GIT

Uso de ramas y repositorios remotos

GIT

Uso de ramas

RAMAS(master,...)

Una rama Git es simplemente un apuntador móvil apuntando a un commit.

La **rama por defecto** de Git es la rama **master** (el tronco del árbol).

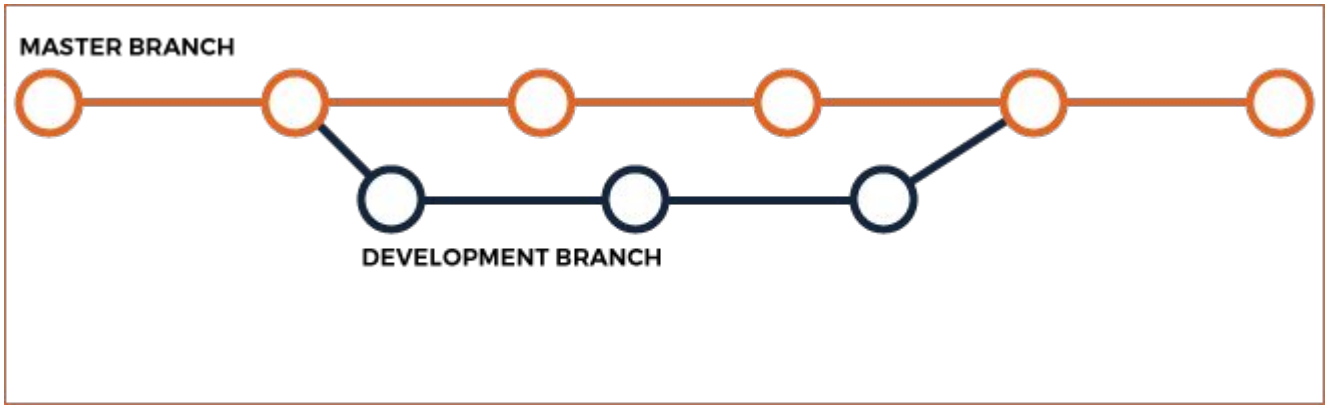
Con la primera confirmación de cambios que realicemos, se creará esta rama principal **master** apuntando a dicha confirmación.

En cada confirmación de cambios que realicemos, la rama irá avanzando automáticamente. Y la rama **master** apuntará siempre a la última confirmación realizada.

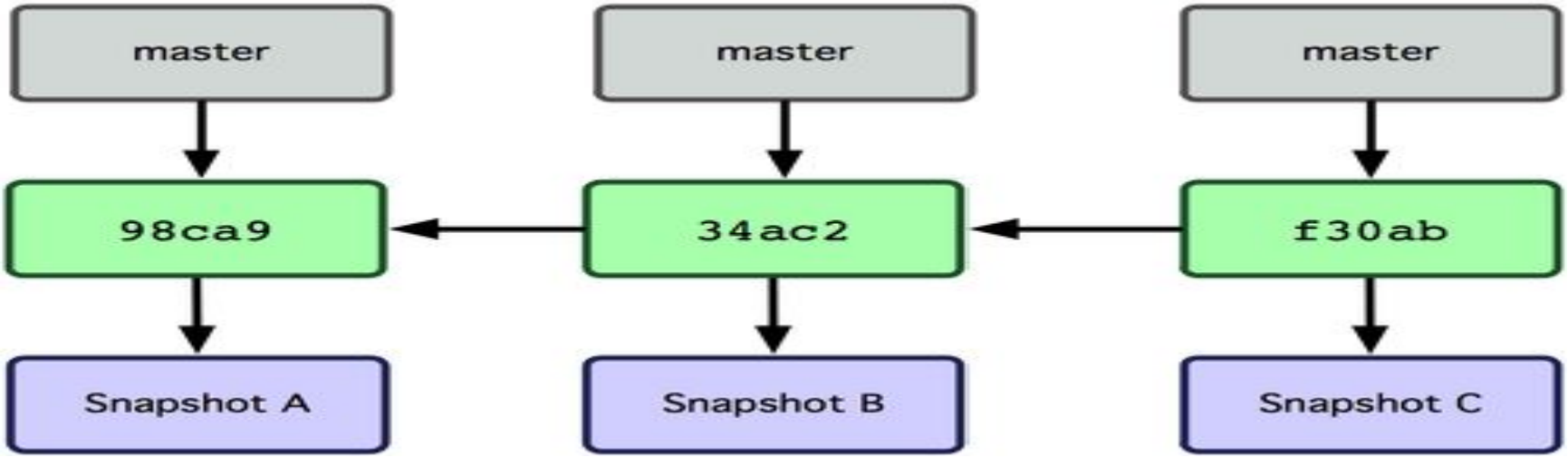
RAMAS(master,...)

Las ramas son utilizadas para desarrollar funcionalidades aisladas unas de otras.

Usa otras ramas para el desarrollo y fusiona la rama principal cuando termina.



Ejemplo (Rama master)



Comandos básicos

- Inicializar repositorio:
  - `git init`
- Obtener repositorio del servidor
  - `git clone https://gitlab.com/….`
- Ver el estado del repositorio:
  - `git status`
- Ver estructura actual:
  - Resumen: `git branch`
  - De una rama concreta: `git show-branch nombreRama`
  - De todas las ramas: `git show-branch -a`
- Crear rama:
  - `git branch nombreRama`

Comandos básicos

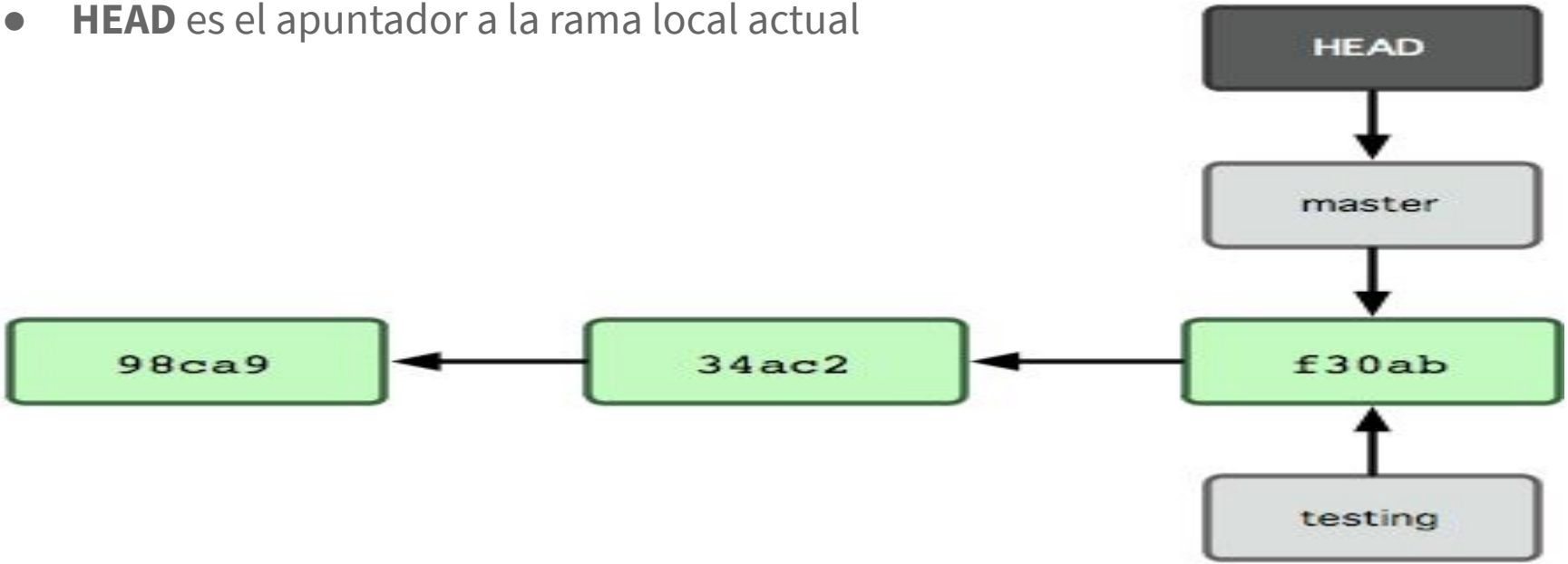
- Configuración:
  - Asignación de usuario y mail
    - `git config --global user.name "ElTeuNom"`
    - `git config --global user.email elTeu@correu.com`
  - Asignación de herramienta de merge, en este caso “meld”
    - `git config --global merge.tool meld`
    - `git config --global diff.tool meld`
- Flujo de trabajo típico en local
  - Añadir modificar ficheros en área de trabajo
  - git add .
  - git commit -m “Mensaje clarificador!!!”
    - ó git commit -am “Mensaje clarificador!!!”

Creamos nueva rama “Testing”

- `git branch testing`
- Podemos comprobar si se ha creado viendo la estructura:
  - `git show-branch -a --list`
- Aparece asterisco donde estamos actualmente

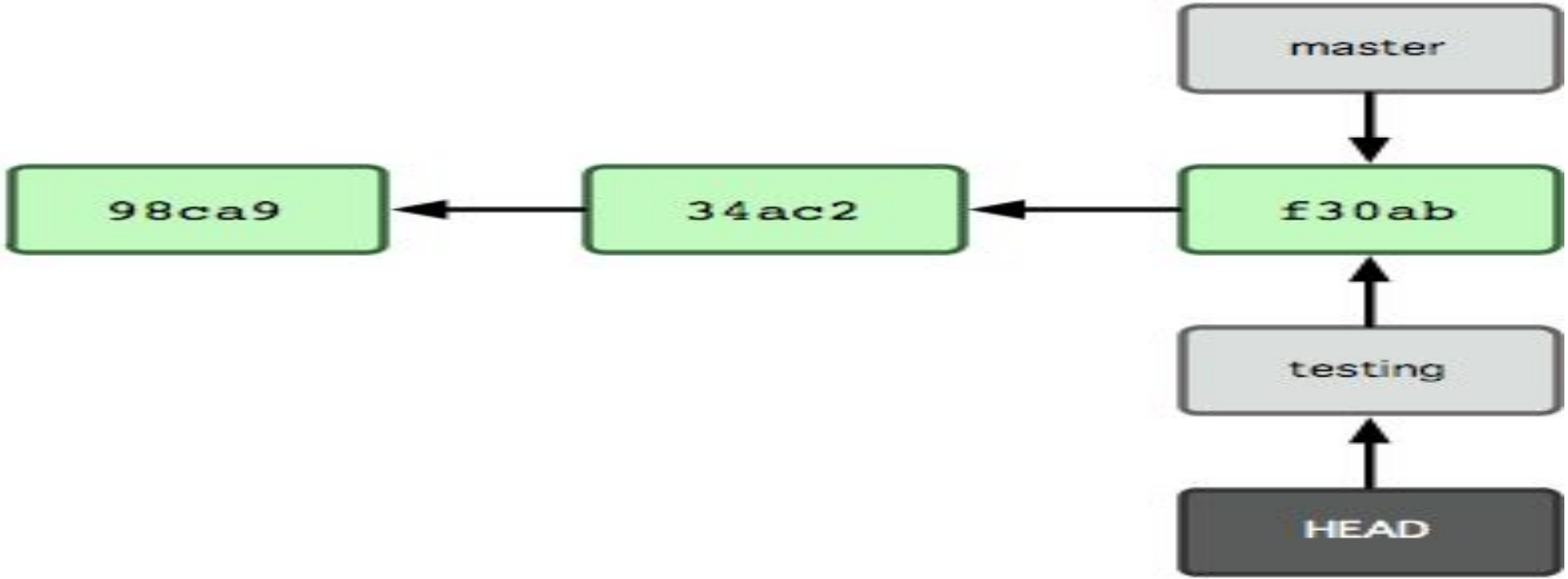
¿Dónde estamos? “**HEAD**”.

- Y, ¿cómo sabe Git en qué rama estás en este momento?
- Pues..., mediante un apuntador especial **HEAD**.
- **HEAD** es el apuntador a la rama local actual



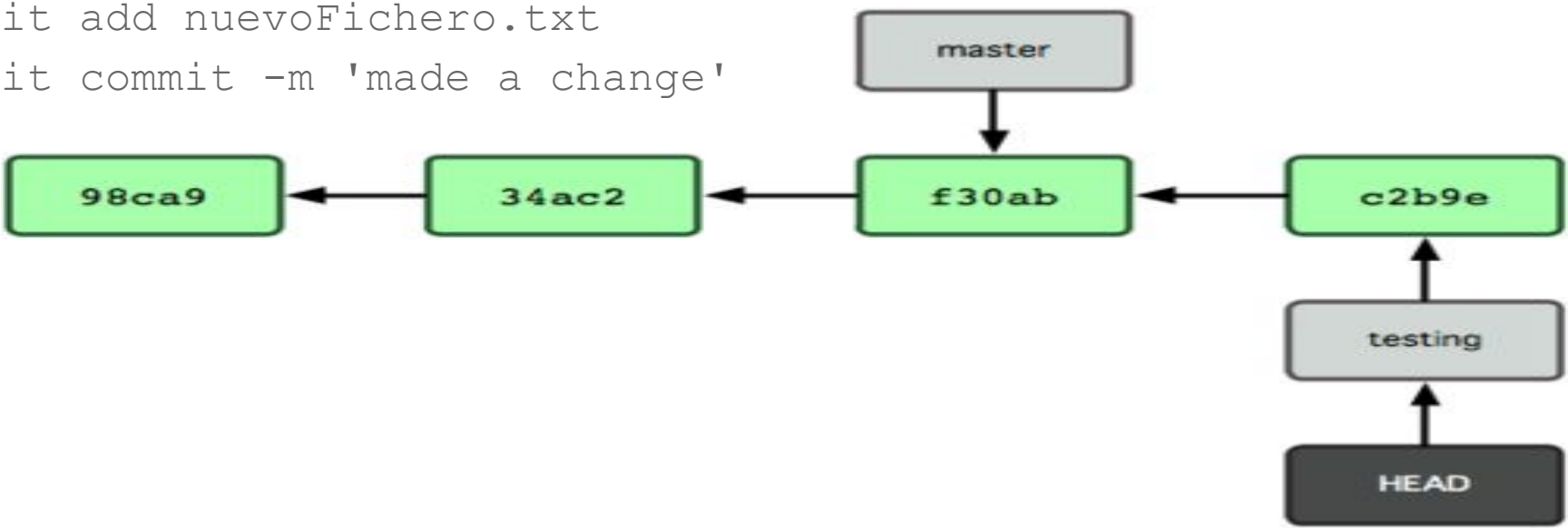
Cambio a la rama “testing”?

- `git checkout testing`
  - //nos movemos a la nueva rama “testing” que hemos creado anteriormente, y se actualiza “HEAD”



Ahora hacemos cambios en la rama “testing”

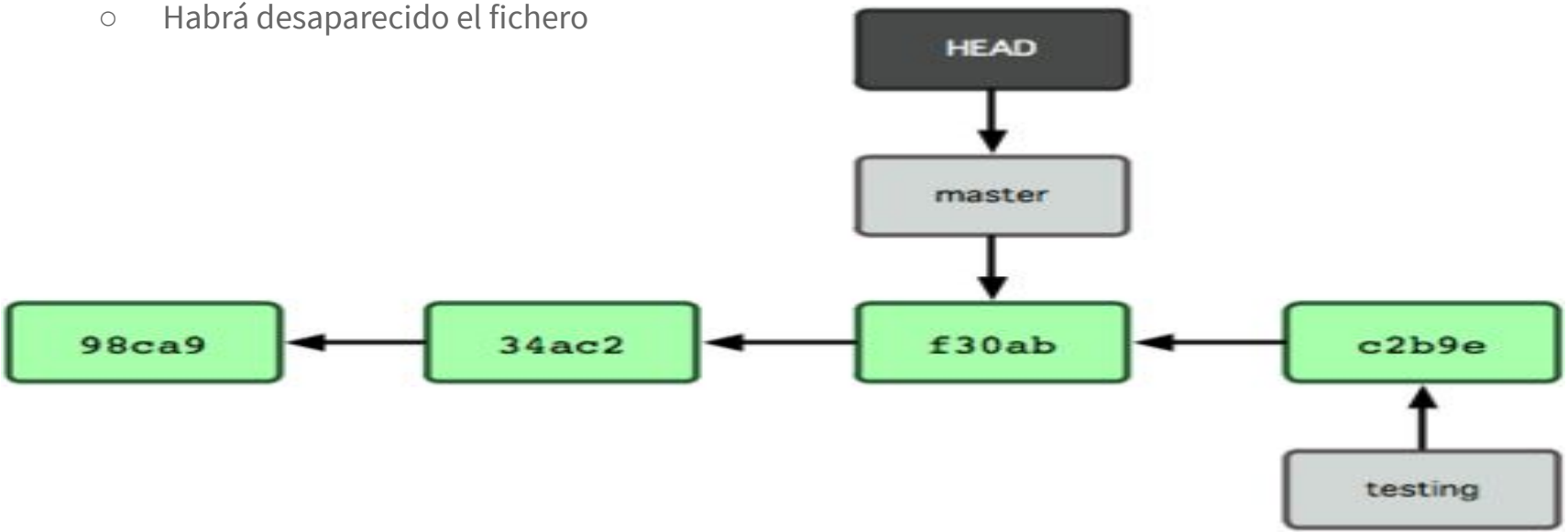
- `vi nuevoFichero.txt`
- `git add nuevoFichero.txt`
- `git commit -m 'made a change'`



- Ahora la rama **master** y la rama **testing** son distintas

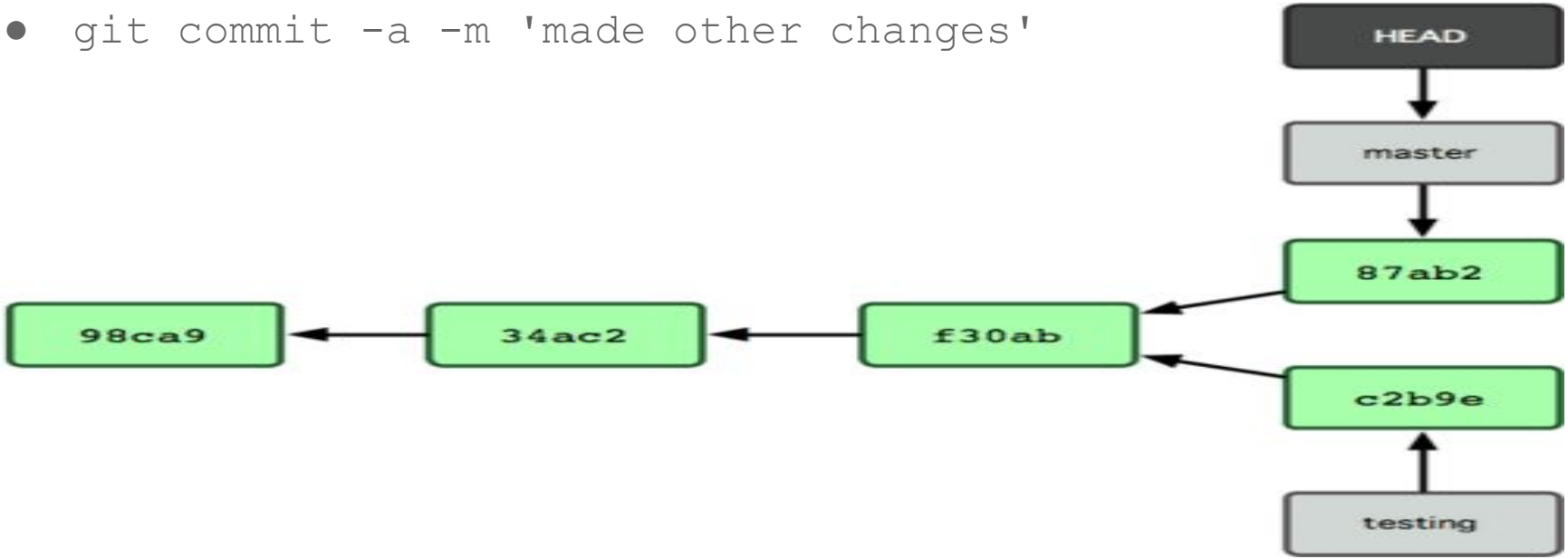
¿Cómo volvemos a la rama master?

- `git checkout master` //al hacer “checkout”, se mueve el puntero HEAD
- `git show-branch -a --list`
  - Habrá desaparecido el fichero



Hacemos un cambio distinto en master

- `vi otroFichero.txt` //creo otro fichero
- `git add otroFichero.txt`
- `git commit -a -m 'made other changes'`



Problema

Vamos a presentar un ejemplo simple y “real” de ramificar y de fusionar.

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 (iss53).
3. Realizas algo de trabajo en esa rama.

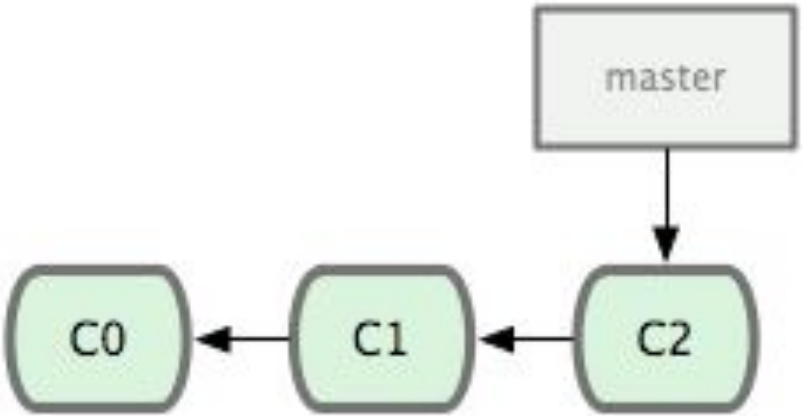
Problema continuación

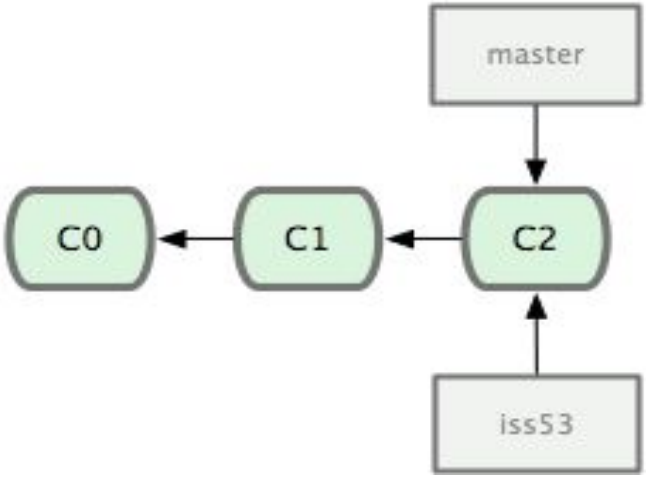
En este momento, recibes una llamada avisando de un problema crítico que tienes que resolver. Y sigues los siguientes pasos:

1. Vuelves a la rama de producción original (**master**).
2. Creas una nueva rama para el problema crítico (**hotfix**) y lo resuelves trabajando en ella.
3. Tras las pertinentes pruebas, fusionas (**merge**) esa rama y la envías (**push**) a la rama de producción (**master**).
4. Vuelves a la rama del tema en que andabas antes de la llamada (**iss53**) y continuas tu trabajo.

1. Creas una nueva rama para el cambio 53

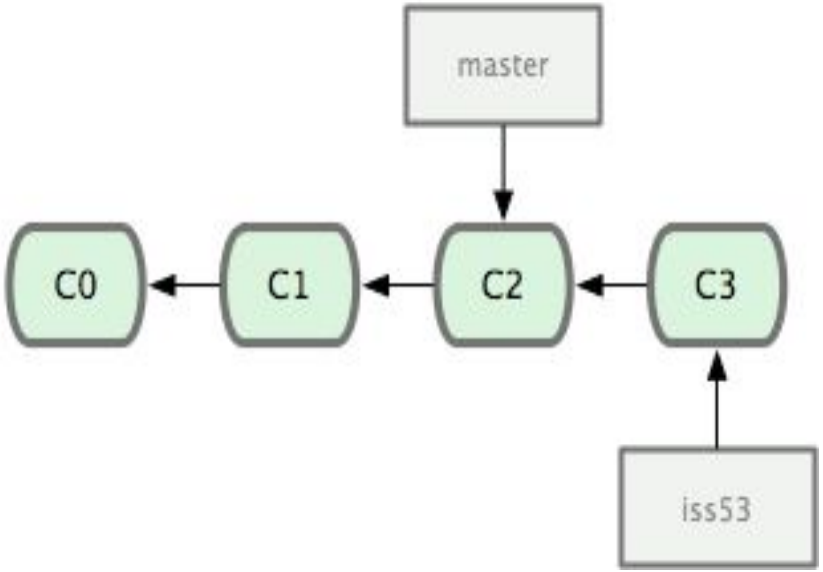
```
git branch iss53
git checkout iss53
git checkout -b iss53 = git branch iss53 + git checkout iss53
```





2. Tú estás tranquilamente trabajando...

- `vim index.html`
- `git add index.html` // si es un nuevo fichero
- `git commit -a -m 'Creating new footer [issue 53]'`



3. ¡Mierda! ¡Me llaman de un problema urgente!

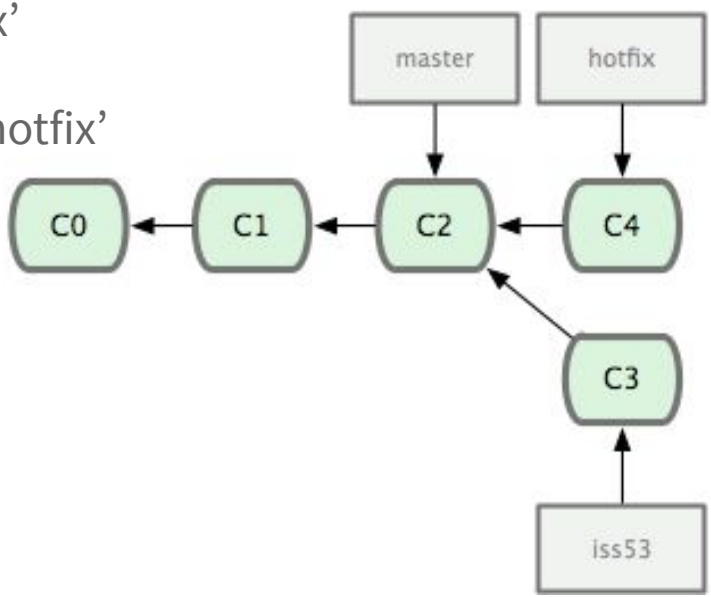
```
git checkout master
```

```
git branch hotfix
```
// Created new branch ‘hotfix’

```
git checkout hotfix
```
// Switch to new branch ‘hotfix’

```
vi readme.html
```

```
git commit -a -m 'fixed broken email'
```
[hotfix]: created 3a0874c:"fixed broken email"
1 files changed, 0 insertions(+), 1 deletions(-)

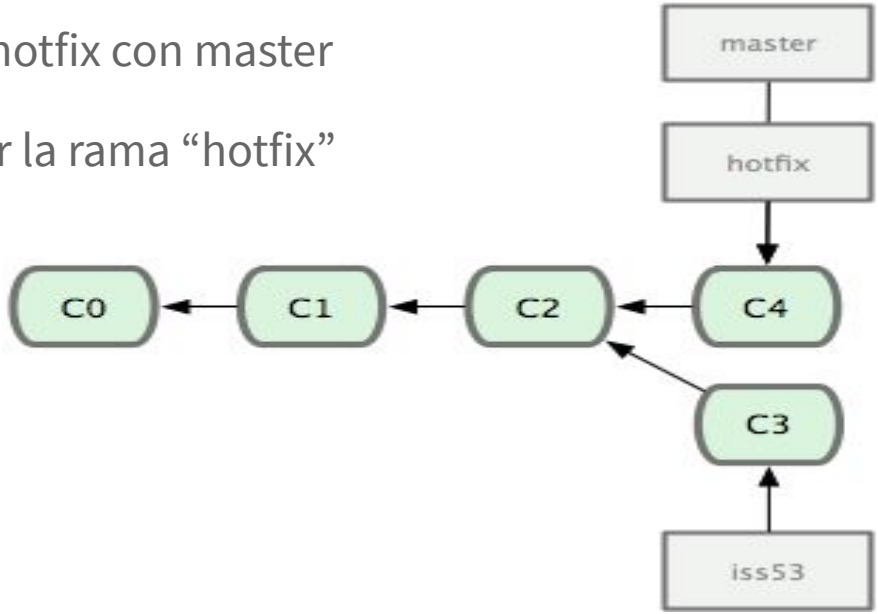


4. Pasamos el cambio a producción

`git checkout master` // Volvemos a la rama master

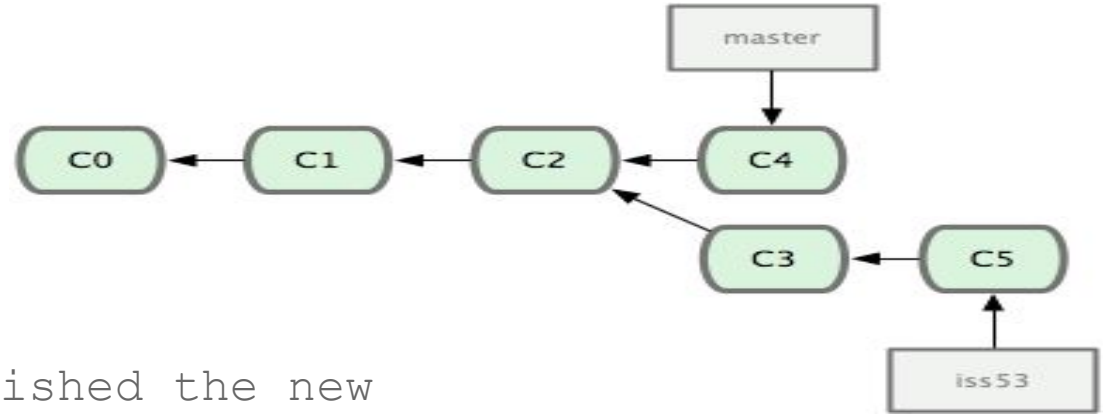
`git merge hotfix` // Fusionamos la rama hotfix con master

`git branch -d hotfix` // PODEMOS borrar la rama “hotfix”



6. Volvemos a la rama “iss53” para seguir con los cambios

```
git checkout iss53 // cambiamos a la rama "iss53"
vi index.html
```




```
git commit -a -m 'finished the new
footer [issue 53]'
```

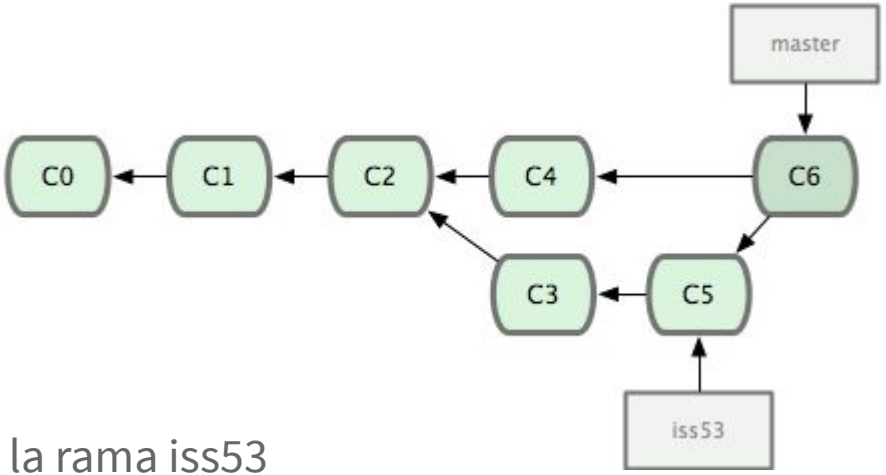
Pero tenemos un problema....

Todos los cambios que hemos hecho en readme.html 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.

7. Fusionamos todos los cambios en la rama “master”

- `git checkout master` // nos movemos a la rama **master**
- `git merge iss53` // le decimos que fusione **master** con la rama **iss53**

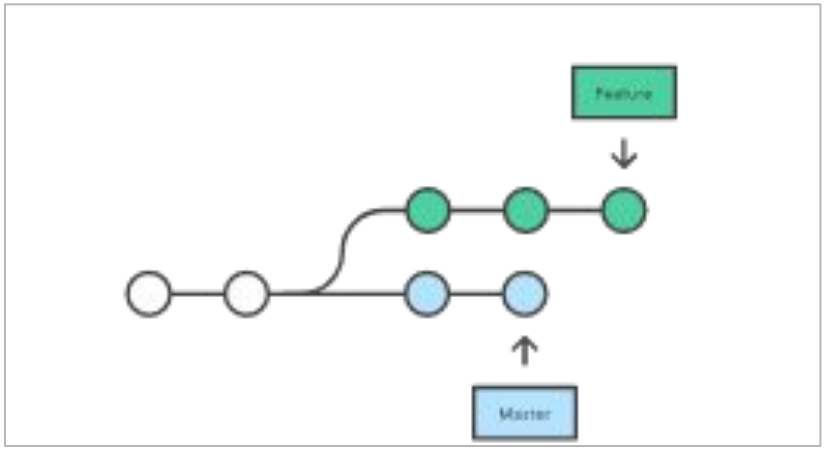


- `git branch -d iss53` // Borro la rama iss53

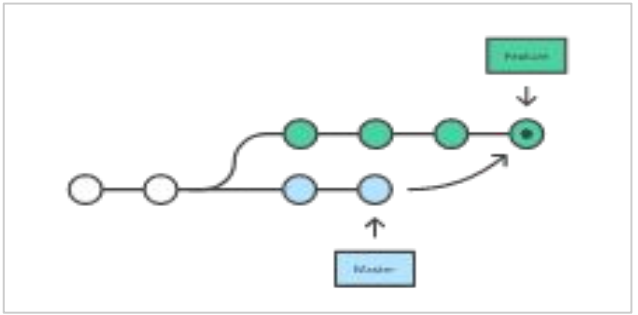
https://git-scm.com/book/es/v1/Ramificaciones-en-Git-Procedimientos-b%C3%A1sicos-para-ramificar-y-fusionar

Merge vs rebase

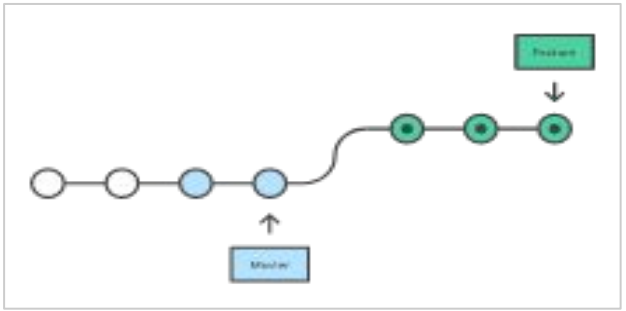
Son dos maneras de integrar cambios entre ramas.



Merge



Rebase



https://www.atlassian.com/git/tutorials/merging-vs-rebasing

Conflictos entre ramas en fusiones.

- Si en tu trabajo del problema #53 (branch **iss53**) has modificado una misma porción (index.html) que también ha sido modificada en el problema hotfix (branch **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.
    ```
- Podemos hacer “git status” para que nos diga el conflicto
    ```

Para ver las diferencias entre dos ramas...

$ git diff master iss53

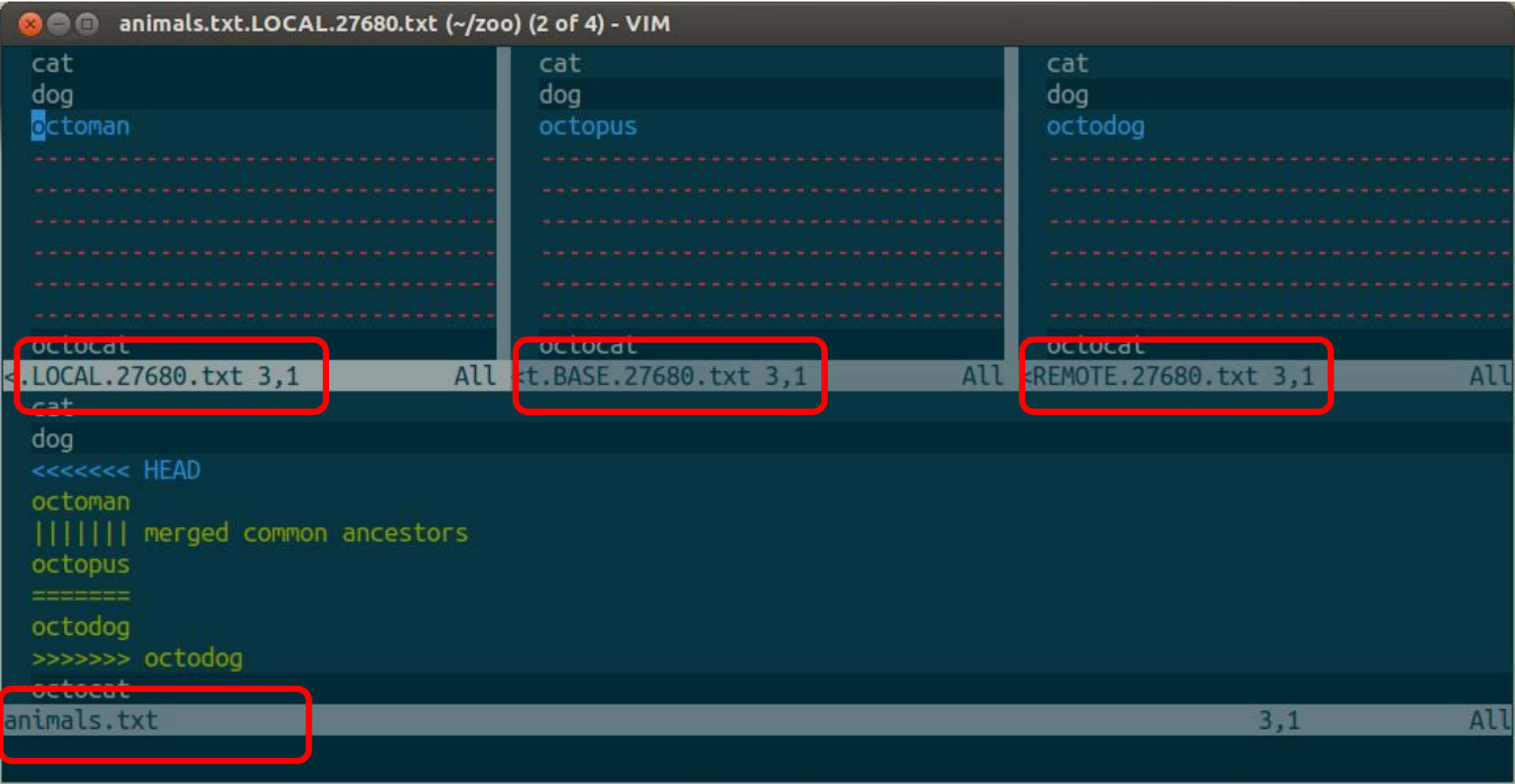
```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

git mergetool

- Hay que hacer configuración inicial con
  - `git config merge.tool vimdiff    //o meld...`
- git **mergetool**

  Merging the files: index.html

  Normal merge conflict for 'index.html':
  {local}: modified
  {remote}: modified
  Hit return to start merge resolution tool (opendiff):
- Documentación: https://gist.github.com/karenyyng/f19ff75c60f18b4b8149

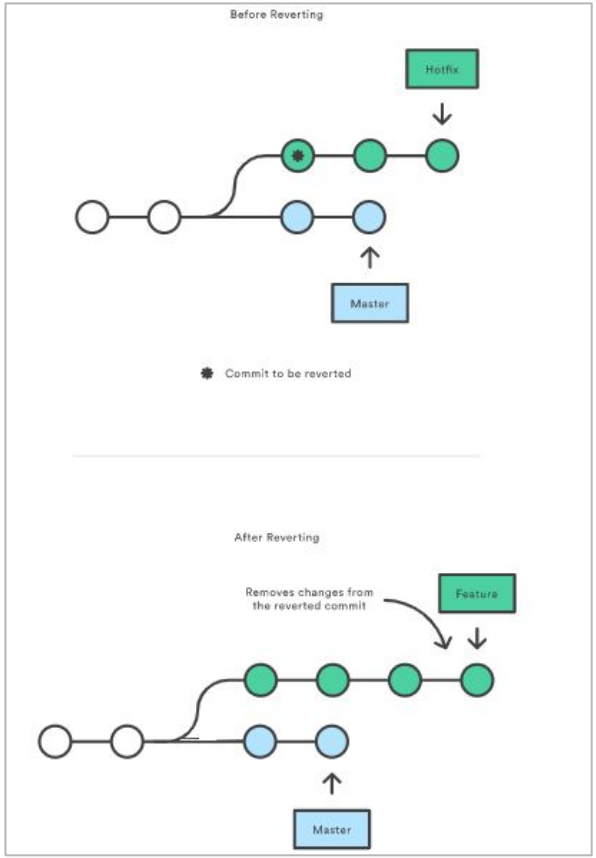

animals.txt.LOCAL.27680.txt (~/zoo) (2 of 4) - VIM
cat
dog
octoman
octocat
.LOCAL.27680.txt 3,1 All
cat
dog
octopus
octocat
t.BASE.27680.txt 3,1 All
cat
dog
octodog
octocat
REMOTE.27680.txt 3,1 All
cat
dog
<<<<<<< HEAD
octoman
||||||| merged common ancestors
octopus
=======
octodog
>>>>>>> octodog
octocat
animals.txt 3,1 All

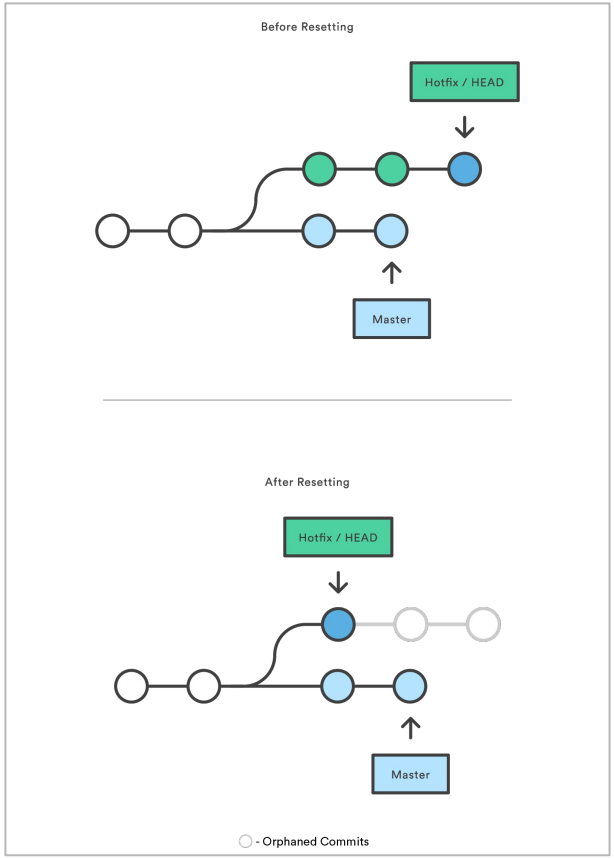
Comandos interesantes

- Mostrar histórico de commits:
  - git log
  - git log --oneline #versión abreviada
  - Opción-p para ver los cambios hechos en los ficheros
- Deshacer un commit concreto:
  - Deshacer los cambios de un commit concreto
  - Para deshacer un commit concreto: git revert #commit
  - No se pierden los commits, no se “manipula” el historial de commits
  - Se crea un nuevo commit con los cambios del commit “recuperado”
- Volver a un commit anterior:
  - Temporalment:
    - git checkout #commit
  - Definitivamente (perdiendo los cambios, reescribiendo la historia):
    - git reset --hard #commit

Deshacer cambios: revert vs reset

https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting





GIT revert vs reset

Command	Scope	Common use cases
`git reset`	Commit-level	Discard commits in a private branch or throw away uncommited changes
`git reset`	File-level	Unstage a file
`git checkout`	Commit-level	Switch between branches or inspect old snapshots
`git checkout`	File-level	Discard changes in the working directory
`git revert`	Commit-level	Undo commits in a public branch
`git revert`	File-level	(N/A)

https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting

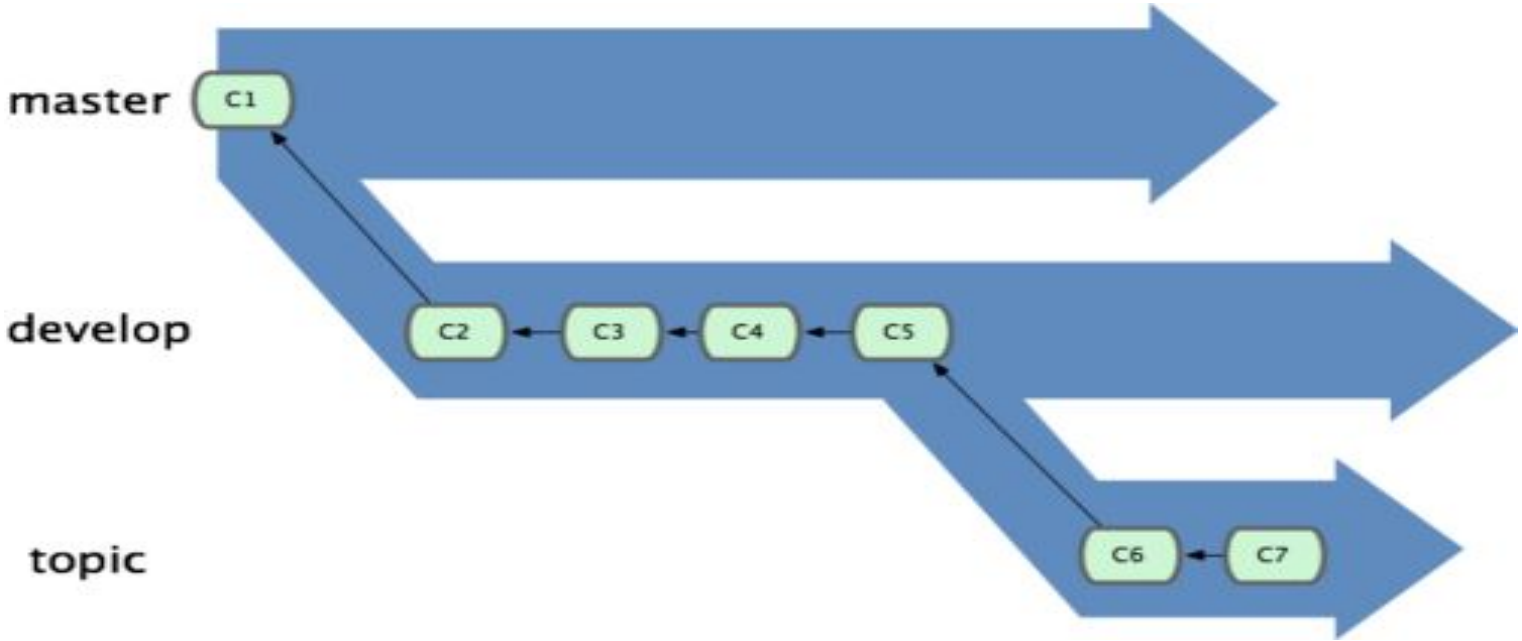
GIT tags

- Una tag/etiqueta de GIT se usa para identificar de una manera más sencilla un commit
- Se suele usar para hacer públicas versiones SW
- Para mostrar las etiquetas disponibles: git tag
- Para crear una etiqueta en el commit actual: git tag -a nomEtiqueta -m 'Missatge'
- Per crear una etiqueta en un commit anterior: git tag -a nombreEtiqueta -m 'Mensage' #idCommit
    - para saber el ID del commit: git log
- Referencias: https://git-scm.com/book/es/v1/Fundamentos-de-Git-Creando-etiquetas

.gitignore

- Para ignorar ciertos ficheros de un repositorio
- Se crea el fichero .gitignore en el directorio del proyecto
- Por ejemplo para nuestros proyectos eclipse:
  - bin/
  - target/
  - *.jar
  - *.class
  - .project
  - .classpath
- Hay que definirlo antes de hacer add. Sinò: http://stackoverflow.com/questions/8021441/git-how-to-ignore-hidden-directories

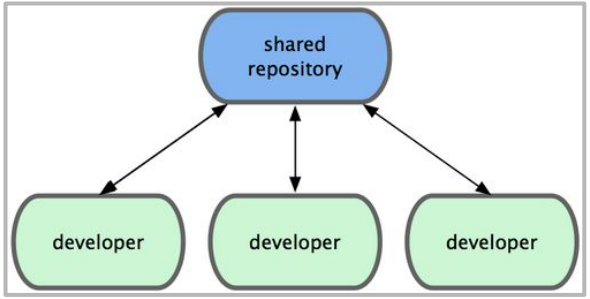
Esquema de ramas

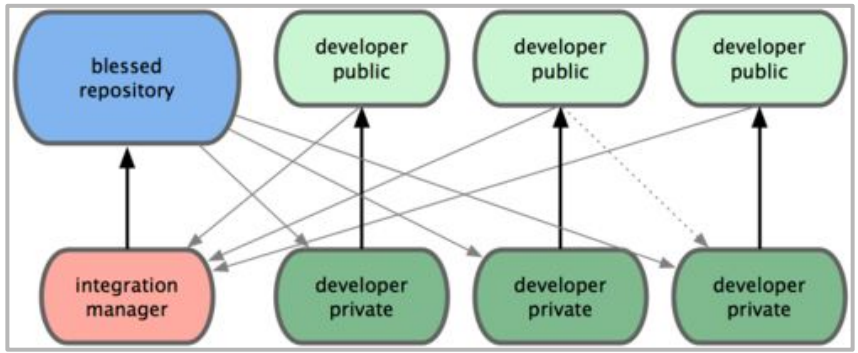


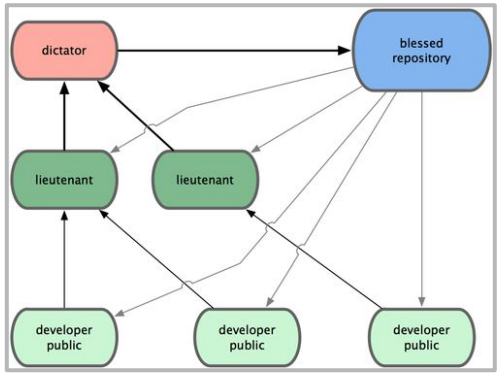
https://git-scm.com/book/es/v1/Ramificaciones-en-Git-Flujos-de-trabajo-ramificados

Flujos de trabajo

- Hay distintas maneras de trabajar en equipo usando GIT:
  - Centralizado
  - Integrador
  - Dictador y tenientes





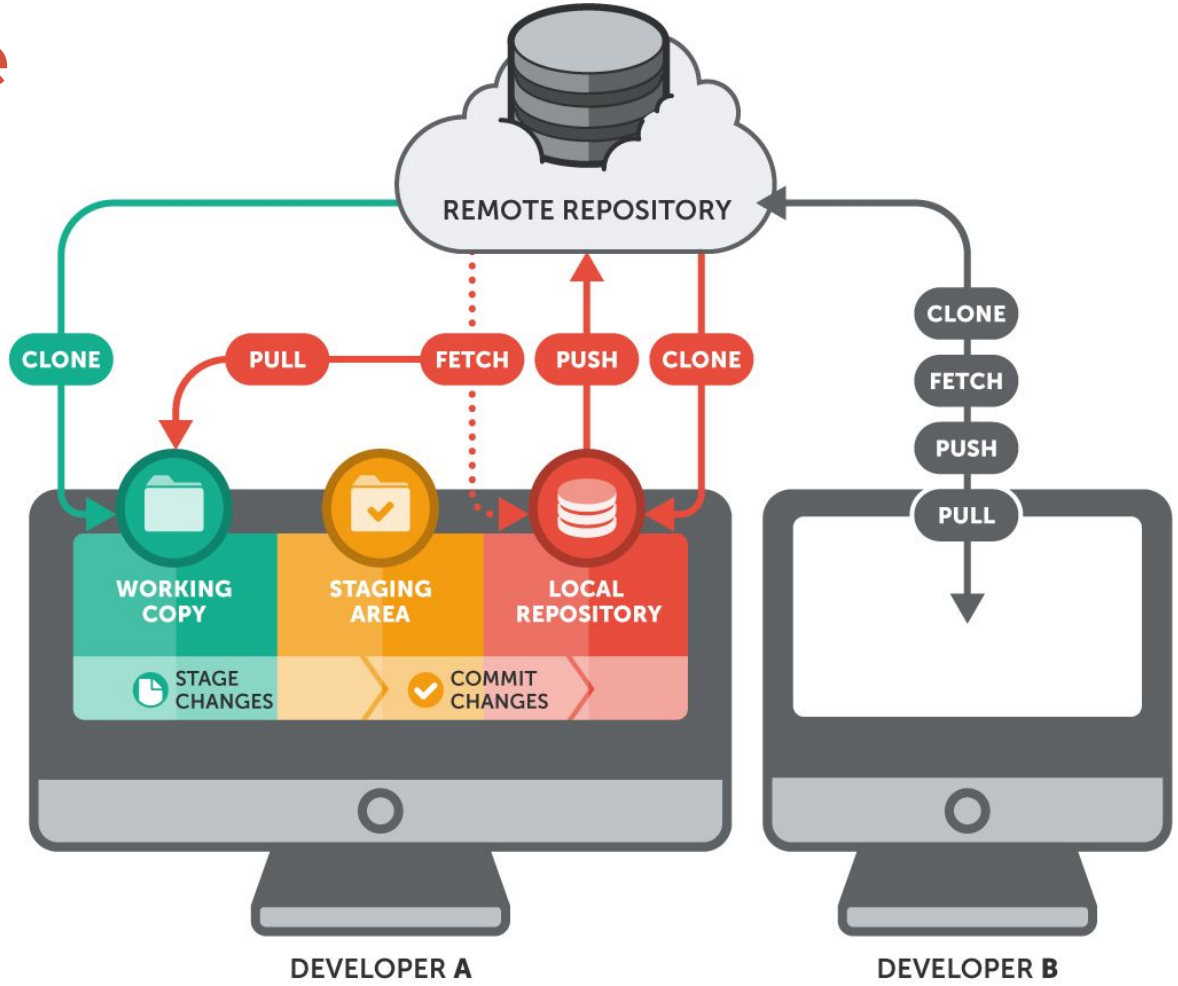


https://git-scm.com/book/es/v1/Git-en-entornos-distribuidos-Flujos-de-trabajo-distribuidos

GIT

Repositorios remotos

GIT remote

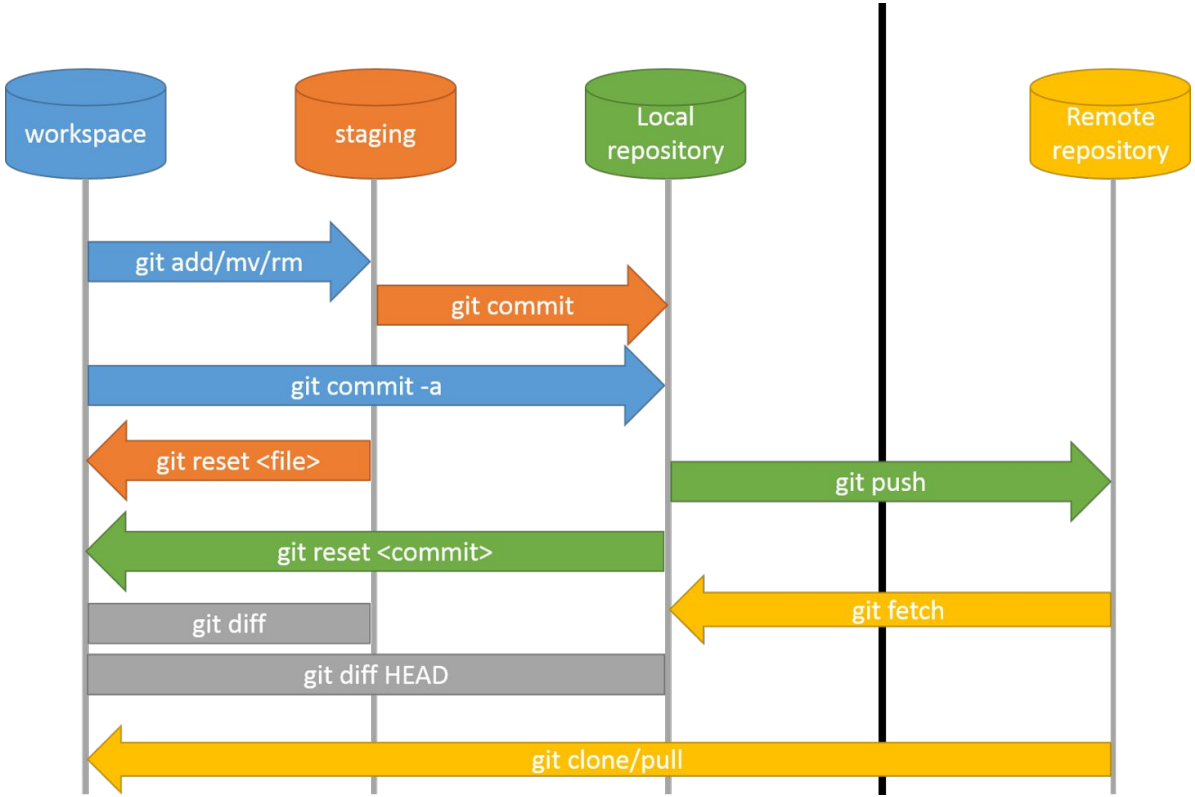

REMOTE REPOSITORY
CLONE
PULL
FETCH
PUSH
CLONE
CLONE
FETCH
PUSH
PULL
WORKING COPY
STAGING AREA
LOCAL REPOSITORY
STAGE CHANGES
COMMIT CHANGES
DEVELOPER A
DEVELOPER B

Comandos básicos

- `git clone <url>`
- `git remote`
- `git fetch <alias>`
- `git merge <alias/rama>`
- `git pull` `# fetch + merge`
- `git push <alias> <rama>`

https://www.atlassian.com/git/tutorials/syncing

Flujo de trabajo bàsico



git clone

- `git clone urlRepo`

Por defecto se crea una conexión **origin** que apunta al repositorio remoto. (como la rama master)

git remote

- `git remote` `#muestra repositorios remotos`
- `git remote -v` `#incluye url`
- `git remote add nombreRepo url`

git fetch

Importa commits del repositorio remoto al local. En ramas remotas.

- `git fetch <remote>`
- `git fetch <remote> <rama>`
- `git remote add nombreRepo url`
    - Para ver las ramas:
    - `$ git branch -r // o --all`
    - `origin/master`
    - `origin/documentacion`
- `git checkout nombreRama`

git fetch

Después del fetch podemos revisar los cambios:

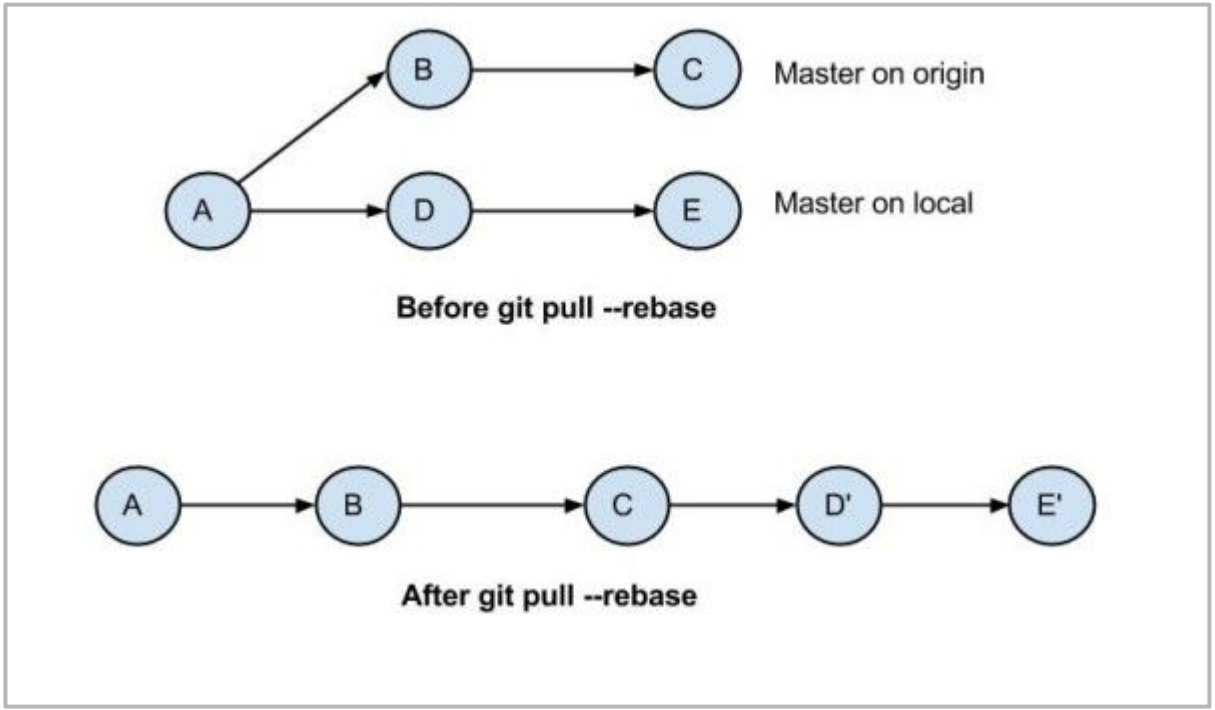
- `git log --oneline master..origin/master`
  - Para ver los cambios. Y si está Ok:
- `git checkout master`
- `git log origin/master`
- `git merge origin/master`

git pull

- `git pull <remote>`

  Hace el fetch y un automáticamente un merge. Si no hay conflictos. Si hay conflictos hay que resolverlos y hacer después el commit

git pull --rebase



En lugar de hacer un merge, nuestros cambios se aplican sobre los últimos cambios del repositorio origen.

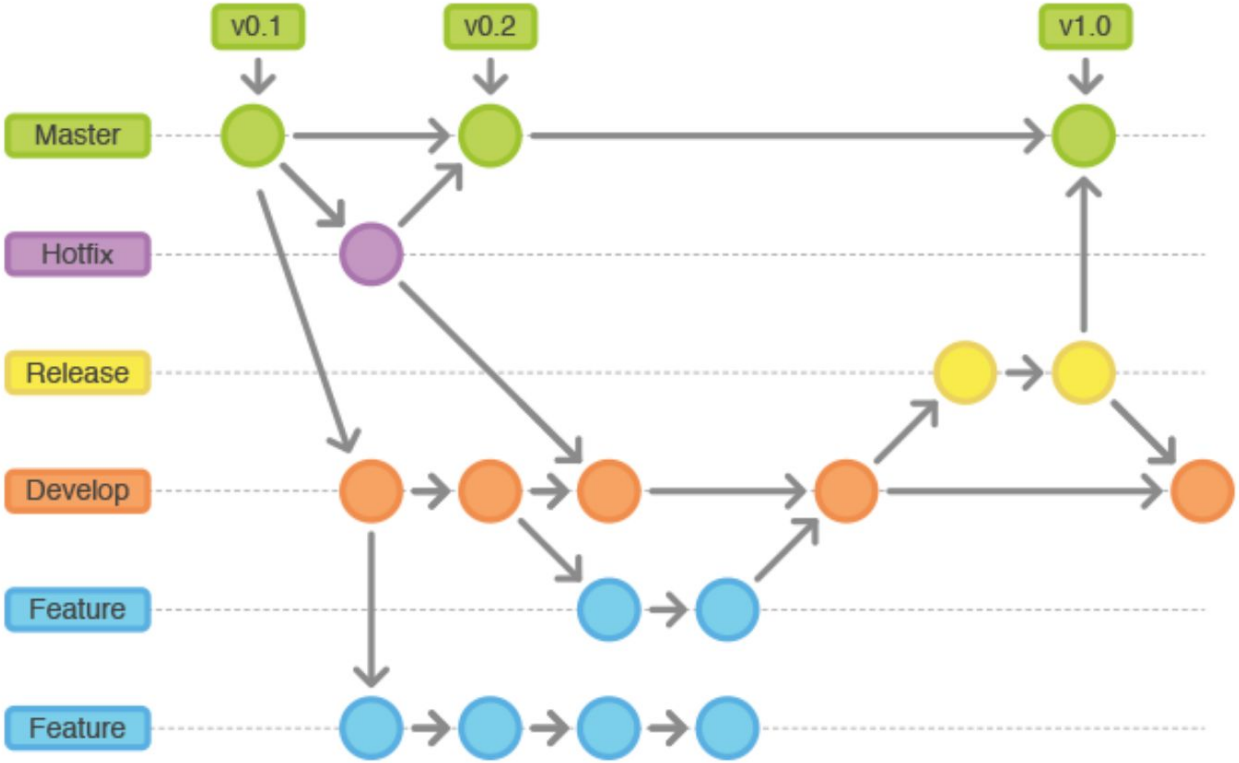
git push

Sube los commits del repositorio local al remoto

- **`git push <alias> <rama>`**
  - `git push origin master`

Sólo podremos hacer el push si nuestro repositorio está actualizado con el remoto. Sino tendremos que hacer un pull, fusionarlo con nuestro trabajo y hacer el push.

Ejemplo de flujo de trabajo



Ejemplo de flujo de trabajo