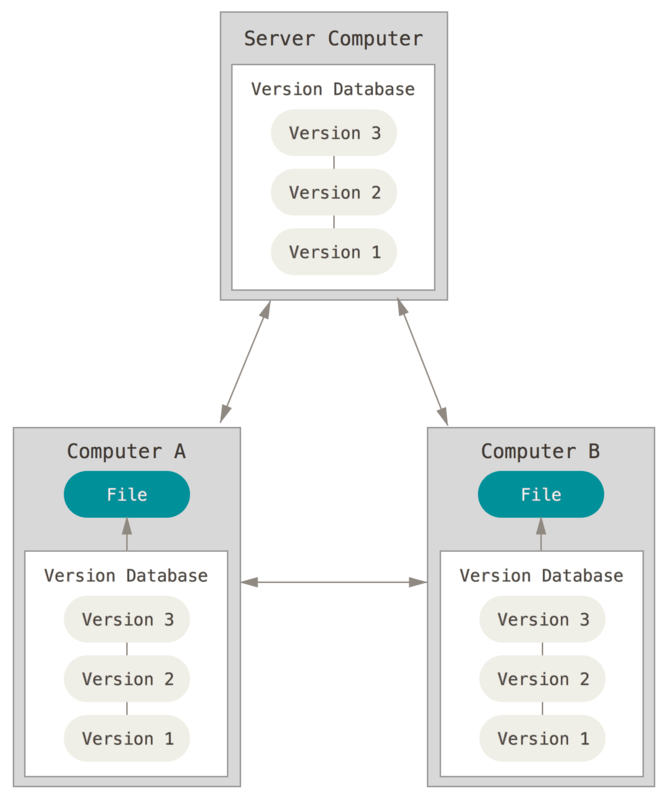
# GIT intermedio

## ¿Qué es un control de versiones?



Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante. Dicho sistema te permite regresar a versiones anteriores de tus archivos, regresar a una versión anterior del proyecto completo, comparar cambios a lo largo del tiempo, ver quién modificó por última vez algo que pueda estar causando problemas, ver quién introdujo un problema y cuándo, y mucho más. Usar un VCS también significa generalmente que si arruinas o pierdes archivos, será posible recuperarlos fácilmente.

Ver más en: [Sobre el Control de Versiones - Acerca del Control de Versiones](https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones#ch01-introduction)

## [Comandos básicos:](http://rogerdudler.github.io/git-guide/index.es.html)

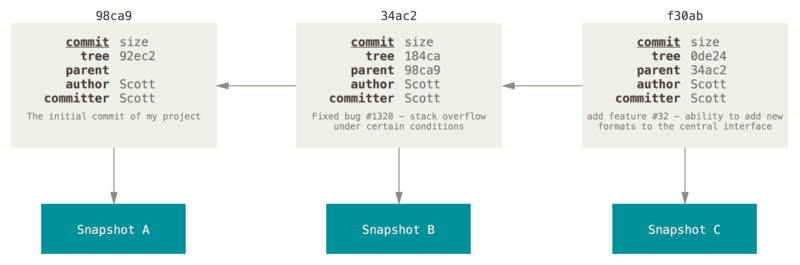
* **git branch** (Rama) para crear o manipular ramas.
* **git checkout** (Mover o salir) Revertir cambios y cambiar entre ramas
* **git commit** (Confirmar) añade archivos al repositorio local desde el Staging
* **git fetch** (Ir a buscar) Revisa si hay cambios en el repositorio remoto
* **git merge** (Mezclar) Mezcla los cambios de una rama indicada a la rama actual
* **git pull** (Halar) trae archivos desde el repositorio remoto al local, es igual a hacer fetch + merge
* **git push** (Empujar) añade archivos al repositorio remoto desde el repositorio local

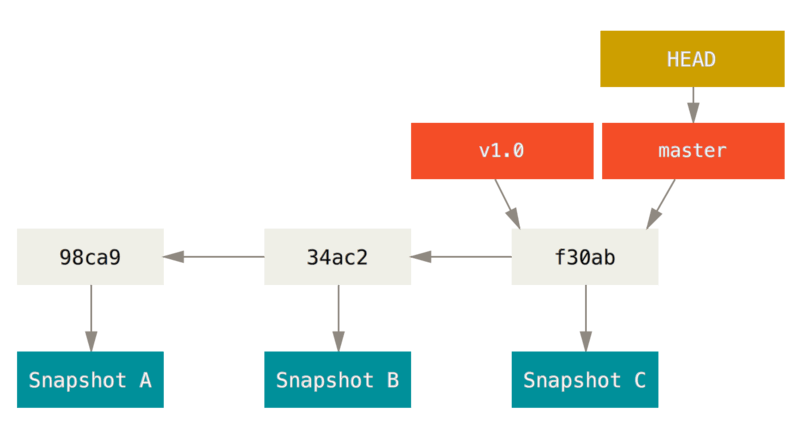
## ¿Qué es una rama?

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. Git no almacena los datos de forma incremental (guardando sólo diferencias), sino que los almacena como una serie de instantáneas o snapshots (copias puntuales de los archivos completos, tal y como se encuentran en ese momento).

En cada confirmación de cambios (**commit**), Git almacena una instantánea de tu trabajo preparado. Dicha instantánea contiene además unos metadatos con el autor y el mensaje explicativo, y uno o varios apuntadores a las confirmaciones (commit) que sean padres directos de esta.

Cuando creas una confirmación con el comando git commit, Git realiza sumas de control de cada subdirectorio 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. Si haces más cambios y vuelves a confirmar, la siguiente confirmación guardará un apuntador a su confirmación precedente.





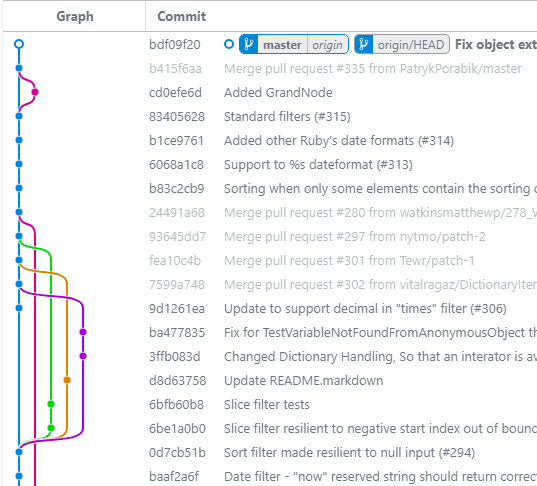
Una rama Git es simplemente un apuntador móvil apuntando a una de esas confirmaciones. La rama por defecto de Git es la rama **master**. Con la primera confirmación de cambios que realicemos, se creará esta rama principal **master** apuntando a dicha confirmación. En cada confirmación de cambios que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

## Git: los conceptos de "master", "origin" y "HEAD"

### master

Dicho de manera rápida y básica, una rama no es más que un nombre que se da a un commit, a partir del cual se empieza a trabajar de manera independiente y con el que se van a enlazar nuevos commits de esa misma rama. Las ramas pueden mezclarse de modo que todo el trabajo hecho en una de ellas pase a formar parte de otra.

Existe una rama predeterminada que se crea automáticamente cuando se crea un repositorio que se llama rama master.

Por regla general a master se la considera la rama principal y la raíz de la mayoría de las demás ramas. Lo más habitual es que en master se encuentre el "código definitivo", que luego va a producción, y es la rama en la que se mezclan todas las demás tarde o temprano para dar por finalizada una tarea e incorporarla al producto final.

### origin

Git es un sistema de control de código distribuido. Esto quiere decir que, aunque todos los desarrolladores tienen una copia exacta del mismo repositorio en su disco duro, existen uno o más repositorios remotos contra los que trabajamos. Estos repositorios remotos se suelen llamar simplemente "remotos".

“origin” es simplemente el nombre predeterminado que recibe el repositorio remoto principal contra el que trabajamos. Cuando clonamos un repositorio por primera vez desde GitHub o cualquier otro sistema remoto, el nombre que se le da a ese repositorio "maestro" es precisamente origin.

### HEAD

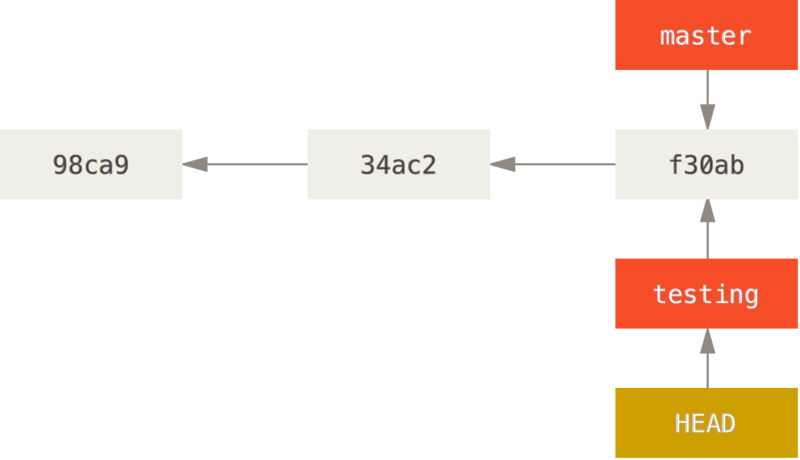
Se refiere al commit en el que está tu repositorio posicionado en cada momento. Por regla general HEAD suele coincidir con el último commit de la rama en la que estés, ya que habitualmente estás trabajando en lo último. Pero si te mueves hacia cualquier otro commit anterior entonces el HEAD estará más atrás.

Ver más en [Git: los conceptos de "master", "origin" y "HEAD"](https://www.campusmvp.es/recursos/post/git-los-conceptos-de-master-origin-y-head.aspx)

### Crear una Rama Nueva

¿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:

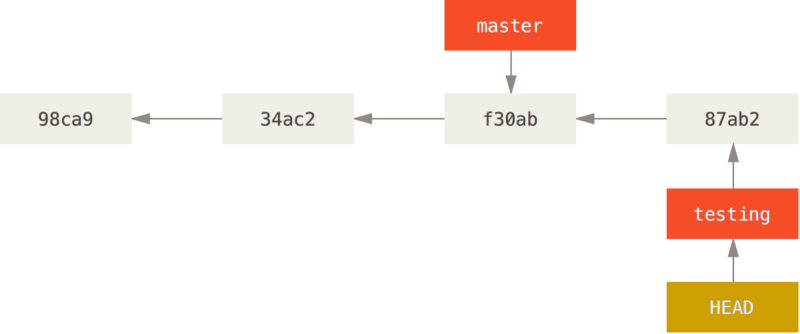
$ git branch testing

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

### Cambiar de Rama

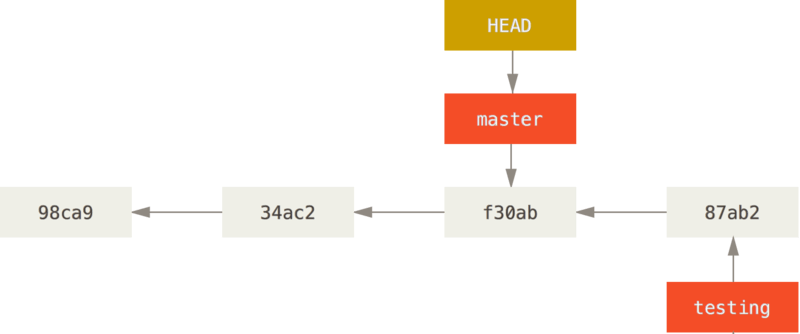
Para saltar de una rama a otra, tienes que utilizar el comando git **checkout**. Hagamos una prueba, saltando a la rama testing recién creada:

$ git checkout testing



Esto mueve el apuntador **HEAD** a la rama testing.

¿Cuál es el significado de todo esto? Bueno…​, lo veremos tras realizar otra confirmación de cambios: La rama apuntada por HEAD avanza con cada confirmación de cambios

Observamos algo interesante: la rama testing avanza, mientras que la rama master permanece en la confirmación donde estaba cuando lanzaste el comando git checkout para saltar. Volvamos ahora a la rama master:

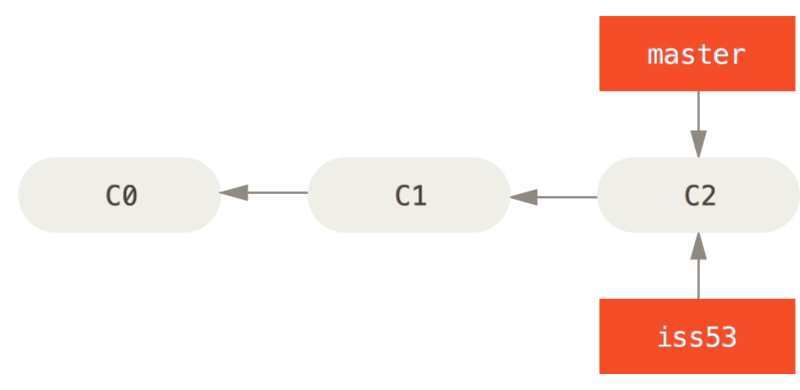
$ git checkout master

Ver más en: [Ramificaciones en Git - ¿Qué es una rama?](https://git-scm.com/book/es/v2/Ramificaciones-en-Git-%C2%BFQu%C3%A9-es-una-rama%3F)

## Procedimientos Básicos para Ramificar

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

Decides trabajar en el problema #53, según el sistema que tu compañía utiliza para llevar el seguimiento de los problemas. Para crear una nueva rama y saltar a ella

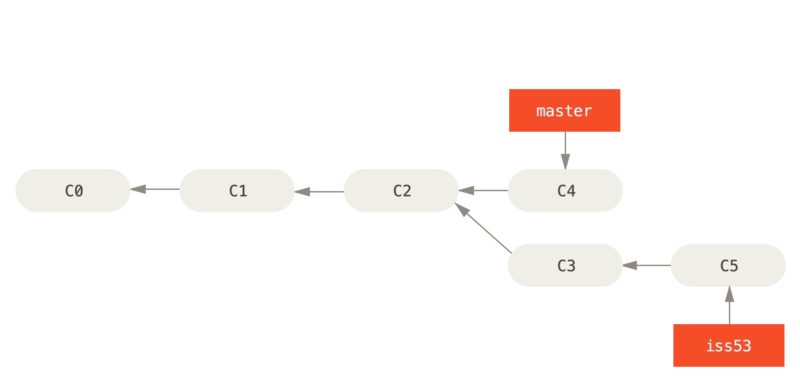


$ git checkout -b iss53

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 (checked out) en este momento (es decir, a la que apunta HEAD):

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

Se podría haber trabajado independientemente en las dos ramas haciendo confirmaciones en las dos, los apuntadores avanzan independientemente.

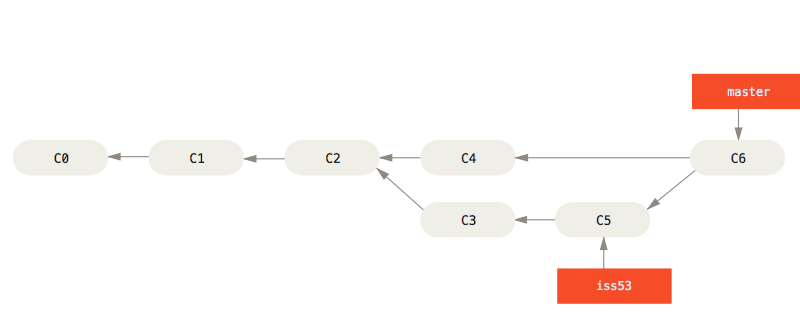
## 

## Procedimientos Básicos de Fusión

Supongamos que tu trabajo con el problema #53 ya está completo y listo para fusionarlo (merge) con la rama master. Para ello, vas a fusionar la rama iss53. Simplemente, activa (checkout) la rama donde deseas fusionar y lanza el comando git merge:

$ git checkout master

$ git merge iss53

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.

Ahora que todo tu trabajo ya está fusionado con la rama principal, no tienes necesidad de la rama iss53. Por lo que puedes borrarla.

$ 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.

$ git merge iss53

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.

Ver más en:

* [Procedimientos Básicos para Ramificar y Fusionar](https://git-scm.com/book/es/v2/Ramificaciones-en-Git-Procedimientos-B%C3%A1sicos-para-Ramificar-y-Fusionar)
* [Curso Git - Ramas y Fusiones](https://www.youtube.com/watch?v=-UQulO_0prM&feature=emb_logo)

## Ramas Remotas

Son referencias al estado de las ramas en tus repositorios remotos. Son ramas locales que no puedes mover; se mueven automáticamente cuando estableces comunicaciones en la red.

Suelen referenciarse como (remoto)/(rama). Por ejemplo, si quieres saber cómo estaba la rama master en el remoto origin, puedes revisar la rama origin/master. O si estás trabajando en un problema con un compañero y este envía (push) una rama iss53, tú tendrás tu propia rama de trabajo local iss53; pero la rama en el servidor apuntará a la última confirmación (commit) en la rama origin/iss53.

Supongamos que tienes un servidor Git en tu red, en git.ourcompany.com. Si haces un clon desde ahí, Git automáticamente lo denominará origin, traerá (pull) sus datos, creará un apuntador hacia donde esté en ese momento su rama master y denominará la copia local origin/master. Git te proporcionará también tu propia rama master, apuntando al mismo lugar que la rama master de origin; de manera que tengas donde trabajar.

Si haces algún trabajo en tu rama master local, y al mismo tiempo, alguien más 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.

### Traer y Fusionar

$ git fetch origin

Para sincronizarte, puedes utilizar el comando git fetch origin. Este comando localiza en qué servidor está el origen (en este caso git.ourcompany.com), recupera cualquier dato presente allí que tú no tengas, y actualiza tu base de datos local, moviendo tu rama origin/master para que apunte a la posición más reciente.

A pesar de que el comando git fetch trae todos los cambios que no tienes del servidor, este no modifica tu directorio de trabajo. Simplemente obtendrá los datos y dejará que tú mismo los fusiones.

$ git merge origin/master

Sin embargo, existe un comando llamado git pull, el cuál básicamente hace git fetch seguido por git merge en la mayoría de los casos.

$ git pull origin/master

Si tienes una rama de seguimiento configurada, git pull identificará a qué servidor y rama remota sigue tu rama actual, traerá los datos de dicho servidor e intentará fusionar dicha rama remota.

### Publicar

Cuando quieres compartir una rama con el resto del mundo, debes llevarla (push) a un remoto donde tengas permisos de escritura. Tus ramas locales no se sincronizan automáticamente con los remotos en los que escribes, sino que tienes que enviar (push) expresamente las ramas que desees compartir. De esta forma, puedes usar ramas privadas para el trabajo que no deseas compartir, llevando a un remoto tan solo aquellas partes que deseas aportar a los demás.

Si tienes una rama llamada serverfix, con la que vas a trabajar en colaboración; puedes llevarla al remoto de la misma forma que llevaste tu primera rama. Con el comando git push (remoto) (rama):

$ git push origin serverfix

Ver más en:

* [Ramificaciones en Git - Ramas Remotas](https://git-scm.com/book/es/v2/Ramificaciones-en-Git-Ramas-Remotas)
* [Ramificaciones en Git - Reorganizar el Trabajo Realizado](https://git-scm.com/book/es/v2/Ramificaciones-en-Git-Reorganizar-el-Trabajo-Realizado)

## Archivo .gitignore

Git tiene una herramienta imprescindible casi en cualquier proyecto, el archivo "gitignore", que sirve para decirle a Git qué archivos o directorios completos debe ignorar y no subir al repositorio de código. En el gitignore se especificarán todas las rutas y archivos que no se requieren y con ello, el proceso de control de versiones simplemente ignorará estos archivos.

El ejemplo más claro que se puede dar surge cuando se trabaja con sistemas de gestión de dependencias, como npm, Bower, Composer. Al instalar las dependencias se descargan muchos archivos con documentos, tests, demos, etc. Todo eso no es necesario que se mantenga en el sistema de gestión de versiones, porque no forma parte del código de nuestro proyecto en concreto, sino que es código de terceros. Si Git ignora todos esos archivos, el peso total del proyecto será mucho menor y eso redundará en un mejor mantenimiento y distribución del código.

Otro claro ejemplo de uso de gitignore son los archivos que crean los sistemas operativos automáticamente, archivos que muchas veces están ocultos y no los vemos, pero que existen. Si no evitas que Git los procese, estarán en tu proyecto como cualquier otro archivo de código y generalmente es algo que no quieres que ocurra.

### Implementar el gitignore

Simplemente tienes que crear un archivo que se llama ".gitignore" en la carpeta raíz de tu proyecto. Dentro del archivo .gitignore colocarás texto plano, con todas las carpetas que quieres que Git simplemente ignore, así como los archivos.

La notación es muy simple. Por ejemplo, si indicamos la línea bower\_components/

Estamos evitando que se procese en el control de versiones todo el contenido de la carpeta "bower\_components".

Si colocamos la siguiente línea: \*.DS\_Store

Estaremos evitando que el sistema de control de versiones procese todos los archivos acabados de .DS\_Store, que son ficheros de esos que crea el sistema operativo del Mac (OS X) automáticamente.

Hay muchos tipos de patrones aplicables a la hora de especificar grupos de ficheros, con comodines diversos, que puedes usar para poder indicar, de manera muy específica, lo que quieres que Git no procese al realizar el control de versiones. Puedes encontrar más información en la documentación de gitignore, pero generalmente no lo necesitarás porque lo más cómodo es crear el código de este archivo por medio de unas plantillas que ahora te explicaremos.

Ver más en: [Archivo .gitignore](https://desarrolloweb.com/articulos/archivo-gitignore.html)

Si deseas ampliar tu información puedes visitar los siguientes recursos:

* [git - la guía sencilla](http://rogerdudler.github.io/git-guide/index.es.html)
* [Tutorial de Git. Manual básico con ejemplos](https://www.diegocmartin.com/tutorial-git/)
* [Documentación de Git](https://git-scm.com/book/es/v2)
* [Curso de Git Gratis](https://codigofacilito.com/cursos/git) (Vídeo-curso)
* [Documentación de gitignore](https://git-scm.com/docs/gitignore)
* [gitignore.io](https://www.gitignore.io/)