# Branchs

# GIT

- **Trabajando con ramas en Git**
  - Cualquier sistema de control de versiones moderno tiene algún mecanismo para soportar distintas ramas.
  - Cuando hablamos de branching o trabajo con ramas, significa que se ha seleccionado la rama principal de desarrollo (master) y a partir de ahí se ha continuado trabajando sin seguir la rama principal de desarrollo.
  - En muchos sistemas de control de versiones este proceso es costoso, pues a menudo significa crear una nueva copia del código, lo cual puede requerir mucho tiempo cuando se trata de proyectos grandes.

# GIT

- **Trabajando con ramas en Git**
  - Hemos de examinar la forma en que almacena sus datos.
  - Git no los almacena de forma incremental (guardando solo diferencias), sino que los almacena como una serie de instantáneas (copias puntuales de los archivos completos, tal y como se encuentran en ese momento).

# GIT

- **Trabajando con ramas en Git**
  - En cada confirmación de cambios (commit), Git almacena un punto de control que conserva:
    - un apuntador a la copia puntual de los contenidos preparados (staged),
    - unos metadatos con el autor y el mensaje explicativo
    - uno o varios apuntadores a las confirmaciones (commit) que sean padres directos de esta (un padre en los casos de confirmación normal, y múltiples padres en los casos de estar confirmando una fusión (merge) de dos o mas ramas).

# GIT

- **Trabajando con ramas en Git**
  - Vamos a suponer, por ejemplo, que tenemos una carpeta con tres archivos, que se pasan a stage y se confirman (commit).
  - Al preparar los archivos, Git realiza una suma de control de cada uno de ellos (un resumen SHA-1, tal y como se mencionaba en el capítulo 1), almacena una copia de cada uno en el repositorio (estas copias se denominan "blobs"), y guarda cada suma de control en el área de preparación (staging area):

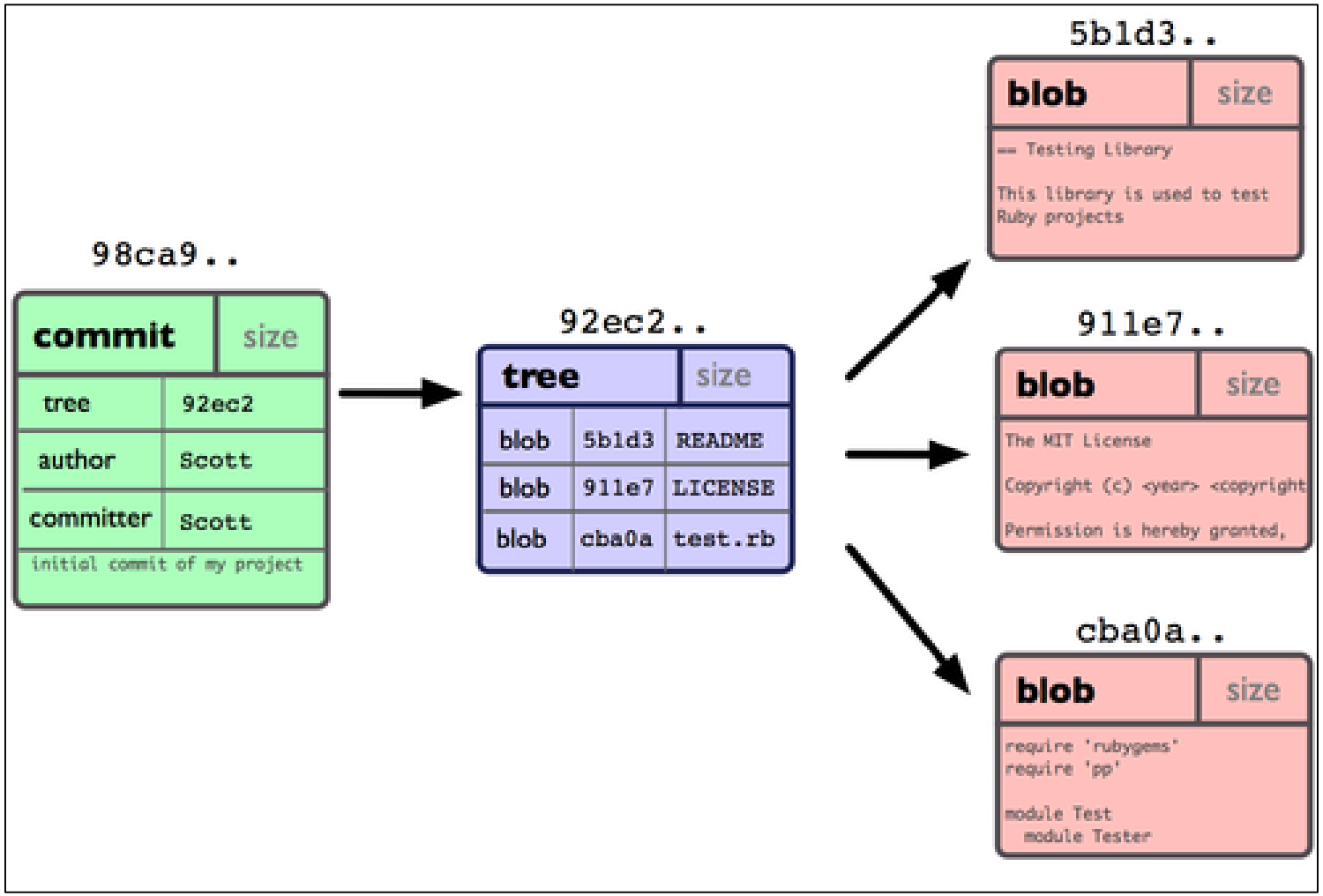
```
$ git add README test.rb LICENSE
$ git commit -m 'initial commit of my project'
```

# GIT

- **Trabajando con ramas en Git**
  - Cuando creas una confirmación con el comando git commit, Git realiza sumas de control de cada subcarpeta (en el ejemplo, solamente tenemos la carpeta principal del proyecto), y guarda en el repositorio Git una copia de cada uno de los archivos contenidos en ella/s.
  - Después, Git crea un objeto de confirmación con los metadatos pertinentes y un apuntador al nodo correspondiente del árbol de proyecto.
  - Esto permitirá poder regenerar posteriormente dicha instantánea cuando sea necesario.
  - En este momento, el repositorio de Git contendrá cinco objetos: un "blob" para cada uno de los tres archivos, un árbol con la lista de contenidos de la carpeta (más sus respectivas relaciones con los "blobs"), y una confirmación de cambios (commit) apuntando a la raiz de ese árbol y conteniendo el resto de metadatos pertinentes.

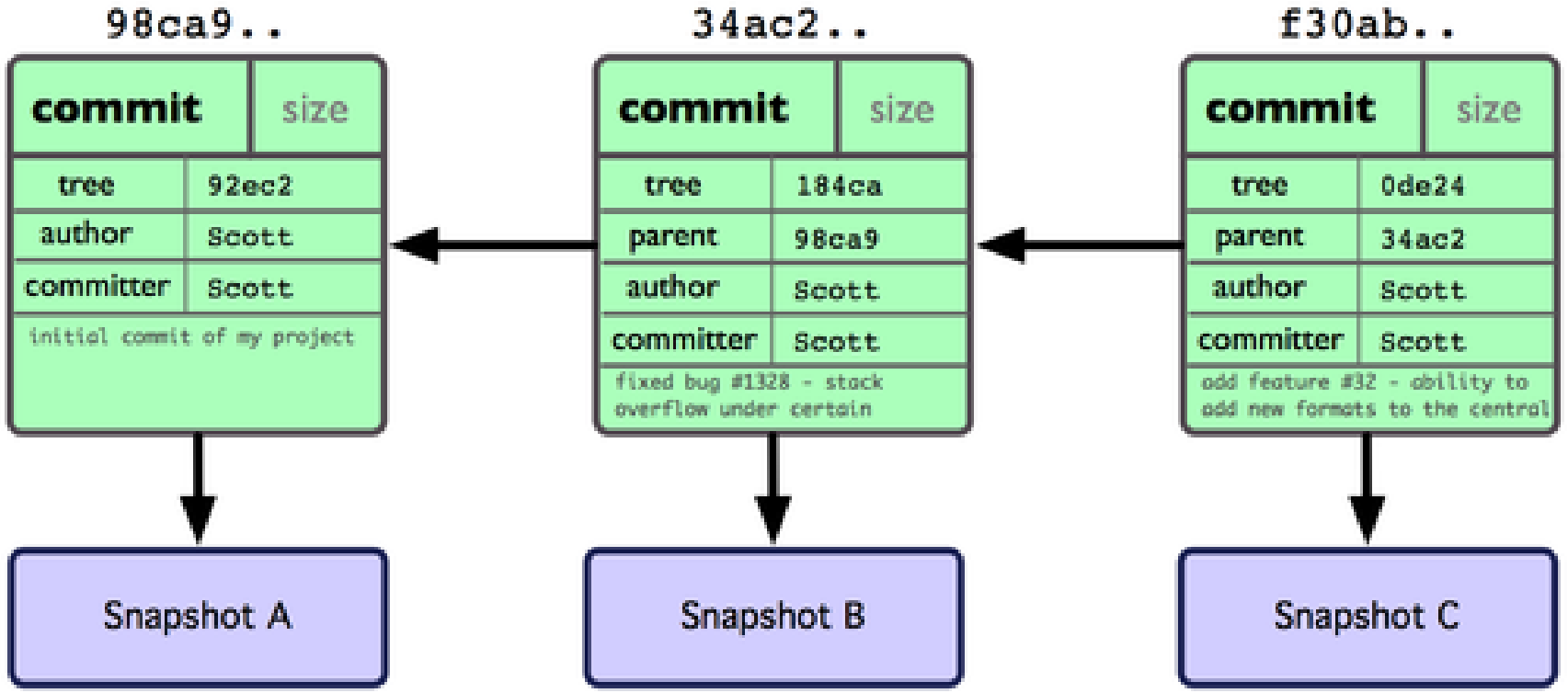
# GIT

- **Trabajando con ramas en Git**



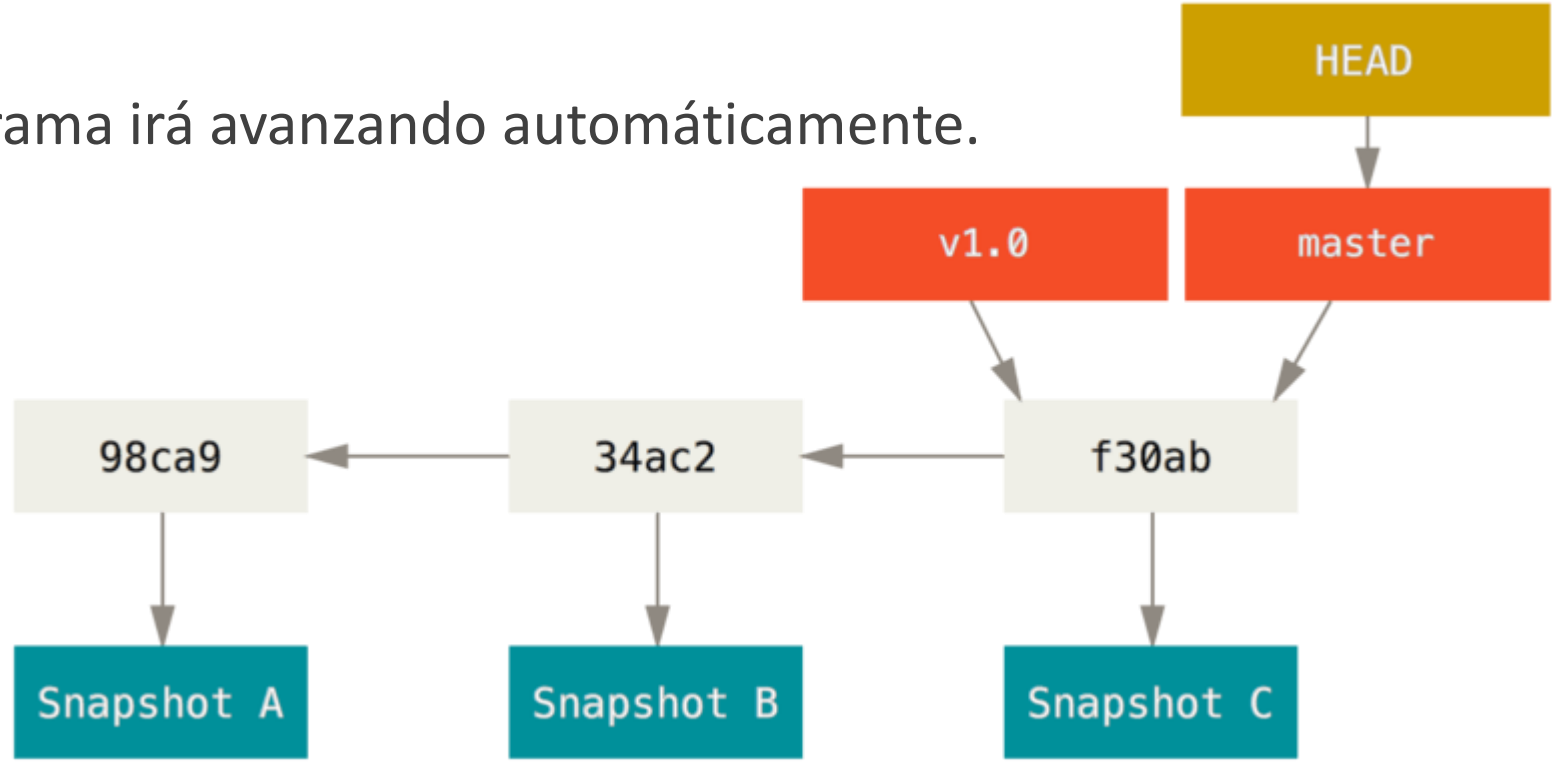
# GIT

- **Trabajando con ramas en Git**
  - Si hacemos más cambios



# GIT

- Una rama Git es simplemente un puntero contra uno de esos commits.
- La rama por defecto de Git es la rama master. Con el primer commit de cambios que realicemos, se creará esta rama master que apunta al último commit realizado.
- En cada commit, la rama irá avanzando automáticamente.



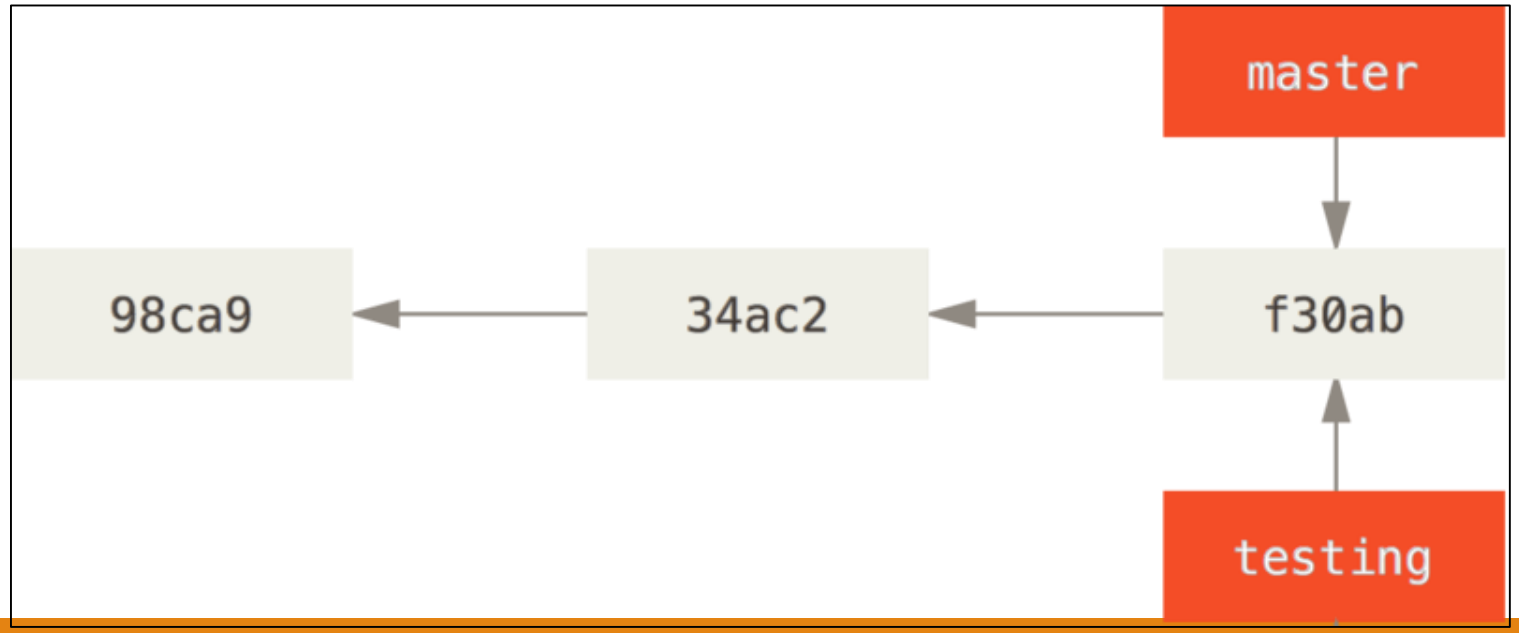
# GIT

## Crear una rama

- Al crear una rama se crea un nuevo puntero con el que podemos trabajar para que lo puedas mover libremente.
- Para crear una nueva rama denominada testing se usa el comando git branch:

```
$ git branch testing
```

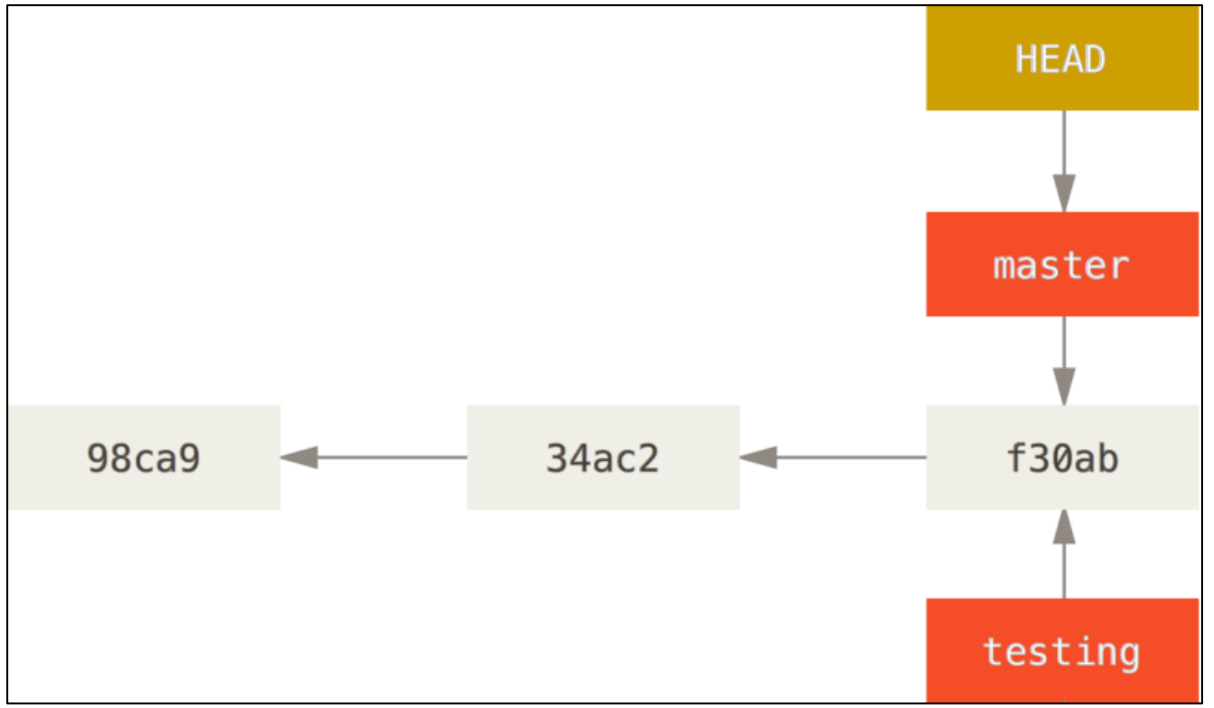
- Esto crea un puntero al mismo commit donde estamos en ese momento.



# GIT

## Trabajar con una rama

- ¿Cómo sabe Git en qué rama estás en este momento?
- Mediante un puntero especial denominado HEAD.
- En Git, es simplemente un apuntador a la rama local en la que estemos situados. En este caso, en la rama master.
- El comando git branch solamente crea una nueva rama, y no salta a dicha rama..
- Podemos ver a donde apuntan las ramas en ese momento




```
$ git log --oneline --decorate
cb87ec9 (HEAD -> master, prueba) Primer commit
```

# GIT

## Saltar a una rama

- Para saltar de una rama a otra, tienes que utilizar el comando git checkout.

```
git checkout testing
```

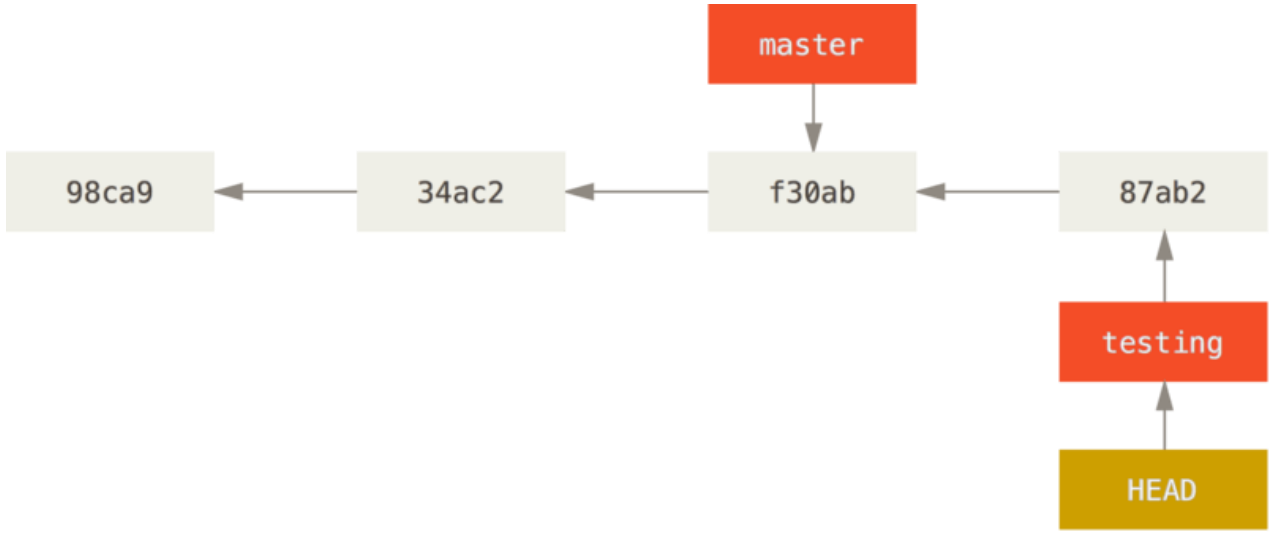
- Esto mueve el apuntador HEAD a la rama testing

# GIT

## Saltar a una rama

- Si ahora hacemos un commit avanzamos por la rama testing, no por la rama master

```
 git log --oneline --decorate
01c703b (HEAD -> testing) commit en testing
cb87ec9 (prueba, master) Primer commit
```



# GIT

## Saltar a una rama

- Si ahora volvemos a la master
- Este comando realiza dos acciones:
  - Mueve el puntero HEAD de nuevo a la rama master, y revierte los archivos del directorio de trabajo, dejándolos tal y como estaban en la última instantánea confirmada de dicha rama master.
  - Esto supone que los cambios que se haga desde este momento en adelante divergerán de la antigua versión del proyecto.
  - Básicamente, lo que se está haciendo es rebobinar el trabajo que habias hecho temporalmente en la rama testing, de tal forma que se pueda avanzar en otra dirección diferente.

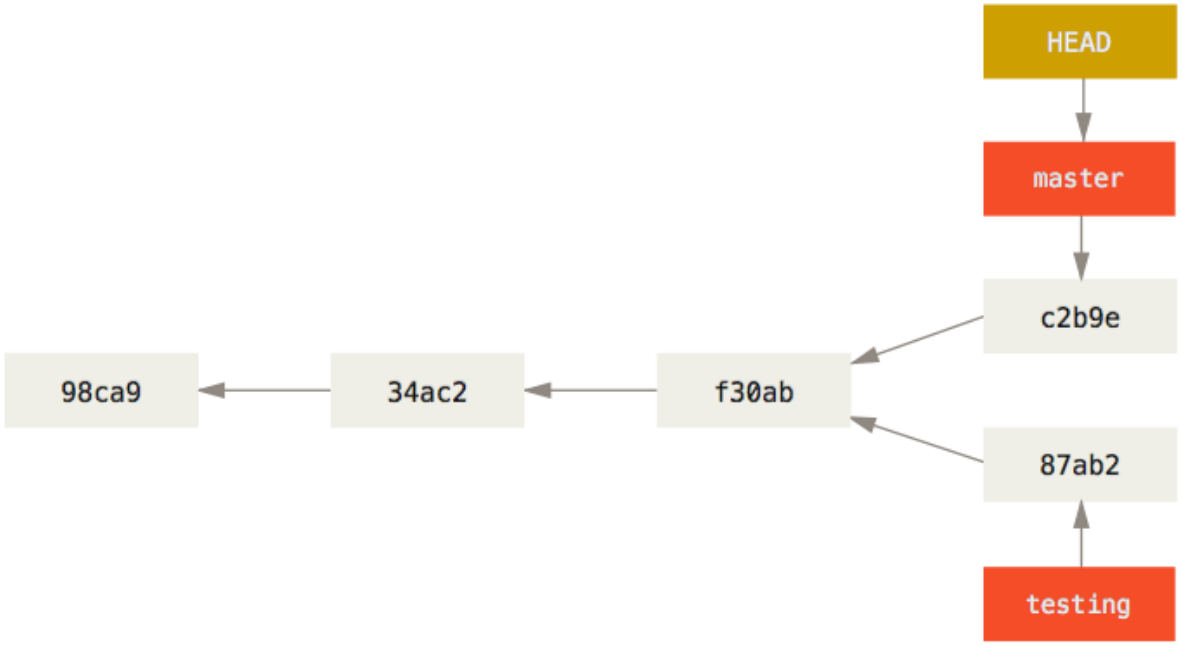
```
$ git checkout master
Switched to branch 'master'

Sergio@Alberto-PC MINGW64 ~/practica (master)
$ git log --oneline --decorate
cb87ec9 (HEAD -> master, prueba) Primer commit
```

# GIT

## Ramas

- Si ahora hacemos un commit en el master
- Esto permite trabajar en dos direcciones, con dos ramas distintas

```
git branch
  master
  prueba
* testing
```



# GIT

## Ramas

- Podemos crear y saltar a una rama con un solo comando

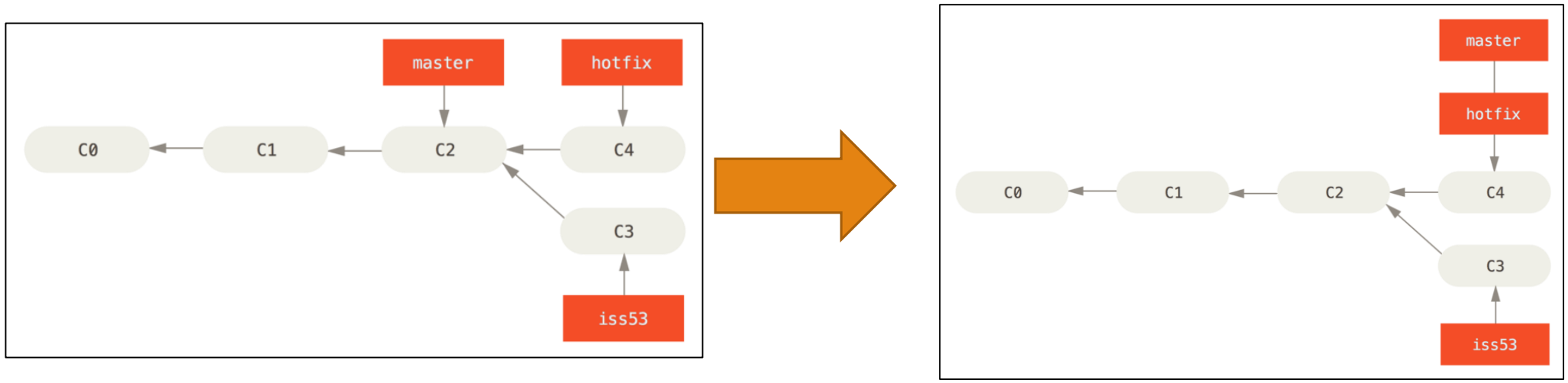
```
$ git checkout -b rama2
Switched to a new branch 'rama2'

$ git branch
  master
  prueba
* rama2
  testing
```

# GIT

## Ramas

- Podemos fusionar ramas con el comando “merge”
- Por ejemplo

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```

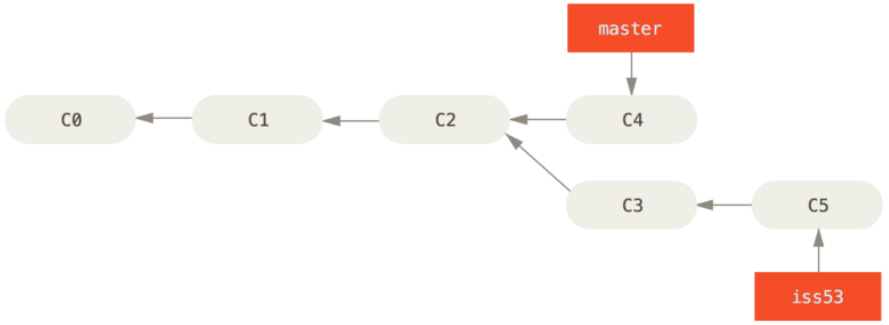


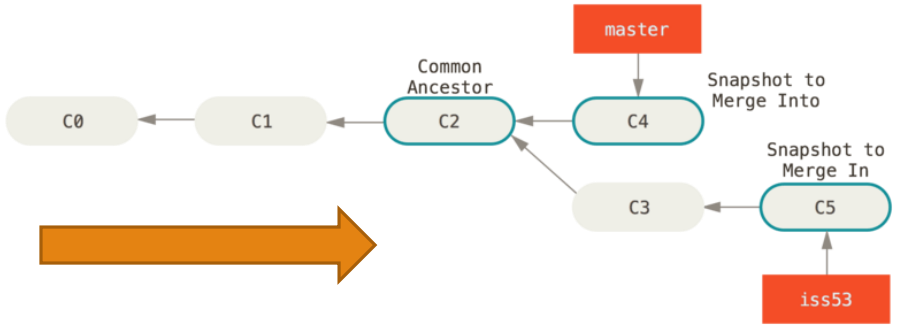
# GIT

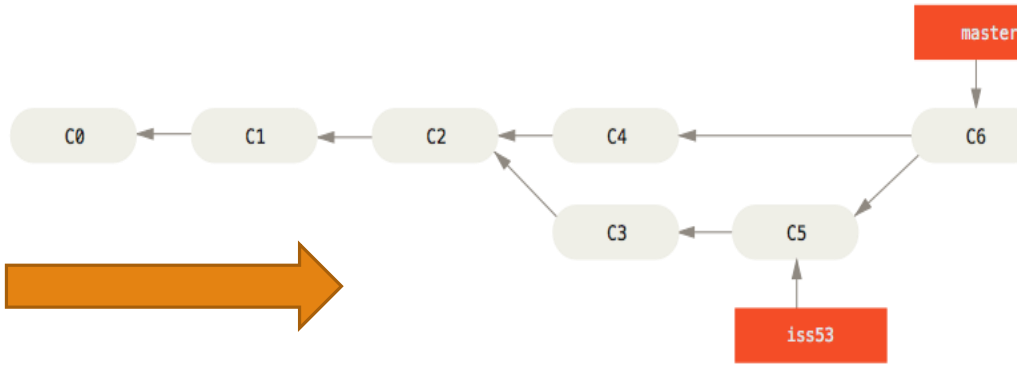
## Ramas

### Otro ejemplo de 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(+)
```



# GIT

## Conflictos en el merge

- En algunas ocasiones, los procesos de fusión no suelen ser correctos.
- Si hay modificaciones dispares en el mismo sitio de un archivo en las dos ramas distintas que queremos fusionar, Git no será capaz de fusionarlas directamente.
- Git avisará con un mensaje del conflicto

```
$ 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

## Conflictos en el merge

- Git no crea automáticamente una nueva fusión confirmada (merge commit).
- Se hace una pausa en el proceso, esperando a que se resuelva el conflicto.
- Para ver qué archivos permanecen sin fusionar en un determinado momento de una fusión, se puede usar el comando git status:

```
[master*]$ git status
index.html: needs merge
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   unmerged:   index.html
#
```

# GIT

## Conflictos en el merge

- Todo aquello que sea conflictivo y no se haya podido resolver, se marca como "sin fusionar" (unmerged).
- Git añade a los archivos conflictivos unas marcas especiales de resolución de conflictos.
- Estas marcas o marcadores nos ayudan cuando abrimos los archivos implicados y los editamos manualmente para corregirlos. Por ejemplo:

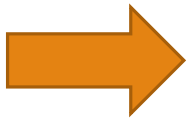
```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=======
<div id="footer">
 please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```

# GIT

## Conflictos en el merge

- Una vez corregido el problema y eliminadas los caracteres <<<<<<< , ======= y >>>>>>> se lanza el comandos git add para marcar cada archivo modificado.
- Marcar archivos como preparados (staging), indica a Git que losconflictos han sido resueltos.

```
<<<<<<< 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
```



```
<div id="footer">
please contact us at email.support@github.com
</div>
```

# GIT

## Conflictos en el merge

- También podemos utilizar otras herramientas.
- Podemos usar el comando git mergetool. Esto arrancará la correspondiente herramienta de visualización podremos 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):
```

# GIT

## Conflictos en el merge

- Podemos usar de nuevo git status para comprobar que el conflicto se ha resuelto
- Volvemos a pulsar git commit para terminar el trabajo de MERGE que había quedado colgado

```
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
```

# GIT

- **Práctica**
  - Ponernos en la rama MASTER
  - Crear un nuevo fichero denominado prueba_merge.txt
  - Poner una línea de texto
  - Crear una nueva rama denominada “parche2”
  - Modificar el fichero prueba_merge.txt y le ´modificamos la línea
  - Volvemos a la rama MASTER
  - Modificamos la misma línea del fichero
  - Intentamos hacer un merge
  - Solucionamos el conflicto

# GIT

- **Ramas**
  - Podemos borrar una rama con “git branch –d rama”
  - Por ejemplo:

```
$ git branch -d prueba
Deleted branch prueba (was cb87ec9).

$ git branch
  master
* rama2
  testing
```

# GIT

## Ramas

- Con git Branch –v podemos encontrar más información de las ramas

```
$ git branch -v
  master   074e41b commit en branch de nuevo
* parche2  15091ef en parche2
  rama2    01c703b commit en testing
  testing  01c703b commit en testing
```

- Si quiero conocer las que han sido fusionadas

```
$ git branch --merged
    master
* parche2
```