Sintaxis y Semántica de los Lenguajes
Entendiendo el manejo del versionado, el uso de branch y merge en Git

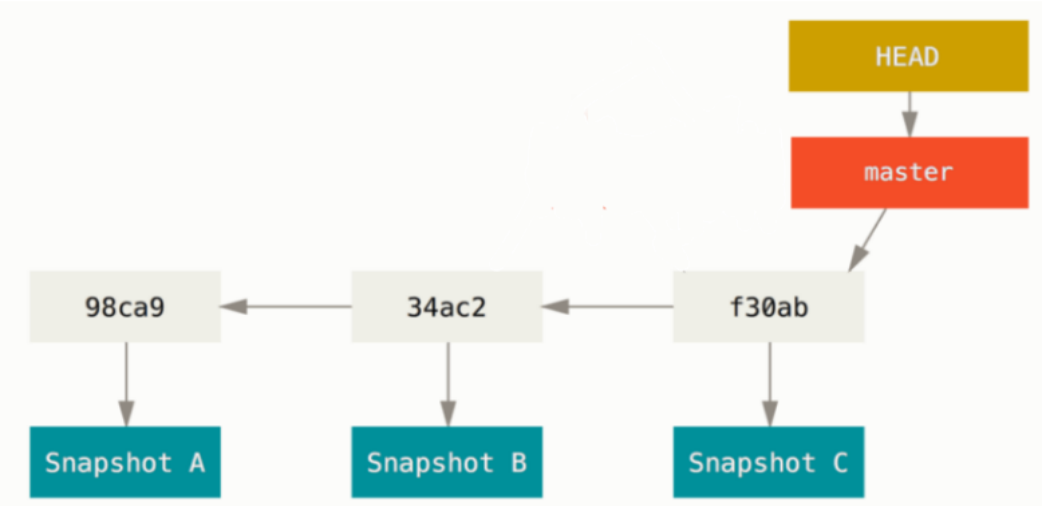
Antes de profundizar en los comandos de branch and merge debemos comprender un poco mejor el funcionamiento de Git.

Actualmente cualquier sistema de versionado permite la ramificación del trabajo (branching). Cuando hablamos de ramificaciones, significa que tomamos la rama principal de desarrollo, generalmente denominada “master”, y a partir de allí continuamos trabajando sin seguir la rama principal de desarrollo. De forma sencilla, es como si estuviésemos realizado una copia de lo trabajado hasta el momento, y comenzamos a trabajar de forma paralela sin afectar a la rama principal.

La forma en la que Git maneja las ramificaciones es rápida, haciendo así de las operaciones de ramificación algo casi instantáneo, al igual que el avance o el retroceso entre distintas ramas. Git promueve un ciclo de desarrollo donde las ramas se crean (branch) y se unen ramas entre sí (merge), incluso varias veces en el mismo día. Entender y manejar esta opción proporciona una poderosa y exclusiva herramienta que puede cambiar la forma de desarrollo.

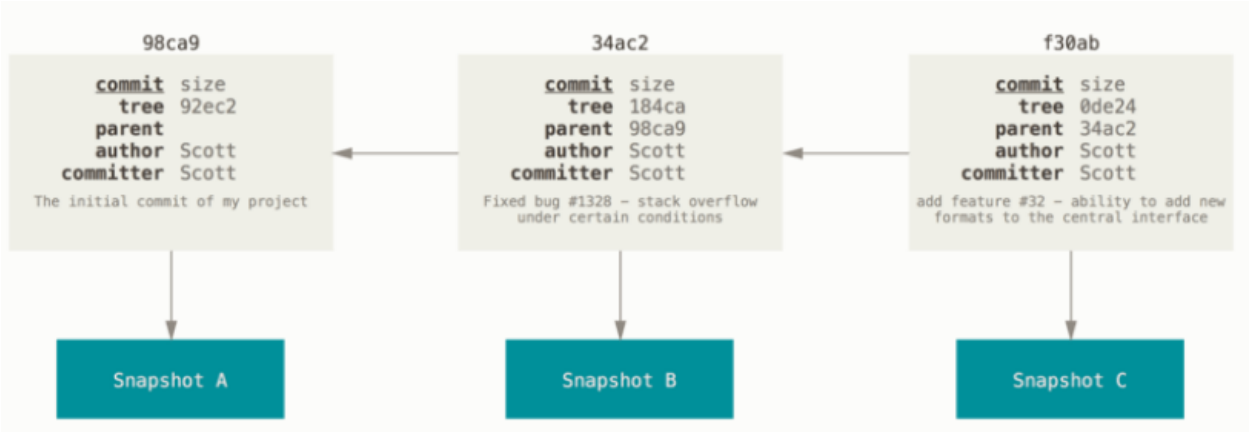
En cada confirmación de cambios (commit), Git almacena una instantánea del trabajo preparado. Dicha instantánea contiene metadatos con el autor y el mensaje explicativo, y uno o varios apuntadores a las confirmaciones (commit) que sean padres directos de esta.

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 apuntando al último commit que realicemos. A continuación mostramos un esquema para comprender mejor.

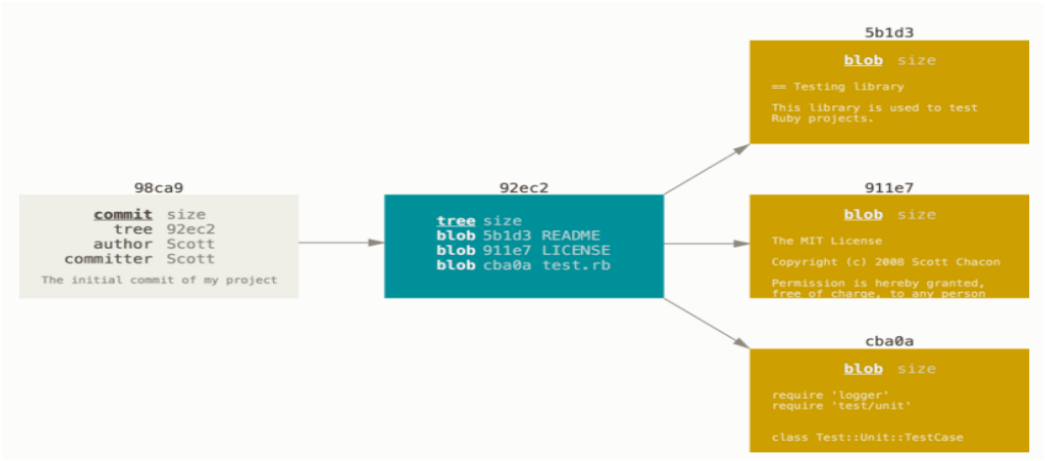


HEAD apunta a la rama en la cual nos encontramos trabajando actualmente, en este caso “master”. Y la rama master apunta a la última instantánea (Snapshot C) correspondiente al último commit realizado sobre la rama master. Pero, es importante saber que cada instantánea almacena información que le permite apuntar a las instantáneas anteriores, de modo tal que tenemos la posibilidad de retroceder y avanzar sobre la rama de trabajo master a cualquier punto (commit) de la misma.

Una aclaración no menor, la rama “master” en Git, no es una rama especial. Es como cualquier otra rama. La única razón por la cual aparece en casi todos los repositorios es porque es la que crea por defecto el comando git init. A continuación otro esquema con mayor nivel de detalle de las instantáneas (snapshot) dentro de una rama



Como puede verse, cada instantánea apunta la anterior donde dice **parent**, además figuran otros metadatos como el **autor**, el que realizó el comit y el **mensaje** asociado a esa confirmación. Por ejemplo, el snapshot C está asociado al commit con código f30ab, que apunta al commit anterior con código 34ac2. Por último aparece un dato importante **tree**, ese dato apunta un árbol que detalla todos los cambios realizados en ese commit. A continuación una esquema para comprender mejor:



Allí puede verse el commit 98ca9 que apunta al árbol 92ec2 que a su vez apunta a tres nodos que referencian a tres archivos donde se hicieron los cambios, cada uno con su propio código.

Ahora bien, ¿qué son esos códigos? En el ejemplo del esquema tenemos una carpeta con tres archivos que hemos preparado (stage) y confirmado (commit). Al preparar los archivos, Git realiza una suma de control de cada uno de ellos, almacena una copia de cada uno en el repositorio, denominadas blobs, y guarda cada suma de control (el código que vemos)

Esa suma de control garantiza la integridad del repositorio, dado que todo archivo almacenado en el repositorio tiene una suma de control que lo referencia, por lo que no es posible hacer una modificación en un archivo sin que Git lo sepa dado que cambiaría el código asociado a la suma de control. Eso garantiza que Git pueda detectar todos los cambios.

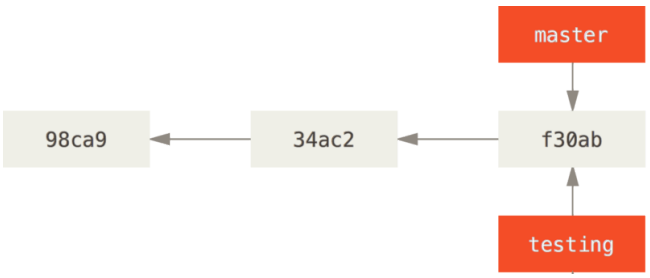
Puntualmente el mecanismo de control de suma que utiliza se denomina es un códiog hash SHA-1. Dicho código es una cadena de 40 caracteres hexadecimales que se calculan en base al contenido de cada archivo y cada carpeta. Un código SHA-1 hash se ve del siguiente modo 24b9da6552252987aa493b52f8696cd6d3b00373

Verán este tipo de códigos por todo Git, pero en general solo veremos los últimos 5 caracteres de este código hash dado que es muy extraño que haya una coincidencia a menos que nuestro repositorio sea demasiado grande. De todos modos, Git tiene el código completo de forma interna. ¿Han realizado el cálculo de cuántas posibles combinaciones hay? 16^{40}

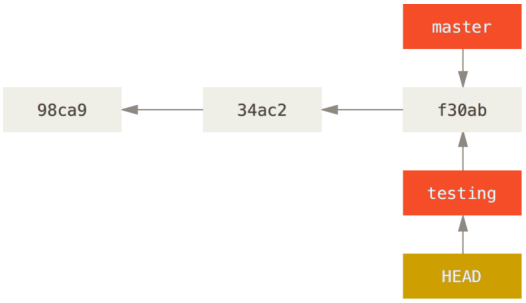
Git almacena en su base de datos cada valor hash para referenciar a los archivos. Ya que un mismo nombre de archivo puede tener varios códigos hash asociados a cada uno de los cambios que va teniendo a lo largo del tiempo.

Creando una rama (branching)

Ahora bien, ¿qué sucede cuando creamos una rama mediante el comando branch? Simplemente se crea un nuevo apuntador con el nombre indicado apuntando a la última versión confirmada que está apuntando la rama actual. Por ejemplo, si estamos en la rama “master” y realizamos un **git branch testing**, se creará un nuevo apuntador llamado “testing” que por el momento estará apuntando al mismo commit apuntado por la rama master. Veamos un esquema

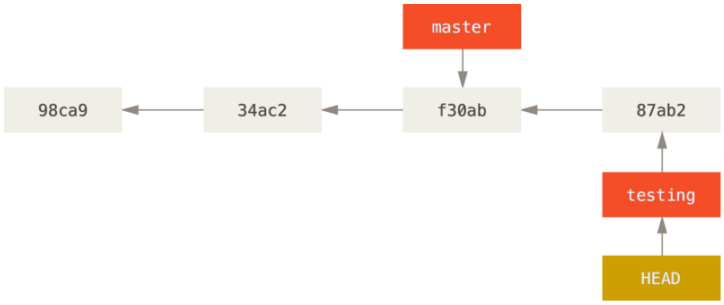


Podemos ver que tenemos dos ramas, que están apuntado al mismo commit, en este caso con código f30ab. Ahora, debemos indicarle a Git que queremos trabajar en esta nueva rama “testing” dado que Git aún no lo sabe, solo creó la rama. Para saltar de una rama a otra, usamos el comando el comando git checkout. En este ejemplo deberíamos indicar: ***git checkout testing*** para que el HEAD apunte a la rama testing.

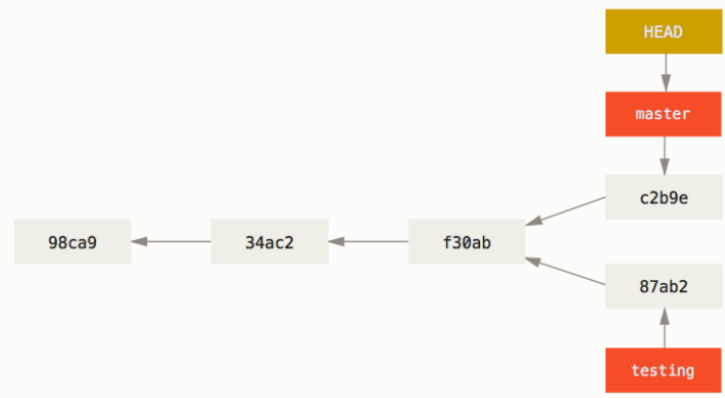


Para crear una rama y saltar a ella en un solo comando ***git checkout -b nombreRama***

Hasta el momento ambas ramas apuntan al mismo commit, las diferencias se empiezan a generar al momento de realizar un nuevo commit trabajando sobre la rama testing. Si modificamos un archivo y confirmamos el cambio, tendremos lo siguiente



Aquí ya vemos que la rama master no tiene conocimiento del último commit 87ab2. Ahora, si volvemos a saltar a la rama master y hacemos algún cambio sucederá lo siguiente



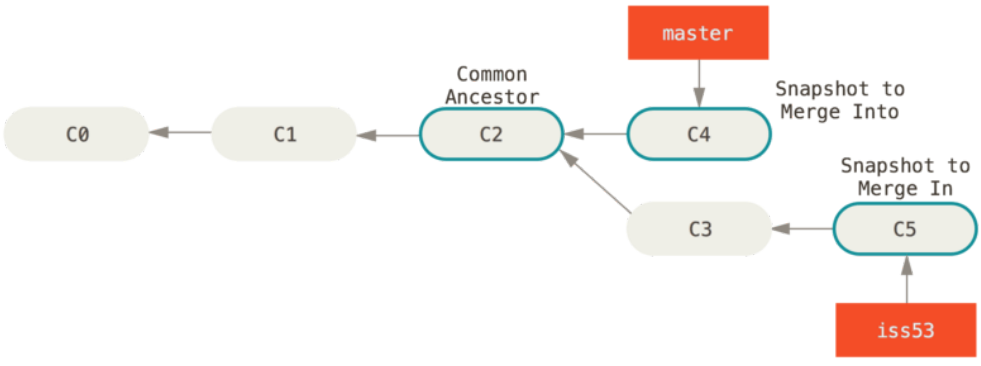
Puede ver que cada rama tiene una confirmación (commit) distinta, cada una con sus propios cambios. Eso permite que sobre un mismo repositorio se trabaja de forma simultánea teniendo un registro de todos los cambios en cada una de las ramas. Es importante destacar que cuando saltamos a una rama en Git, los archivos del directorio de trabajo cambian.

Si saltamos a una rama antigua, el directorio de trabajo retrocederá para verse como lo hacía la última vez que confirmaste un cambio en dicha rama. Si Git no puede hacer el cambio limpiamente, no te dejará saltar, por lo que es importante que antes de saltar de una rama a la otra confirmes los cambios en la rama de trabajo actual.

Debido a que una rama Git es realmente un simple archivo que contiene los 40 caracteres de una suma de control SHA-1, no cuesta nada el crear y destruir ramas en Git. Esto contrasta fuertemente con los métodos de ramificación usados por otros sistemas de control de versiones, en los que crear una rama nueva supone el copiar todos los archivos del proyecto a un directorio adicional nuevo. Esto puede llevar segundos o incluso minutos, dependiendo del tamaño del proyecto; mientras que en Git el proceso es siempre instantáneo. Y además, debido a que se almacenan también los nodos padre para cada confirmación, el encontrar las bases adecuadas para realizar una ***fusión*** entre ramas es un proceso automático y generalmente sencillo de realizar. Animando así a los desarrolladores a utilizar ramificaciones frecuentemente.

Realizando un merge (fusión)

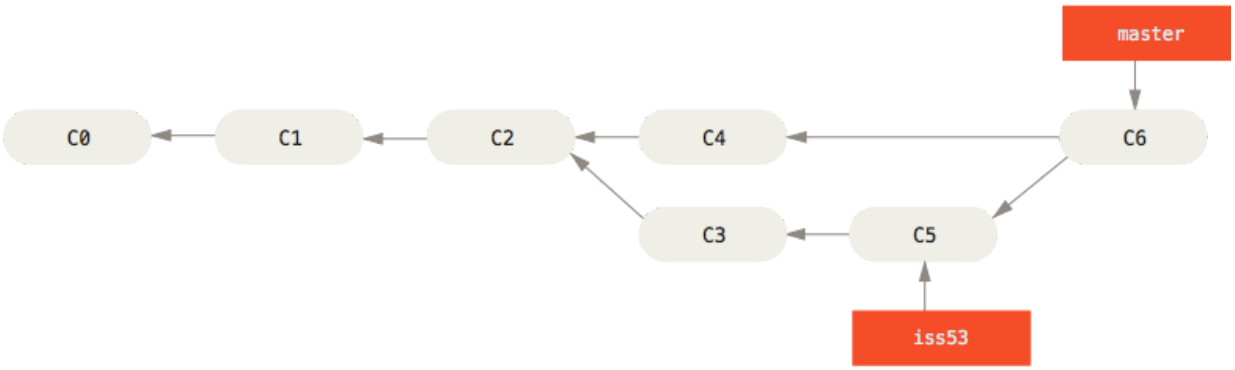
Supongamos que hemos trabajado en dos ramas “master” y “iss53”, y queremos fusionar ambas para tener una versión única en el repositorio a partir de la cual se seguirá trabajando en conjunto dado que en cada rama ya se completó el trabajo para el cual fueron destinadas.



Podemos ver que ambas ramas tiene un antecesor común C2. Ese fue el punto donde probablemente de la rama master se hizo un branch creando la rama iss53. Y luego cada rama continuó trabajando por separado.

Ahora llegó el momento de unirlas mediante el comando merge. Para eso, es fundamental movernos a la rama sobre la cual queremos realizar la fusión (merge). En este caso nos vemos a la rama master y a partir de allí ejecutaremos el comando merge ***git merge nombreRama***

Git realizará una fusión a tres bandas, utilizando las dos instantáneas apuntadas por el extremo de cada una de las ramas (C4 y C5) y el ancestro común a ambas (C2), creando una nueva instantánea (snapshot) resultante de la fusión de las mismas; y finalmente 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 dado que C6 tiene dos antecesores C4 correspondiente al último commit de la rama master y C5 correspondiente al último commit de la rama iss53.



Como puede observarse la fusión (merge) quedó apuntada por la rama “master”. Y en cambio la rama iss53 quedó apuntando a su última confirmación C5. Ahora que todo tu trabajo ya está fusionado con la rama master. Es probable que luego de una fusión deseemos eliminar algunas ramas de trabajo. Para destruir una rama ***git branch -d nombreRama***

Problemas con la fusión (merge)

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 pretendemos fusionar, Git no será capaz de fusionarlas directamente

En este caso Git no crea automáticamente una nueva fusión confirmada (merge commit), sino que hace una pausa en el proceso, esperando a que se resuelvan los conflictos. Para ver qué archivos permanecen sin fusionar en un determinado momento conflictivo de una fusión, puedes usar el comando ***git status***

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 serán de guía cuando se abran manualmente los archivos implicados para su edición y corrección. Tras resolver todos los bloques conflictivos, debemos ejecutar git add para marcar cada archivo modificado y luego realizar git commit para confirmar los cambios a la versión fusionada de ambas ramas. Si en lugar de resolver directamente preferimos utilizar una herramienta gráfica, se puede usar el comando ***git mergetool***

Este documento ha sido elaborado en base al manual de referencia de Git, toda la información de forma más detallada la encontrará en https://git-scm.com/book/en/v2 Para profundizar en el uso de Git recomiendo su lectura completa.

Espero que les sea de utilidad