Sistemas de control de versiones.

Entornos de desarrollo

ÍNDICE DE CONTENIDOS

- Alias
- Ramas.
- Repositorio remoto.
- Apuntadores
- Ramas

Git: Alias

Git no deduce automáticamente tu comando si lo tecleas parcialmente. Si no quieres teclear el nombre completo de cada comando de Git, puedes establecer fácilmente un alias para cada comando mediante **git config**, ejemplo:

- $ git config --global alias.co checkout
- $ git config --global alias.br branch
- $ git config --global alias.ci commit
- $ git config --global alias.st status

Esto es util para “crear” nuevos comandos, por ejemplo, si queremos quitar un archivo del área de prepación:

- $ git config --global alias.unstage 'reset HEAD --'

Git: Alias

Podemos añadir nuestros alias al fichero .gitconfig

```
[alias]
b = branch
ci = commit
…
```

Para deshacer un alias, usamos la siguiente sentencia:

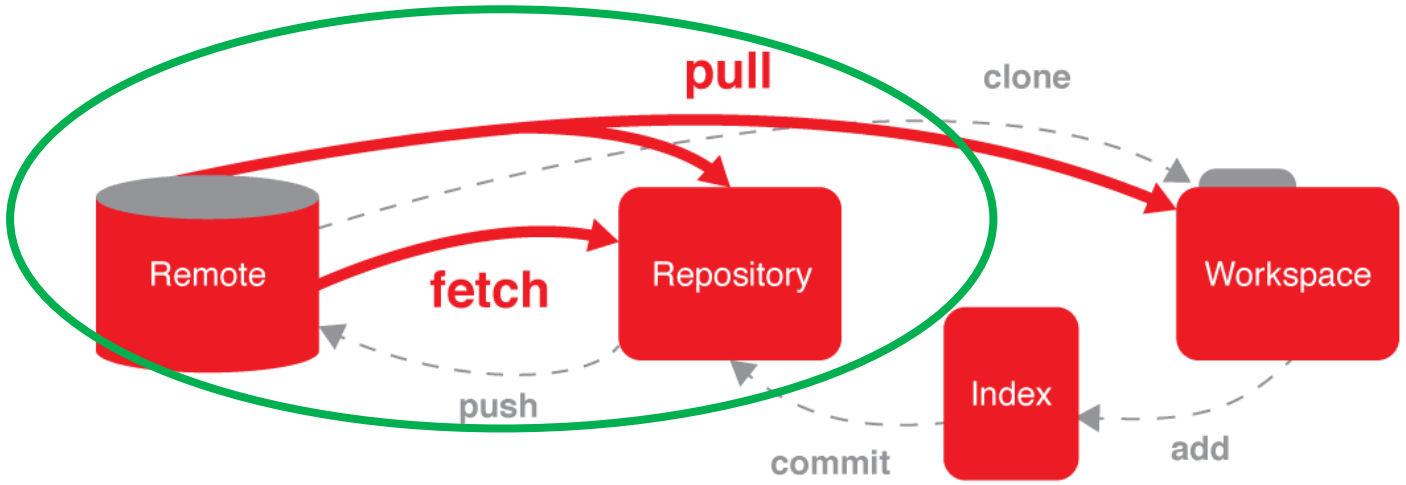
- **$git config --global --unset alias.TUALIAS**

Git: Ramas

Cada desarrollador o equipo de desarrollo puede hacer uso de Git de la forma que le parezca mas conveniente. Sin embargo una buena práctica aceptada por los equipos de desarrollo es mantener cuatro ramas: **master, development, features, y hotfix**.

- **Master**: Es la rama principal. Contiene el repositorio que se encuentra publicado en producción, por lo que debe estar siempre estable.
- **Development**: Es la rama de integración, todas las nuevas funcionalidades se deben integrar en esta rama. Después de que se corrijan los errores la rama master y development se pueden fusionar.
- **Features**: Nuevas funcionalidades que se añaden al proyecto. Una vez que la funcionalidad esté desarrollada, se fusiona con la rama development, donde se integrará con las demás funcionalidades.
- **Hotfix**: Bugs que surgen en producción, por lo que se deben arreglar y publicar de forma urgente. Es una rama derivada de master. Una vez corregido el error, se debe debe fusionar con la rama master; Además, para que no quede desactualizada, se debe realizar la fusión con development.

Repositorio remoto.



Repositorio remoto.

- Los **repositorios remotos** contienen tu proyecto alojado en Internet.
- Puedes tener varios, cada uno de los cuales puede ser de *sólo lectura, o de lectura/escritura*, según los **permisos** que tengas.
- Colaborar con otros implica gestionar estos repositorios remotos, y mandar (**push**) y recibir (**pull**) datos de ellos cuando necesites compartir cosas.

Git: remote

Para ver qué repositorios remotos tienes configurados, dentro de un repositorio git(exista la carpeta .git), puedes ejecutar el comando **git remote**.

Mostrará una lista con los nombres de los remotos que hayas especificado. Si has clonado tu repositorio, deberías ver por lo menos **origin**, nombre por defecto del servidor clonado.

```
$ git remote
origin
```

Git: remote

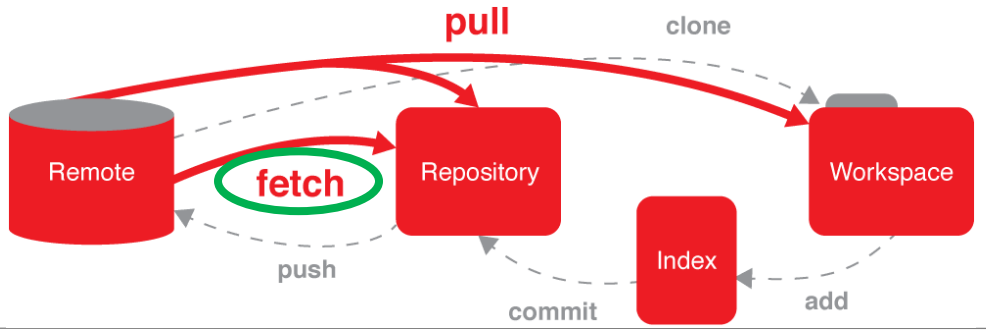
Para añadir un nuevo repositorio GIT remoto, asignándole un nombre con el que referenciarlo fácilmente, ejecuta **git remote add [nombre]**

```
$ git remote add ch https://github.com/aspittel/coding-cheat-sheets.git

                MINGW64 /d/Git/workspace (master)
$ git remote
ch
origin

                MINGW64 /d/Git/workspace (master)
$ git remote -v
ch      https://github.com/aspittel/coding-cheat-sheets.git (fetch)
ch      https://github.com/aspittel/coding-cheat-sheets.git (push)
origin  file:///C:\Users\fran\.git (fetch)
origin  file:///C:\Users\fran\.git (push)
```

Git: fetch



A partir del ejemplo anterior, podemos usar la cadena **ch** en consola, en lugar de la URL.

Por ejemplo, si queremos recuperar toda la información existente en el repositorio remoto, podemos ejecutar: **git fetch ch**

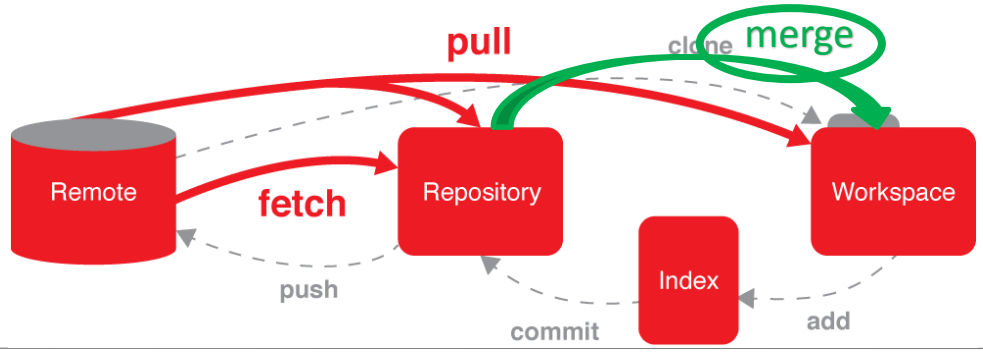
```
$ git fetch ch
warning: no common commits
remote: Enumerating objects: 351, done.
remote: Total 351 (delta 0), reused 0 (delta 0), pack-reused 351
Receiving objects: 100% (351/351), 378.64 KiB | 1.35 MiB/s, done.
Resolving deltas: 100% (181/181), done.
From https://github.com/aspittel/coding-cheat-sheets
 * [new branch]      master     -> ch/master
```

En otras palabras, **git fetch [repositorio_remoto]**, nos permite recuperar todos los datos del proyecto remoto que no tengamos.

git fetch origin recupera toda la información enviada al servidor desde que lo clonamos.

¡**fetch** sólo recupera la información y la pone en tu repositorio local!

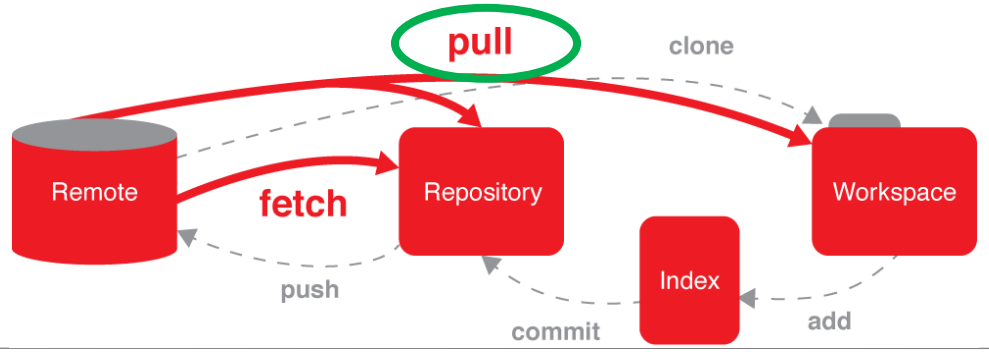
Git: merge



Si fetch trae toda la información del repositorio remoto, **git merge** trae la lleva nuestro espacio de trabajo, es decir, trae toda la información de nuestro repositorio local a nuestro espacio de trabajo.

```
$ git merge
Already up to date.
```

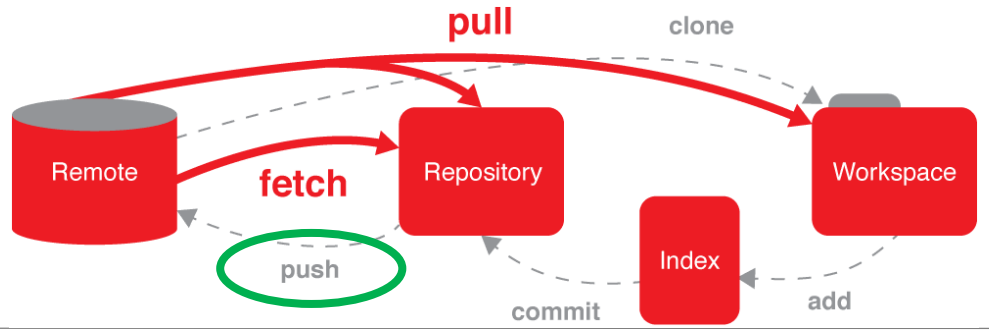
Git: pull



Otra opción, es traer directamente todos los cambios del repositorio remoto a nuestro espacio trabajo. Para ello utilizamos **git pull = git fetch + git merge**

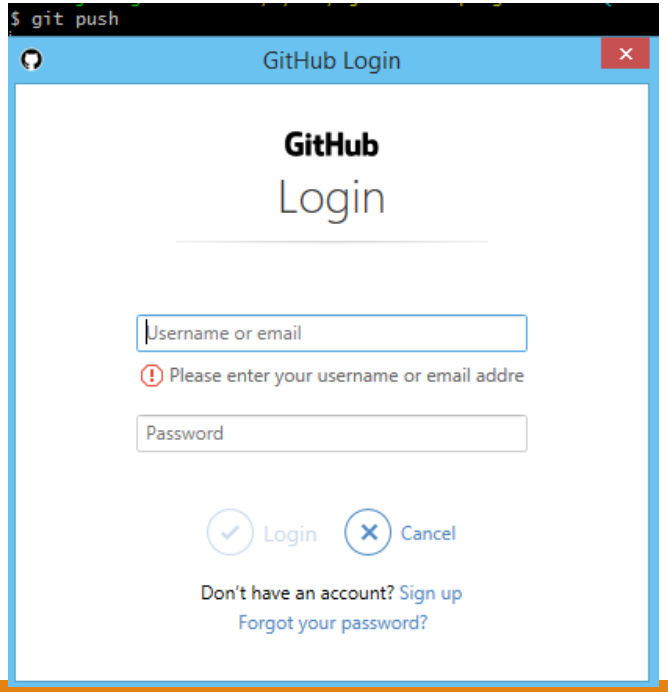
```
$ git pull
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 2 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (2/2), done.
From file:///C:\Users\fran\
   1a4e77f..ac40021  master     -> origin/master
```

Git: push



El proceso contrario a **fetch**, mandar la información de nuestro repositorio local al remoto, se realiza utilizano el comando **git push [nombre_remoto][nombre_rama]**.

En caso que no lo estéis, tendréis que logearos en GitHub para poder realizar el push.



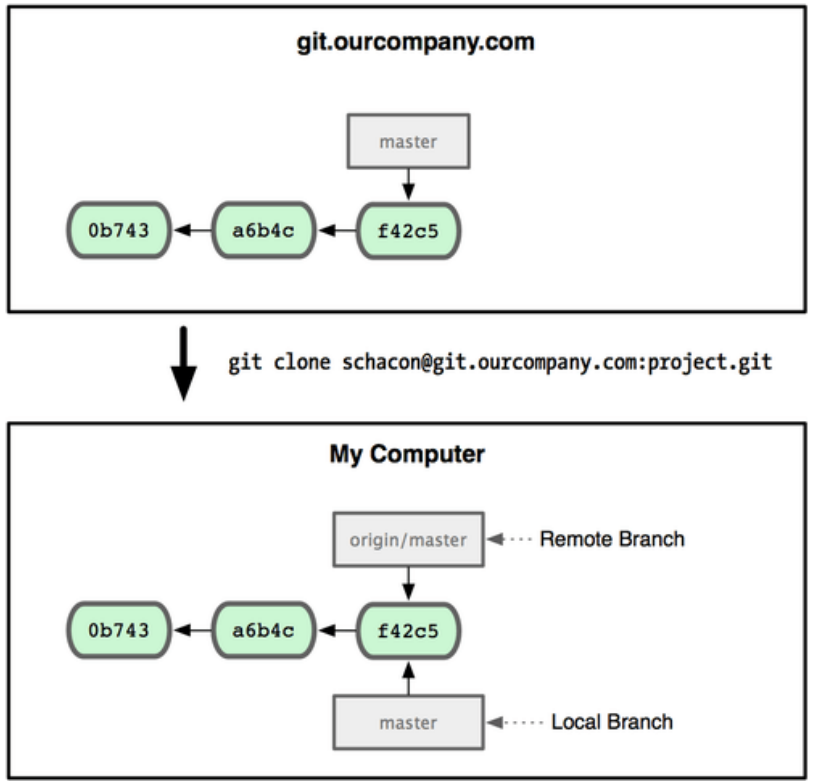

```
$ git push
Everything up-to-date
```

Git: Apuntadores

Supongamos que tenemos el servidor Git en red git.ourcompany.com.

Si hacemos un clón , Git automáticamente lo denominará **origin**, traerá (**pull**) sus datos, creará un apuntador hacia donde esté en ese momento su rama master, denominará la copia local **origin/master**; y será inamovible para ti.

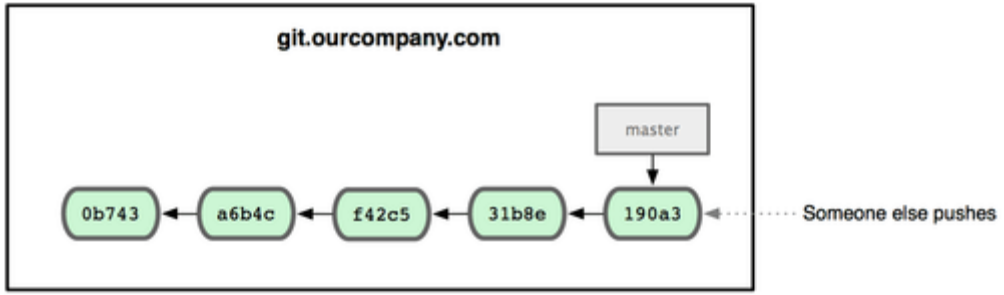
Git nos proporcionará también nuestra propia rama master, apuntando al mismo lugar que la rama **master de origin**; siendo en esta última donde podemos trabajar.

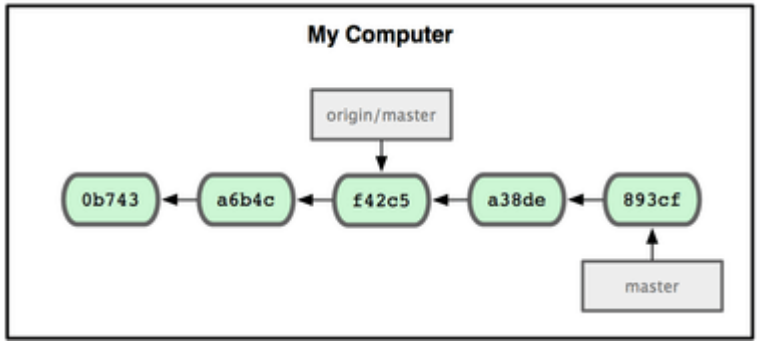


Git: Apuntadores

Un clón Git te proporciona tu propia rama **master** y otra rama **origin/master** apuntando a la rama master original.

Si haces algún trabajo en tu rama **master local**, y al mismo tiempo, alguna otra persona lleva (**push**) su trabajo al servidor *git.ourcompany.com*, actualizando la **rama master** de allí, te encontrarás con que ambos registros avanzan de forma diferente. Además, mientras no tengas contacto con el servidor, tu apuntador a tu rama **origin/master** no se moverá.





Git: tag

Git tiene la habilidad de etiquetar (**tag**) puntos específicos en la historia como importantes. Generalmente se usa esta funcionalidad para marcar puntos donde se ha lanzado alguna versión (v1.0, v2.0,…).

Para ver las etiquetas disponibles, usamos **git tag:**

```
$ git tag
v1.1
```

Este comando lista las etiquetas en orden alfabético; el orden en el que aparecen no es realmente importante.

Git: tag

Crear una etiqueta Git, usamos la opción –a de la siguiente forma:

```
$ git tag -a v13.0 -m 'la version perfecta'

fran@fjcanogra MINGW64 /d/Git/workspace (master)
$ git tag
v13.0
```

Git: tag

Para ver los cambios que contienen la versión 13, anterior, usamos **git show [etiqueta]** :

```
$ git show v13.0
tag v13.0
Tagger: fjcanogra <55139603+fjcanogra@users.noreply.github.com>
Date:

la version perfecta

commit ebd53cd33782c758300feb59aeee03e1c093517a (HEAD -> master, tag: v13.0)
Author: fjcanogra <55139603+fjcanogra@users.noreply.github.com>
Date:

    subiendo

diff --git a/Ejercicios/hellowWorld/src/hellowWorld/Antonio.java b/Ejercicios/he
llowWorld/src/hellowWorld/Antonio.java
new file mode 100644
index 0000000..a24279e
--- /dev/null
+++ b/Ejercicios/hellowWorld/src/hellowWorld/Antonio.java
@@ -0,0 +1,5 @@
+package hellowWorld;
```

Git: tag

Las etiquetas no son enviadas al servidor remoto, por lo que tenemos que enviarlas. Para ello usamos **git push origin [etiqueta]**

```
$ git push origin v13.0
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (8/8), 662 bytes | 73.00 KiB/s, done.
Total 8 (delta 0), reused 0 (delta 0)
```

Ramas

Git almacena sus datos como una serie de instantáneas (copias puntuales de los archivos completos, tal y como se encuentran en ese momento).

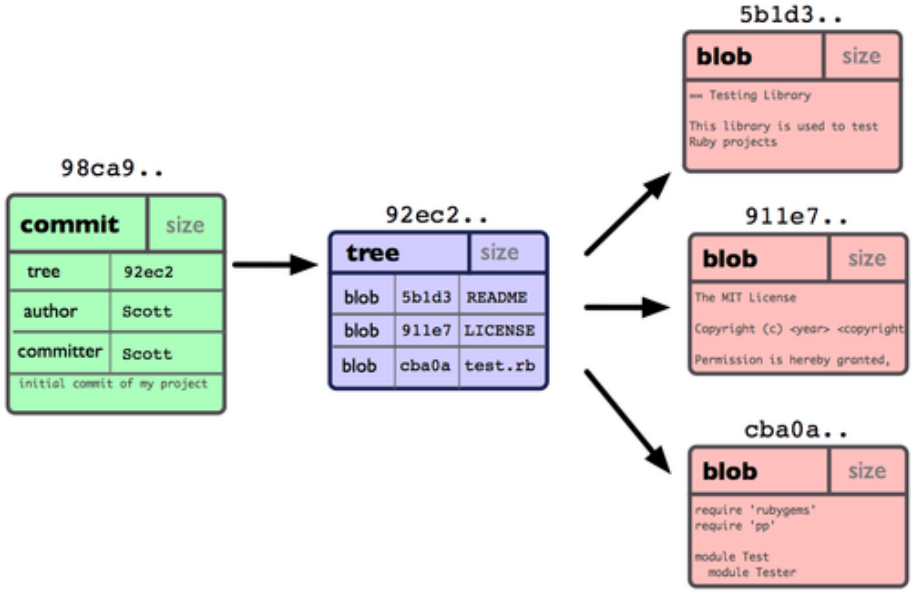
En cada confirmación de cambios (commit), Git almacena un punto de control que conserva:

- apuntador a la copia puntual de los contenidos preparados (staged)
- 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).

Ramas

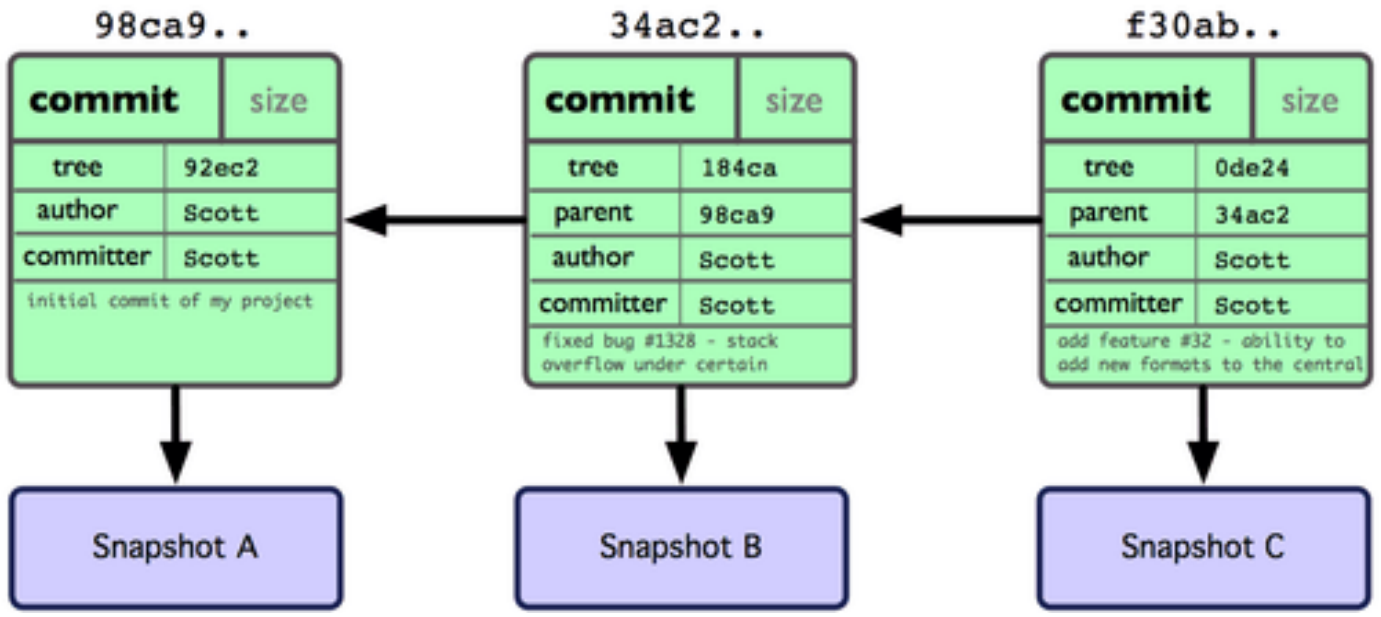
Cuando creas una confirmación con el comando git commit, Git realiza sumas de control de cada subcarpeta (en el ejemplo, solamente tenemos la carpeta principal del proyecto), y las guarda como **objetos árbol** en el repositorio Git. Después, Git crea un **objeto de confirmación** con los metadatos pertinentes y un **apuntador al objeto árbol** raíz del proyecto. Esto permitirá poder regenerar posteriormente dicha instantánea cuando sea necesario.

Se generan una commit con la información subida, con un archivo tree que controlan todos los blos's subidos y los blob's que son los archivos subidos.



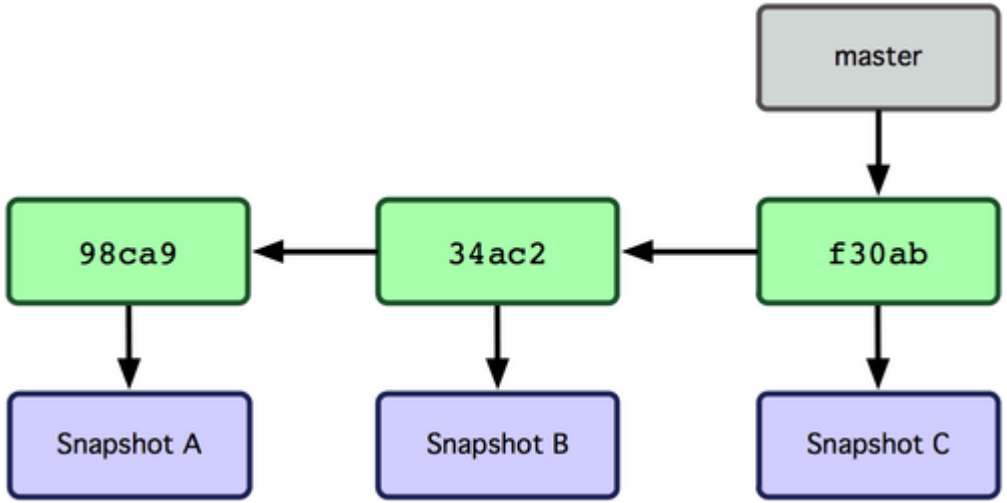
Ramas

Si haces más cambios y vuelves a confirmar, la siguiente confirmación guardará un apuntador a esta su confirmación precedente. Tras un par de confirmaciones más:



Ramas

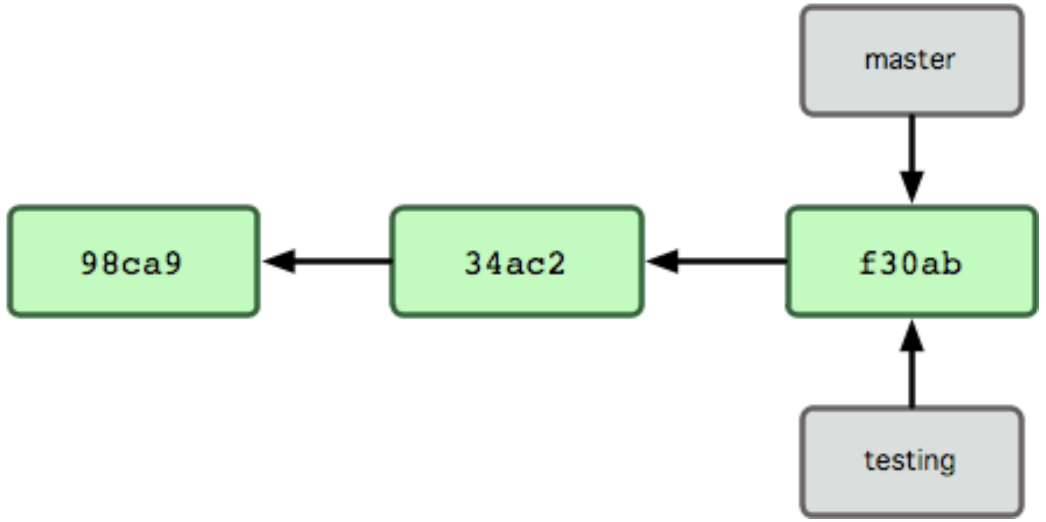
- Una **rama** Git es un apuntador móvil apuntando a una de esas confirmaciones.
- La rama por defecto de Git es la rama **master**. La primera confirmación de cambios que realicemos, se creará esta rama principal master apuntando a dicha confirmación.
- En cada confirmación de cambios que realicemos, la rama irá avanzando automáticamente. Y apuntará siempre a la última confirmación realizada.



Ramas

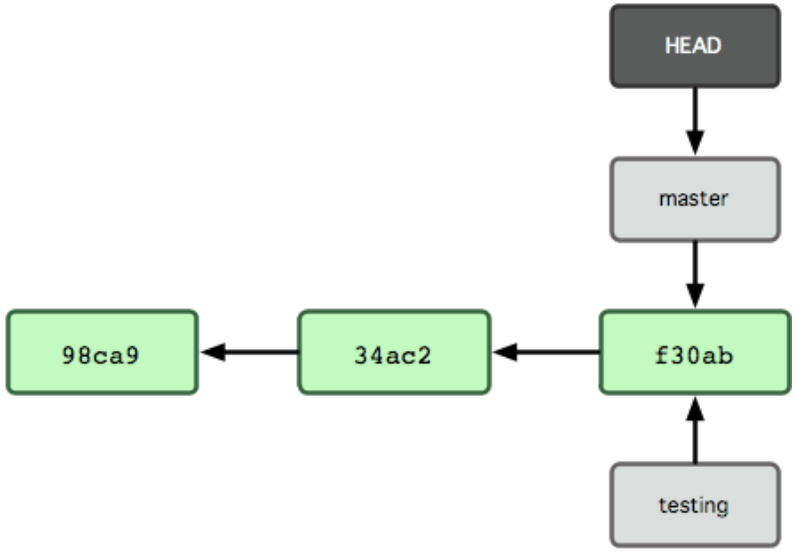
¿Qué sucede cuando creas una nueva rama?

Se crea un **nuevo apuntador** para que lo puedas mover libremente. Para crear una nueva rama, utilizamos el comando **git branch [nombre rama]**.Por ejemplo, si queremos crear la rama "testing“:



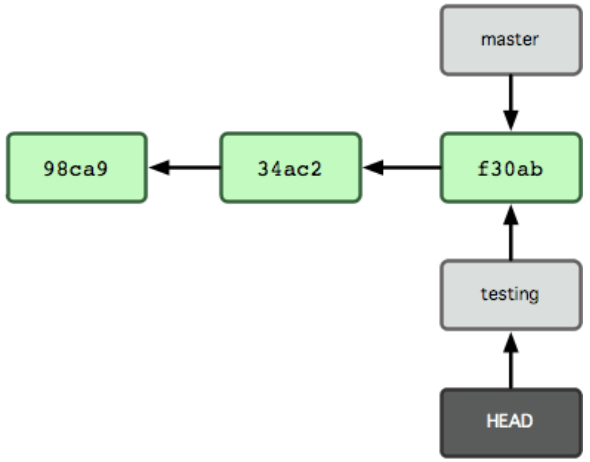
Ramas

Para saber en que rama nos encontramos en cada momento, git tiene un apuntador especial llamado **HEAD.**



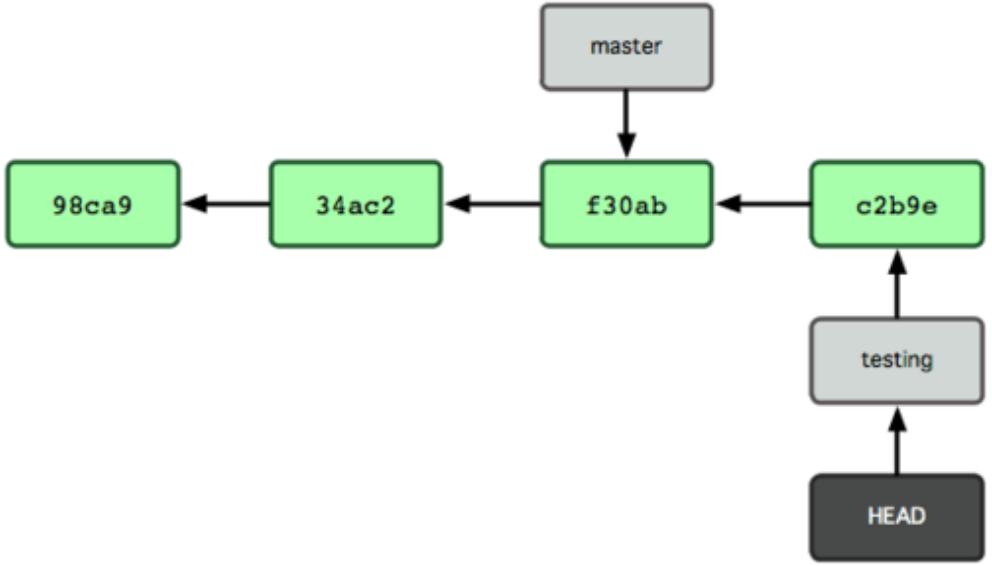
Ramas

Para movernos de una rama a otra, utilizamos el comando **git checkout [nombreRama]**.



Ramas

Si hacemos un commit en la rama testing, el árbol quedaría:



Ramas

Si movemos el apuntador HEAD a la rama master y subimos nuevos cambios. Esto supone que los cambios que hagas desde este momento en adelante divergirán de la antigua versión del proyecto.