulte obvio, claro está.

prev | next
About this site
Patches, suggestions, and comments are welcome.
Git is a member of Software Freedom Conservancy