TEMA VI

CONTROL DE VERSIONES

Branchs(Ramas)

Las ramas son una de las principales utilidades que disponemos en Git para llevar un mejor control del código. Se trata de una bifurcación del estado del repositorio que crea un nuevo camino, de cara a la evolución del código, en paralelo a otras ramas que se puedan generar.

Las ramas nos pueden servir para muchos casos de uso. Por ejemplo:

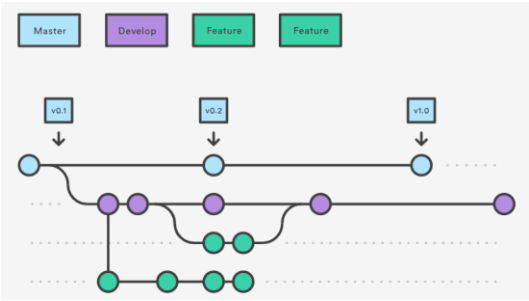
- Para la creación de una funcionalidad que queramos integrar en un programa y para la cual no queremos que la rama principal se vea afectada. Esta función experimental se puede realizar en una rama independiente, de modo que, aunque tardemos varios días o semanas en terminarla, no afecte a la producción del código que tenemos en la rama principal y que permanecerá estable.

- Para realizar pruebas en un punto determinado y seguir avanzando a la vez con otra parte del código

- Para probar distintas posibilidades, por ejemplo estamos desarrollando un proyecto web y queremos ver distintas apariencias.

- Trabajamos en un proyecto complejo con varios programadores y cada uno codifica una parte.

El trabajo con ramas resulta muy cómodo en el desarrollo de proyectos, porque es posible que todas las ramas creadas evolucionen al mismo tiempo, pudiendo el desarrollador pasar de una rama a otra en cualquier momento según las necesidades del proyecto. Si en un momento dado el trabajo con una rama nos ha resultado interesante, útil y se encuentra estable, entonces podemos fusionar ramas para incorporar las modificaciones del proyecto en su rama principal.



Por defecto se empieza a trabajar en la rama master o principales

Crear una rama

Utilizamos el comando **git branch nombre**

Para cambiar de una rama a otro **git checkout nombredelarama**

Para saber en qué rama estamos **git branch**

Una vez estamos en la rama creada podemos trabajar en el proyecto y las demás ramas no se verán afectadas

Cuando subimos una nueva rama a github, nos muestra la opción new pull request

Un pull request es una petición que el propietario de un fork de un repositorio hace al propietario del repositorio original para que este último incorpore los commits que están en el fork

Para eliminar una rama, desde otra **git branch -D nombre**

Fusionar dos ramas

Para fusionar dos ramas, estando en una de ella utilizamos el comando

git merge nombrerama

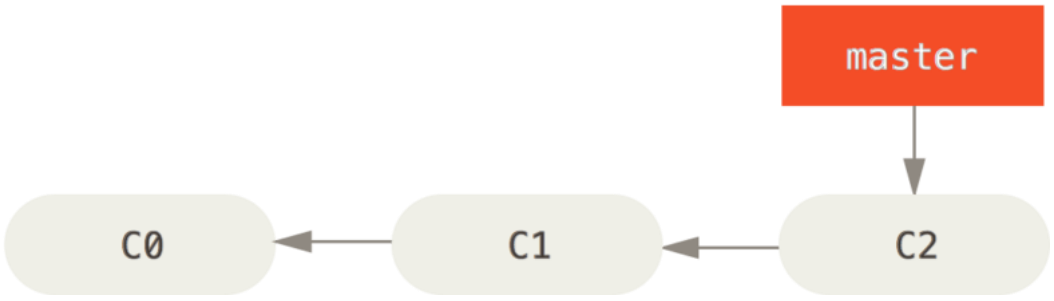
Veamos un ejemplo simple de ramificar y de fusionar, con un flujo de trabajo que se podría presentar en la realidad. Imagina que sigues los siguientes pasos:

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.

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

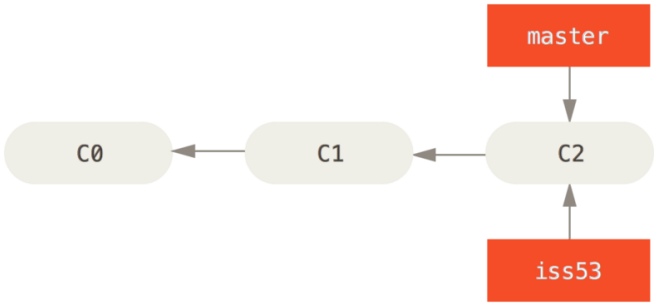


Decides trabajar en el problema que me comunican (#53). Para crear una nueva rama y saltar a ella, en un solo paso, puedes utilizar el comando `git checkout` con la opción `-b`:

```
$ git checkout -b iss53
```

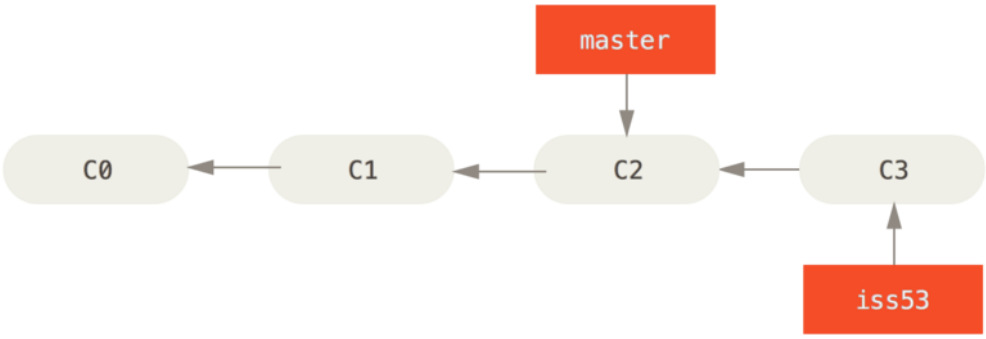
Esto es un atajo para:

```
$ git branch iss53
$ git checkout 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 (checkout) en este momento

```
$ git commit -a -m 'added a new footer [issue 53]'
```



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.

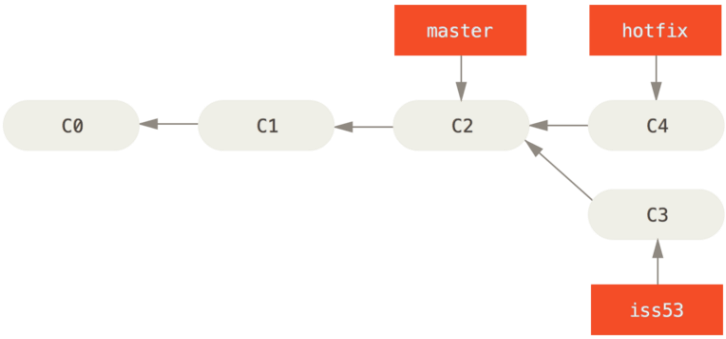
Pero, antes de poder hacer eso, hemos de tomar en cuenta que, si tenemos cambios aún no confirmados en el directorio de trabajo o en el área de preparación, Git no nos

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. Como tenemos confirmados todos los cambios, podemos saltar a la rama `master` sin problemas:

```
$ git checkout 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 es exactamente igual a la que había en 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:



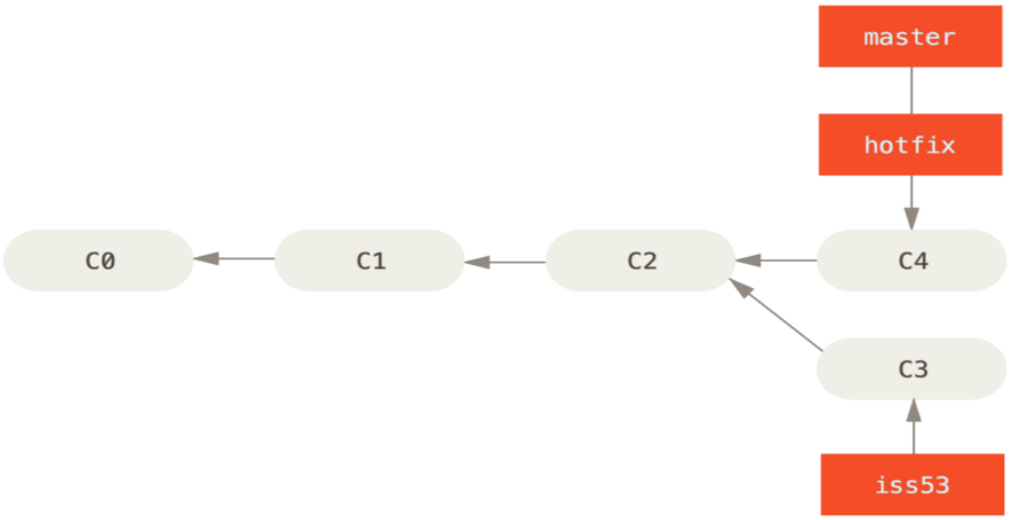
Puedes realizar las pruebas oportunas, asegurarte de que la solución es correcta, e incorporar los cambios a la rama `master` para ponerlos en producción. Esto se hace con el comando `git merge`:

```
$ 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.

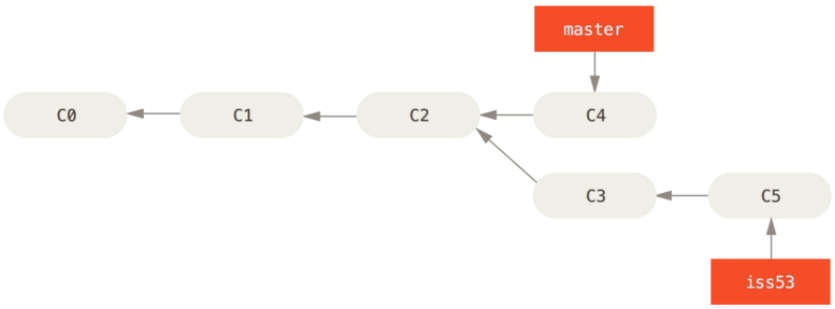


Tras haber resuelto el problema urgente que había interrumpido tu trabajo, puedes volver a donde estabas. Pero antes, es importante borrar la rama `hotfix`, ya que no la vamos a necesitar más, puesto que apunta exactamente al mismo sitio que la rama `master`. Esto lo puedes hacer con la opción `-d` del 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
$ git commit -a -m 'finished the new footer [issue 53]'
```

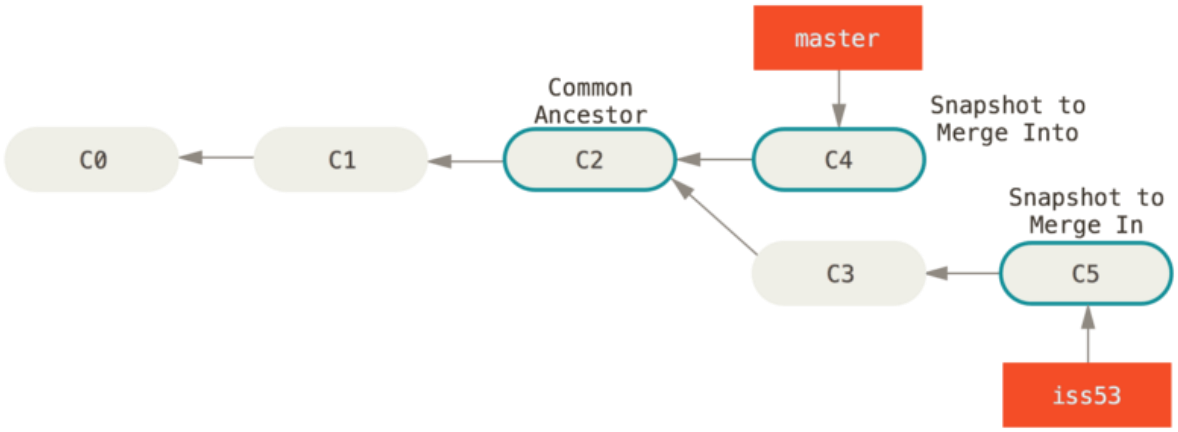


Hay que tener en cuenta que todo el trabajo realizado en la rama `hotfix` no está en los archivos de la rama `iss53`. Si fuera necesario agregarlos, puedes fusionar (merge) la rama `master` sobre la rama `iss53` utilizando el comando `git merge master`, o puedes esperar hasta que decidas fusionarlas.

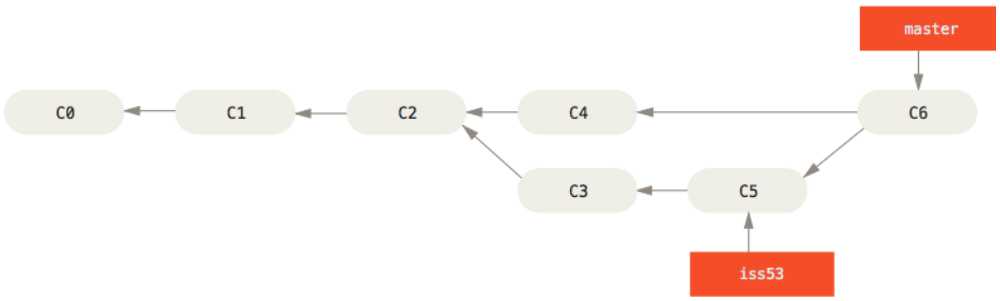
Supongamos que tu trabajo con el problema #53 ya está completo y listo para fusionarlo con la rama `master`. Activa (checkout) la rama donde deseas fusionar y lanza el comando `git merge`:

```
$ git checkout master
$ git merge iss53
```

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.



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.



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 cambiar manualmente el contenido de uno o de otro lado.

Fork

Es una ramificación de todo el proyecto, nos será muy útil a la hora de trabajar en equipo.

Supongamos que un desarrollador tiene un repositorio con un proyecto y una serie de commits. Es público y por tanto accesible para cualquiera. Otro desarrollador tiene que implementar un proyecto y descubre que lo que hay en el repositorio del primero le es muy útil y puede empezar a partir de ahí.

1. Si se hace un fork, copia el repositorio del primer desarrollador en su github. Tendrá un repositorio nuevo que es una copia.

2. Lógicamente, este segundo desarrollador clonará este repositorio en uno suyo local.

3. A partir de aquí desarrolla su nuevo proyecto y al terminar hace un commit en su repositorio local

4. Aquí es dónde apreciamos la ventaja de haber hecho un fork porque se puede hacer un **pull request**. Esto es una "sugerencia" al desarrollador del primer proyecto para que pueda ver tus "cambios" y decida si quiere aceptar o no el commit. Si lo acepta se produce un merge

Para hacer un fork, desde mi repositorio hago una búsqueda al repositorio que quiero "clonar" en el mío. Tendré la opción fork, es tan simple como pulsarla.

A partir de aquí, clono en mi repositorio local y trabajo en el proyecto. Hago commit y un push a mi repositorio github. Hasta aquí ninguna novedad.

Para hacer un pull request basta con pulsar sobre la opción. Lo primero que hace es comparar el repositorio "origen" con nuestros cambios. Si se puede hacer el merge nos avisa y nos pide confirmación. A partir de aquí se establece un "chat" entre el dueño del repositorio y yo.

Al dueño del repositorio le aparecerá que tiene un pull request y podrá ver el commit mío, con los cambios en los archivos. En la opción Revertir Cambios podrá aceptarlos, revertirlos, mensajear al segundo desarrollador.

Para probar podemos utilizar un repositorio que github proporciona para estas pruebas

octocat/Spoon-Knife.