Git

Trabajando con repositorios en GitHub

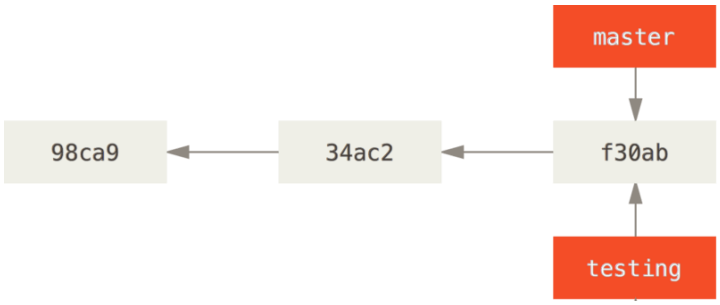
Branches

Cualquier sistema de control de versiones moderno tiene algún mecanismo para soportar el uso de ramas. Cuando hablamos de ramificaciones, significa que tú has tomado la rama principal de desarrollo (master) y a partir de ahí has continuado trabajando sin seguir la rama principal de desarrollo.

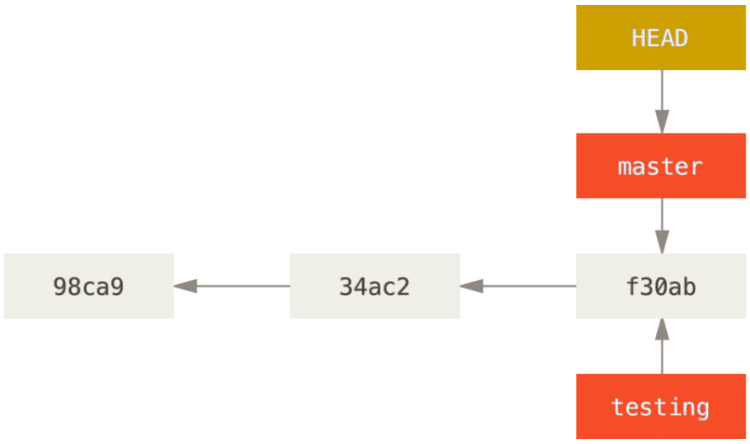
¿Qué sucede cuando creas una nueva rama? Bueno…, simplemente se crea un nuevo apuntador para que lo puedas mover libremente. Por ejemplo, supongamos que quieres crear una rama nueva denominada "testing". Para ello, usarás el comando `git branch`:

Esto creará un nuevo apuntador apuntando a la misma confirmación donde estés actualmente.

```
$ git branch testing
```



Y, ¿cómo sabe Git en qué rama estás en este momento? Pues…, mediante un apuntador especial denominado HEAD. Aunque es preciso comentar que este HEAD es totalmente distinto al concepto de HEAD en otros sistemas de control de cambios como Subversion o CVS. En Git, es simplemente el apuntador a la rama local en la que tú estés en ese momento, en este caso la rama `master`; pues el comando `git branch` solamente crea una nueva rama, pero no salta a dicha rama.

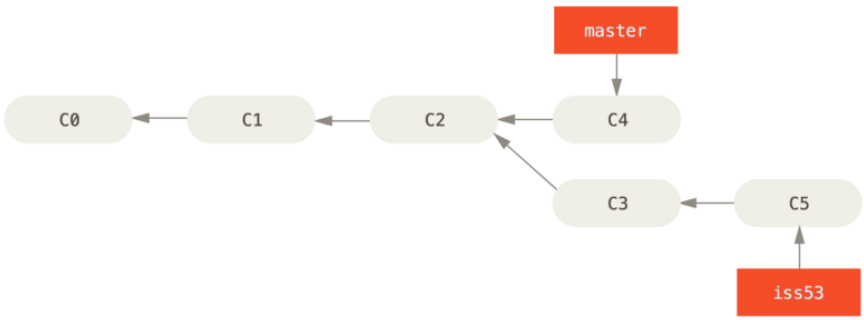


Para saltar de una rama a otra, tienes que utilizar el comando `git checkout`. Hagamos una prueba, saltando a la rama `testing` recién creada:

```
$ git checkout testing
```

Merge

Partiendo del siguiente diagrama:



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(+)
```

Conflicts

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.

Comandos avanzados

Pull request

Las pull requests son una funcionalidad que facilita la colaboración entre desarrolladores. Cuando realizas una pull request, lo que haces es solicitar que otro desarrollador (por ejemplo, el mantenedor del proyecto) incorpore (o haga un pull) una rama de tu repositorio al suyo. Por tanto, para realizar esta solicitud, debes proporcionar la siguiente información: el repositorio de origen, la rama de origen, el repositorio de destino y la rama de destino.

Fork

La función de fork en Git se define como una operación usual en el sistema de Git que se encarga de la creación de una copia de un repositorio en la cuenta de usuario.

Cabe destacar que este repositorio copiado será igual al repositorio desde el que se realiza el fork en Git. A pesar de esto, cuando se cree la copia, cada repositorio se ubicará en espacios diferentes y tendrán la posibilidad de evolucionar de forma diferente, de acuerdo a las acciones del usuario en cada uno de estos recursos de Git.

Rebase

El comando git rebase te permite cambiar fácilmente una serie de confirmaciones, modificando el historial de tu repositorio. Puedes reordenar, editar o combinar confirmaciones.

- Normalmente, se usaría git rebase para lo siguiente:
- Editar mensajes de confirmación previos.
- Combinar varias confirmaciones en una.
- Eliminar o revertir confirmaciones que ya no son necesarias.

Cambiar de base las confirmaciones con una rama

Para cambiar de base todas las confirmaciones entre otra rama y el estado de rama actual, puedes ingresar el siguiente comando en tu shell (ya sea el símbolo del sistema para Windows o la terminal para Mac y Linux):

```
$ git rebase --interactive OTHER-BRANCH-NAME
```

Cambiar de base las confirmaciones en un momento específico

Para cambiar de base las últimas confirmaciones en tu rama actual, puedes ingresar el siguiente comando en tu shell:

```
$ git rebase --interactive HEAD~7
```

Stash

El comando git stash almacena temporalmente (o guarda en un stash) los cambios que hayas efectuado en el código en el que estás trabajando para que puedas trabajar en otra cosa y, más tarde, regresar y volver a aplicar los cambios más tarde. Guardar los cambios en stashes resulta

práctico si tienes que cambiar rápidamente de contexto y ponerte con otra cosa, pero estás en medio de un cambio en el código y no lo tienes todo listo para confirmar los cambios.

El comando git stash coge los cambios sin confirmar (tanto los que están preparados como los que no), los guarda aparte para usarlos más adelante y, acto seguido, los deshace en el código en el que estás trabajando. Por ejemplo:

```
$ git status
On branch main
Changes to be committed:

    new file:   style.css

Changes not staged for commit:

    modified:   index.html

$ git stash
Saved working directory and index state WIP on main: 5002d47 our n
ew homepage
HEAD is now at 5002d47 our new homepage

$ git status
On branch main
nothing to commit, working tree clean
```

Puedes volver a aplicar los cambios de un stash mediante el comando git stash pop:

```
$ git status
On branch main
nothing to commit, working tree clean
$ git stash pop
On branch main
Changes to be committed:

    new file:   style.css

Changes not staged for commit:

    modified:   index.html

Dropped refs/stash@{0} (32b3aa1d185dfe6d57b3c3cc3b32cbf3e380cc6a)
```

Clean

El comando `git clean` se utiliza para eliminar archivos no deseados de tu directorio de trabajo. Esto podría incluir la eliminación de artefactos de construcción temporal o la fusión de archivos en conflicto.

Cherry-pick

`git cherry-pick` es un potente comando que permite que las confirmaciones arbitrarias de Git se elijan por referencia y se añadan al actual HEAD de trabajo. La ejecución de cherry-pick es el acto de elegir una confirmación de una rama y aplicarla a otra. `git cherry-pick` puede ser útil para deshacer cambios. Por ejemplo, supongamos que una confirmación se aplica accidentalmente en la rama equivocada. Puedes cambiar a la rama correcta y ejecutar cherry-pick en la confirmación para aplicarla a donde debería estar.

Para demostrar cómo utilizar git cherry-pick, supongamos que tenemos un repositorio con el siguiente estado de rama:

```
    a - b - c - d   Main
         \
           e - f - g Feature
```

Usar git cherry-pick es sencillo y se puede ejecutar de la siguiente manera:

```
git cherry-pick commitSha
```

En este ejemplo, commitSha es una referencia de confirmación. Puedes encontrar una referencia de confirmación con el comando git log. En este caso, imaginemos que queremos aplicar la confirmación f a la rama principal. Para ello, primero debemos asegurarnos de que estamos trabajando con la rama principal.

```
git checkout main
```

A continuación, ejecutamos cherry-pick con el siguiente comando:

```
git cherry-pick f
```

Una vez ejecutado, el historial de Git se verá así:

```
    a - b - c - d - f   Main
         \
           e - f - g Feature
```

La confirmación f se ha introducido correctamente en la rama principal.