
ANALYZE
DEVELOPMENT
DESIGN
IMPLEMENT

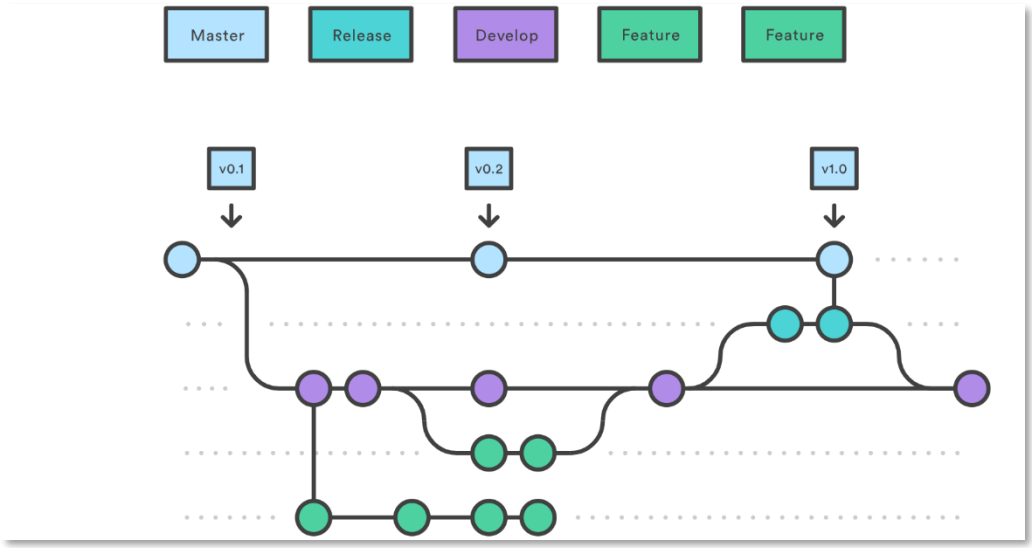
CASO PRÁCTICO

En *GoyoProg* están utilizando la herramienta GIT, con GitHub y la metodología *GitFlow*. A lo largo del desarrollo ha surgido la necesidad de que varios desarrolladores del equipo de programadores realicen cambios sobre el mismo código de forma simultánea, por lo que deberán crear ramas (*branches*) de forma que, una vez terminado cada desarrollo, pueda fusionarse (*merge*) dentro de una misma rama principal(Ej. *main*).

Tema 4. RAMIFICACIÓN EN EL CONTROL DE VERSIONES GIT

6.1 Uso de Ramas

Metodología ***GitFlow***



Existen dos distinciones principales en las ramas que define la metodología *GitFlow*:

- **Las ramas principales**: rama *Main* y la rama *Develop*.
- **Las ramas auxiliares**: rama *Feature*, rama *Release* y rama *Hotfix*.

Es un riesgo 'hacer commits' de código directamente en la rama *Main*, porque contiene el código actualmente desplegado en entorno productivo y puede generar defectos en el repositorio en las siguientes subidas a producción, por lo que siempre **es recomendable integrar el código nuevo en otras ramas** antes de fusionarlo con las ramas **Main y Develop,** para minimizar los riesgos.

Sobre las ramas auxiliares:

- ✓ **La rama *Feature*,** para nuevos requisitos o nuevas historias de usuario.
- ✓ **La rama *Release*,** para estandarizar una serie de código que ha estado desarrollándose en la rama *Develop*, se saca una rama de este tipo, se somete a una serie de pruebas y tests por parte del equipo de QA; y una vez superados se desplegará en entornos productivos.
- ✓ **La rama *Hotfix***, habitualmente se utiliza para depurar el código que viene de producción, por haberse detectado un defecto crítico en producción que deba resolverse mediante un parche o hotfix.

6.2 Comandos

Recientemente GitHub remplazo a master como rama principal y hay “problemas” ya que en git estás trabajando con master pero en GitHub “main” ahora es la rama principal.

Hay dos maneras de arreglar esto:

- Regresar a master como rama principal en GitHub
- Usar a main como rama principal en git (Recomendable)

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~ (master)
$ git branch -m master main

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~ (main)
$ git push -u origin main
```

Ver ramas creadas (locales y remotas)

```
$ git branch -a
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/clases/entornos (main)
$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/develop
  remotes/origin/feature/cambiosHTML
  remotes/origin/main
```

Crear nueva rama

```
$ git branch <nombre de la rama>
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/clases/entornos (main)
$ git branch nuevarama
```

Eliminar una rama

```
$ git branch -d <nombre de la rama>
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/clases/entornos (main)
$ git branch -d nuevarama
Deleted branch nuevarama (was b9e2c02).
```

Eliminar una rama en remoto

```
$ git push origin --delete <nombre de la rama>
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/clases/entornos (main)
$ git push origin --delete nuevarama
To https://github.com/ruth-svg/entornos.git
 - [deleted]         nuevarama
```

Cambiar a otra rama

```
$ git checkout <nombre de la rama>
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/clases/entornos (main)
$ git checkout develop
Switched to a new branch 'develop'
Branch 'develop' set up to track remote branch 'develop' from 'origin'.
```

Establecer enlace entre rama local y rama remota

```
$ git push --set-upstream origin <nombreRamaLocal>
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/clases/entornos (main)
$ git push --set-upstream origin nuevarama
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'nuevarama' on GitHub by visiting:
remote:      https://github.com/ruth-svg/entornos/pull/new/nuevarama
remote:
remote: Heads up! The branch 'nuevarama' that you pushed to was renamed to 'deve
lop'.
remote:
To https://github.com/ruth-svg/entornos.git
 * [new branch]      nuevarama -> nuevarama
Branch 'nuevarama' set up to track remote branch 'nuevarama' from 'origin'.
```

Mergear ramas

```
$ git merge <nombre de la ramificación>
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/clases/entornos (main)
$ git merge nuevarama
Merge made by the 'recursive' strategy.
 ruth.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 ruth.txt
```

El comando git merge permite tomar las líneas independientes de desarrollo creadas por git branch e integrarlas en una sola rama. La rama actual se actualizará para reflejar la fusión, pero la rama de destino no se verá afectada en absoluto. Git merge se suele utilizar junto con git checkout para seleccionar la rama actual y git branch -d para eliminar la rama de destino obsoleta.

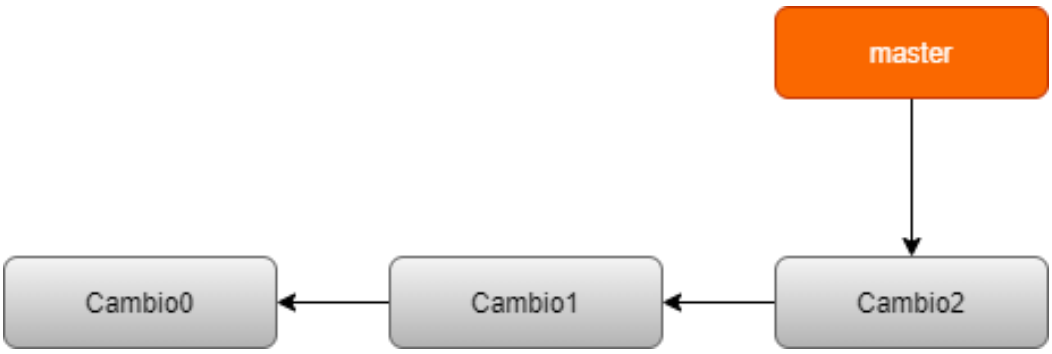
6.2.1 Procedimientos Básicos para Ramificar y Fusionar

Vamos a presentar un ejemplo simple de ramificar y de fusionar, con un flujo de trabajo que se produce en un proyecto software habitualmente:

1. Trabajamos en el desarrollo de una aplicación software.
2. Creamos una rama para un nuevo desarrollo sobre el que queremos trabajar.
3. Realizamos algunos cambios sobre esa rama.

En un determinado momento, mientras desarrollamos ese cambio, recibimos una llamada avisándonos de un problema crítico en entorno productivo que debe ser resuelto lo antes posible. En ese momento, seguimos los siguientes pasos:

1. Volvemos a la rama de producción original (*main*).
2. Creamos una nueva rama para el problema crítico y lo resolvemos trabajando en ella.
3. Tras las pertinentes pruebas, fusionamos (*merge*) esa rama y la envíamos (*push*) a la rama de producción(*main*).
4. Volvemos a la rama del desarrollo en el que trabajábamos antes de la llamada y continuamos nuestro trabajo.



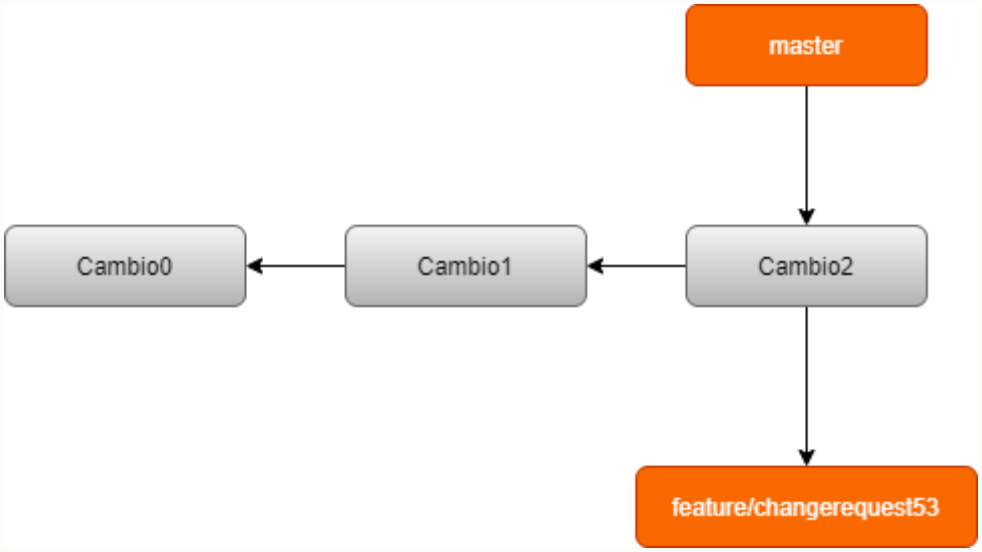
Decidimos trabajar en el desarrollo identificado con el identificador ***changerequest53***, según el sistema que nuestra compañía utiliza para llevar el seguimiento de los cambios en el desarrollo. Para crear una nueva rama y saltar a ella, en un solo paso, podemos utilizar el comando '*git checkout*' con la opción -b:

```
$ git checkout -b feature/changerequest53

Switched to a new branch "feature/changerequest53"
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (main)
$ git checkout -b feature/changerequest53
Switched to a new branch 'feature/changerequest53'
```

Esto es un atajo para estos dos comandos *Git*:

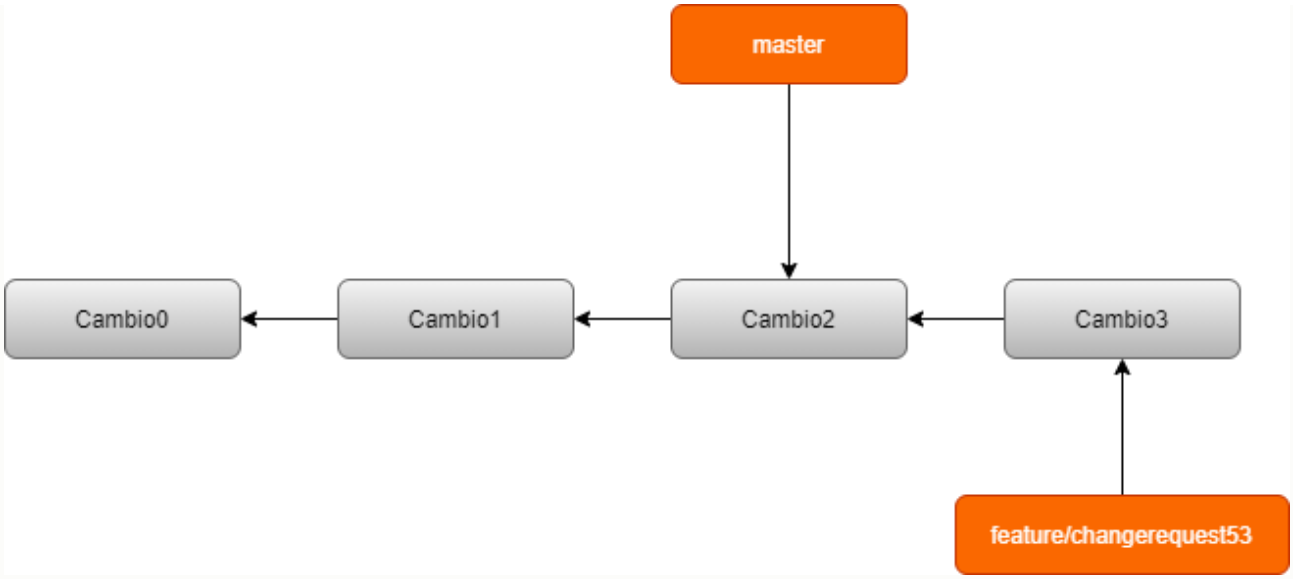
```
$ git branch feature/changerequest53

$ git checkout feature/changerequest53
```



Realizamos cambios sobre el sitio web y hacemos algunas confirmaciones de cambios (*commits*) para dar solución al desarrollo solicitado. Con ello avanzamos la rama `feature/changerequest53`, que es la que tenemos activada en este momento (es decir, a la que apunta HEAD):

```
$ touch index.html

$ git add .

$ git commit -a -m 'added new error control [changerequest53]'
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (feature/changerequest53
)
$ git add .
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (feature/changerequest53
)
$ git commit -a -m "add new error control [changerequest53"
[feature/changerequest53 2923777] add new error control [changerequest53
 1 file changed, 32 insertions(+)
 create mode 100644 index.html
```



Entonces, recibimos una llamada avisándonos de otro problema urgente en el sitio web y debemos resolverlo inmediatamente. Al usar Git, no necesitamos mezclar el nuevo problema con los cambios que ya habíamos realizado sobre el cambio en la rama `feature/changerequest53`; 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 ***main*** y continuar trabajando a partir de allí.

Pero, antes de poder hacer eso, debemos saber 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 (*git stash* y corregir confirmaciones). Podemos volver a retomar los cambios guardados con el comando git stash pop. Por ahora, como tenemos confirmados todos los cambios, podemos saltar a la rama ***main*** sin problemas:

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (feature/changerequest53)
$ git checkout main
error: Your local changes to the following files would be overwritten by checkout:
        index.html
Please commit your changes or stash them before you switch branches.
Aborting
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (feature/changerequest53)
$ git stash
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory
Saved working directory and index state WIP on changerequest53: 2923777 add new error control [changerequest53
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (feature/changerequest53)
$ git status
On branch feature/changerequest53
nothing to commit, working tree clean
```

```
$ git checkout main

Switched to branch 'main'
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (feature/changerequest53)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (main)
$
```

Tras esto, tendremos el directorio de trabajo exactamente igual a como estaba antes de comenzar a trabajar sobre el cambio `feature/changerequest53` y podremos concentrarnos 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.

A continuación, es momento de resolver el problema urgente. Vamos a crear una nueva rama *hotfix*, sobre la que trabajar hasta resolverlo:

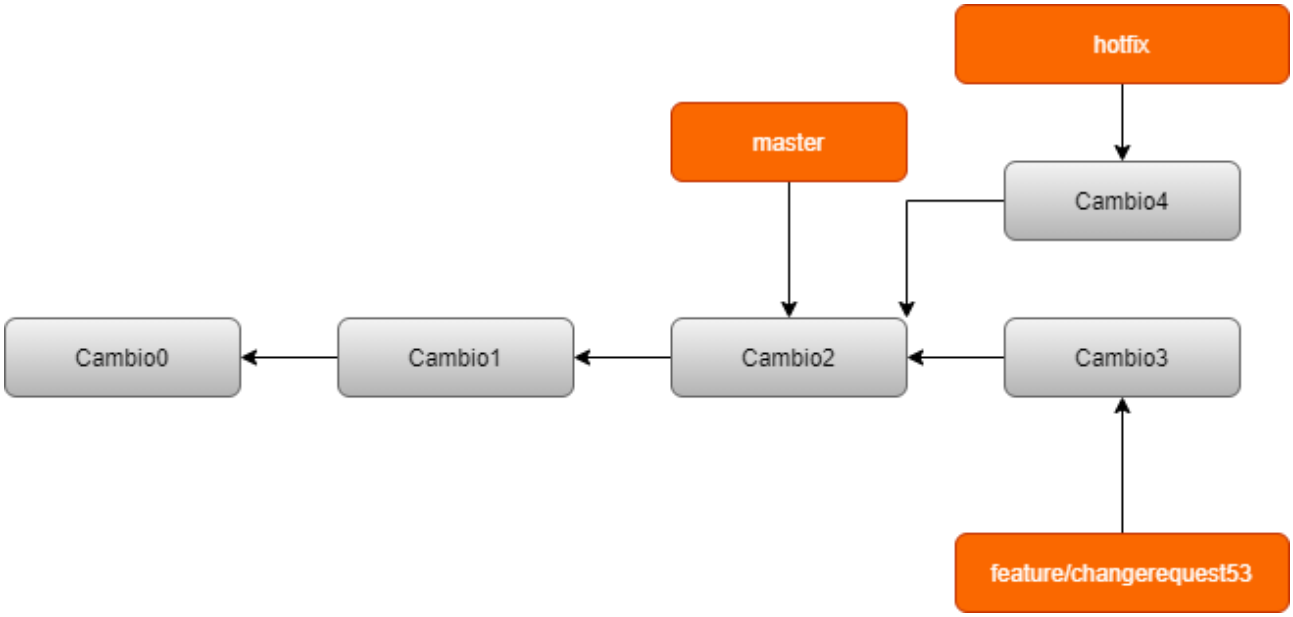
```
$ git checkout -b hotfix

Switched to a new branch 'hotfix'

$ touch index.html

$ git add .

$ git commit -a -m 'Fixed the broken email address problem'
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (main)
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (hotfix)
$ nano index.html
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (hotfix)
$ git add .
warning: LF will be replaced by CRLF in index.html.
The file will have its original line endings in your working directory
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (hotfix)
$ git commit -a -m "Fixed the broken email address problem"
[hotfix 67e9d3c] Fixed the broken email address problem
 1 file changed, 32 insertions(+)
 create mode 100644 index.html
```



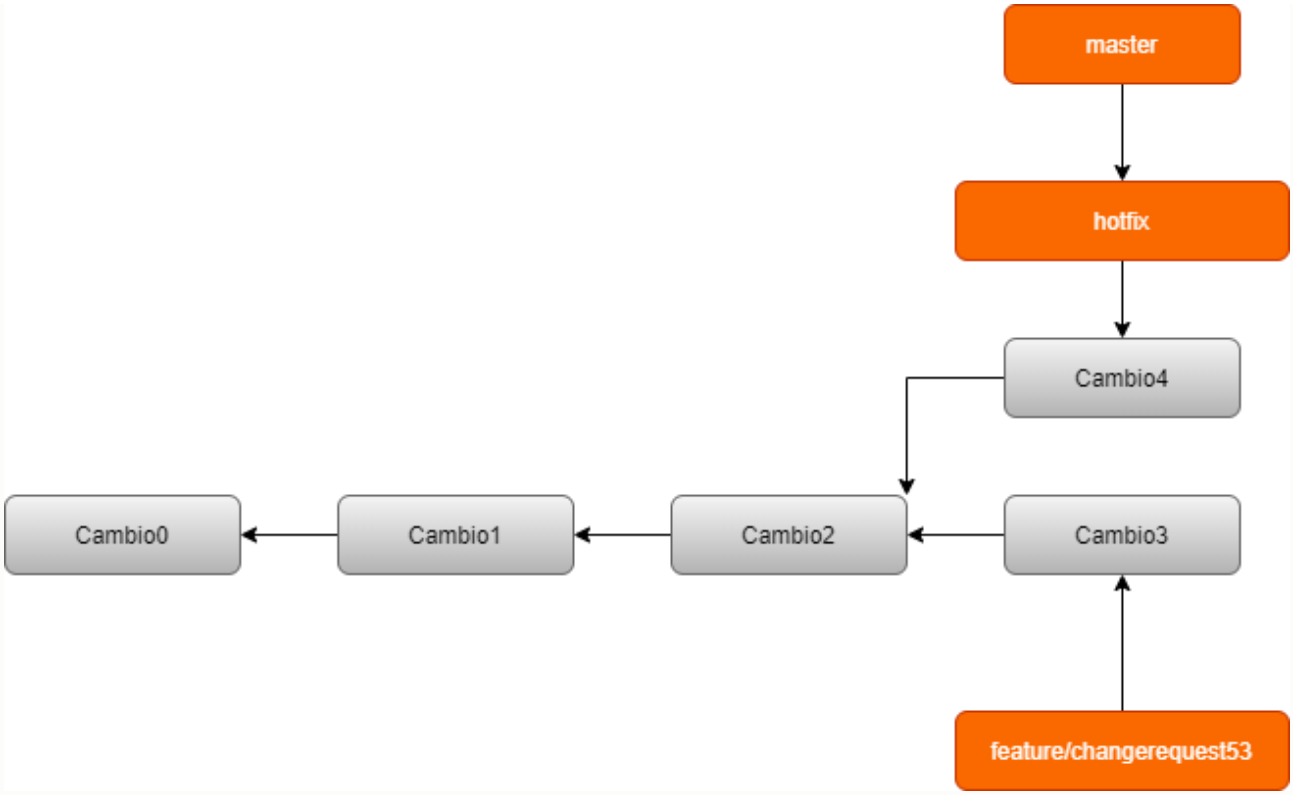
Debemos realizar las pruebas oportunas, asegurarnos de que la solución es correcta, e incorporar los cambios a la rama *main* para ponerlos en producción. Esto se hace con el comando '**git merge'**:

```
$ git checkout main

$ git merge hotfix
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (hotfix)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (main)
$ git merge hotfix
Updating 16a3507..67e9d3c
Fast-forward
 index.html | 32 ++++++++++++++++++++++++++++++++
 1 file changed, 32 insertions(+)
 create mode 100644 index.html
```

Nos aparecerá 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. 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 main. Y podemos desplegarlos.



Tras haber resuelto el problema urgente que había interrumpido nuestro trabajo, podemos volver a donde estábamos. 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 `main`. Esto lo podemos hacer con la opción -d del comando `git branch`:

```
$ git branch -d hotfix

Deleted branch hotfix (3a0874c).
```

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (main)
$ git branch -d hotfix
Deleted branch hotfix (was 67e9d3c).
```

Ahora podríamos volver a retomar los cambios en la rama de feature.

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (main)
$ git checkout feature/changerequest53
Switched to branch 'feature/changerequest53'

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/entornos (feature/changerequest53)
$ git stash pop
On branch feature/changerequest53
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (542a524c1e8c3ac559756c994df04dfeb5419f23)
```

6.3 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 pretendemos fusionar, Git no será capaz de fusionarlas directamente. Por ejemplo, si en nuestro trabajo sobre la rama `feature/changerequest53` hemos modificado una misma porción que también ha sido modificada en el problema `hotfix`, veremos un conflicto como este:

```
$ git merge feature/changerequest53

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 resolvamos nosotros directamente el conflicto. Para ver qué archivos permanecen sin fusionar en un determinado momento conflictivo de una fusión, debemos usar el comando `git status`.

6.3.1 Creación de un conflicto de fusión

Para familiarizarte de verdad con los conflictos de fusión, vamos a crear un conflicto para examinarlo y resolverlo posteriormente.

- Crea un nuevo directorio llamado testmerge, se cambia a ese directorio y lo inicia como nuevo repositorio de Git.
- Crea un nuevo archivo de texto pruebamerg.txt con algo de contenido en él.
- Añade pruebamerge.txt al repositorio y lo confirma.

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones (main)
$ mkdir testmerge

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones (main)
$ cd testmerge

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (main)
$ git init .
Initialized empty Git repository in C:/Users/rhtuf/ramificaciones/testmerge/.git/

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master)
$ ls -la
total 4
drwxr-xr-x 1 rhtuf 197609 0 Mar 25 11:39 ./
drwxr-xr-x 1 rhtuf 197609 0 Mar 25 11:39 ../
drwxr-xr-x 1 rhtuf 197609 0 Mar 25 11:39 .git/

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master)
$ echo "prueba merger" >pruebamerg.txt

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master)
$ git add .
warning: LF will be replaced by CRLF in pruebamerg.txt.
The file will have its original line endings in your working directory

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master)
$ git commit -m "subida prueba merge"
[master (root-commit) da5dc63] subida prueba merge
 1 file changed, 1 insertion(+)
 create mode 100644 pruebamerg.txt
```

Ahora tenemos un nuevo repositorio con una rama main y un archivo pruebamerg.txt con contenido en él.

A continuación, crearemos una nueva rama para utilizarla como fusión conflictiva:

- Crea y verifica una nueva rama llamada nuevarama
- Sobrescribe el contenido de pruebamerg.txt
- Confirma el nuevo contenido

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master)
$ git checkout -b nuevarama
Switched to a new branch 'nuevarama'

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (nuevarama)
$ echo "sobreeescribo la línea">pruebamerg.txt

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (nuevarama)
$ git commit -am "Edito el contenido de pruebamerg.txt"
warning: LF will be replaced by CRLF in pruebamerg.txt.
The file will have its original line endings in your working directory
[nuevarama 4fa0a02] Edito el contenido de pruebamerg.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Con esta nueva rama, hemos creado una confirmación que sobrescribe el contenido de pruebamerg.txt.

En la siguiente cadena de comandos verifica la rama main, añade contenido a pruebamerg.txt y lo confirma. Esto ahora pone nuestro repositorio de ejemplo en un estado en el que tenemos 2 nuevas confirmaciones. Una en la rama main y otra en la rama nuevarama. En este momento, vamos a hacer git merge en nuevarama y vemos qué pasa.

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (nuevarama)
$ git checkout master
Switched to branch 'master'

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master)
$ echo "contenido añadido" >pruebamerg.txt

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master)
$ git commit -am "añadido contenido a pruebamerg.txt"
warning: LF will be replaced by CRLF in pruebamerg.txt.
The file will have its original line endings in your working directory
[master de9815b] añadido contenido a pruebamerg.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
```

En este momento, vamos a hacer git merge en nuevarama y vemos qué pasa.

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master)
$ git merge nuevarama
Auto-merging pruebamerg.txt
CONFLICT (content): Merge conflict in pruebamerg.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Tal y como vemos, Git generará un resultado descriptivo para indicarnos que se ha producido un CONFLICTO. Podemos obtener más información ejecutando el comando git status.

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

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

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

El resultado de git status indica que hay caminos sin fusionar debido a un conflicto. El archivo pruebamerg.txt ahora aparece con el estado modificado. Vamos a examinar el archivo y ver lo que se ha modificado.

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master|MERGING)
$ cat pruebamerg.txt
<<<<<<< HEAD
contenido añadido
=======
sobreeescribo la línea
>>>>>>> nuevarama
```

Aquí hemos usado el comando cat para mostrar el contenido del archivo pruebamerg.txt. Podemos ver algunas nuevas adiciones extrañas:

- <<<<<<< HEAD
- =======
- >>>>>>> nuevarama

Considera estas nuevas líneas como "divisorias de conflictos". La línea ======= es el "centro" del conflicto. Todo el contenido entre el centro y la línea <<<<<<< HEAD es contenido que existe en la rama actual maestra a la que apunta la referencia HEAD. De manera alternativa, todo el contenido entre el centro y >>>>>>> nuevarama es contenido que está presente en nuestra rama de fusión.

La forma más directa de resolver un conflicto de fusión es editar el archivo conflictivo. Abre el archivo pruebamerg.txt en tu editor favorito. Para nuestro ejemplo, simplemente vamos a eliminar todas las líneas divisorias de conflictos y nos quedaremos con las líneas que queramos.

Una vez que el archivo se ha editado, utiliza git add pruebamerg.txt para organizar el nuevo contenido fusionado. Para finalizar la fusión, crea una nueva confirmación ejecutando lo siguiente:

```
rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master|MERGING)
$ nano pruebamerg.txt

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master|MERGING)
$ git add pruebamerg.txt

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master|MERGING)
$ git commit -m "Resuelto conflicto fusión"
[master 5f43354] Resuelto conflicto fusión

rhtuf@LAPTOP-HGE11LLJ MINGW64 ~/ramificaciones/testmerge (master)
$ git merge nuevarama
Already up to date.
```

6.3.2 Comandos de Git que pueden ayudar a resolver los conflictos de fusión

```
git status
```

El comando status se utiliza frecuentemente cuando se trabaja con Git y durante una fusión ayudará a identificar los archivos con conflictos.

```
git log --merge
```

Al pasar el argumento --merge al comando git log, se creará un registro con una lista de confirmaciones que entran en conflicto entre las ramas que se van a fusionar.

```
git diff
```

diff ayuda a encontrar diferencias entre los estados de un repositorio/unos archivos. Esto es útil para predecir y evitar conflictos de fusión.

```
git checkout
```

checkout puede utilizarse para *deshacer* los cambios a archivos o para cambiar ramas.

```
git merge --abort
```

Si se ejecuta git merge con la opción --abort, se saldrá del proceso de fusión y volverá a poner la rama en el estado que tenía antes de que empezara la fusión.

```
git reset
```

git reset puede utilizarse durante un conflicto de fusión para restablecer los archivos conflictivos a un estado que se sabe que es bueno.