

TutoríaGit2020



git

Autor: Roberto Rodríguez Ortiz



Versión 1.1 Octubre 2020

Este documento se publica bajo licencia CreativeCommons “Reconocimiento-Compartir Igual (by-sa)”



Contenido

1. ¿Qué es Git?	4
2. Software a utilizar	6
Servidor	6
Cliente	7
3. Funcionamiento del repositorio.....	8
3.1. Operaciones básicas	9
3.2. Conceptos.....	10
4. Creación del repositorio.....	11
4.1. Clonar el repositorio.....	13
4.2. Inclusión del .gitignore	14
5. Operaciones básicas.....	16
5.1. Añadimos los ficheros de un proyecto.....	16
5.2. Descargamos los cambios realizados por otros usuarios.....	17
5.3. Resolución de conflictos.....	18
6. Operaciones avanzadas.....	22
6.1. Cambiar de commit.....	22
6.2. Creación de una nueva rama	24
6.3. Como establecer la rama origin/master a un commit concreto	26
6.4. Merge	26
6.5. Rebase	28
7. Para saber más	29

1. ¿Qué es Git?

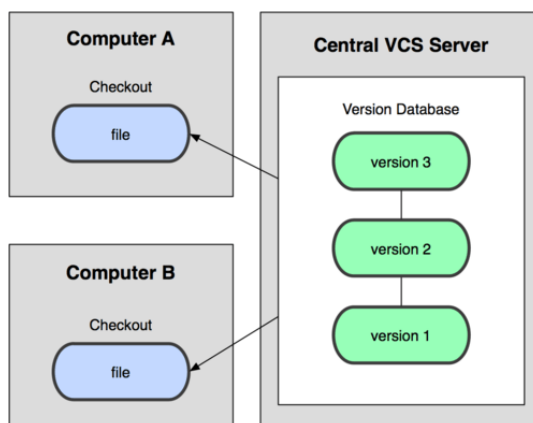
Git es un software de control de versiones que permite gestionar los cambios en un fichero, permitiendo revertir dichos cambios a cualquier estado anterior, comprobar la autoría de cada modificación y facilitando la colaboración entre los miembros de un equipo.

Creado por Linus Torvals para el desarrollo del núcleo de Linux, se ha consolidado como una de las herramientas imprescindibles para el desarrollo de software.

Aunque nos centraremos en su uso para controlar los cambios en código fuente, este sistema puede ser utilizado para gestionar cualquier tipo de documento o fichero, por ejemplo, diseños web ó imágenes.

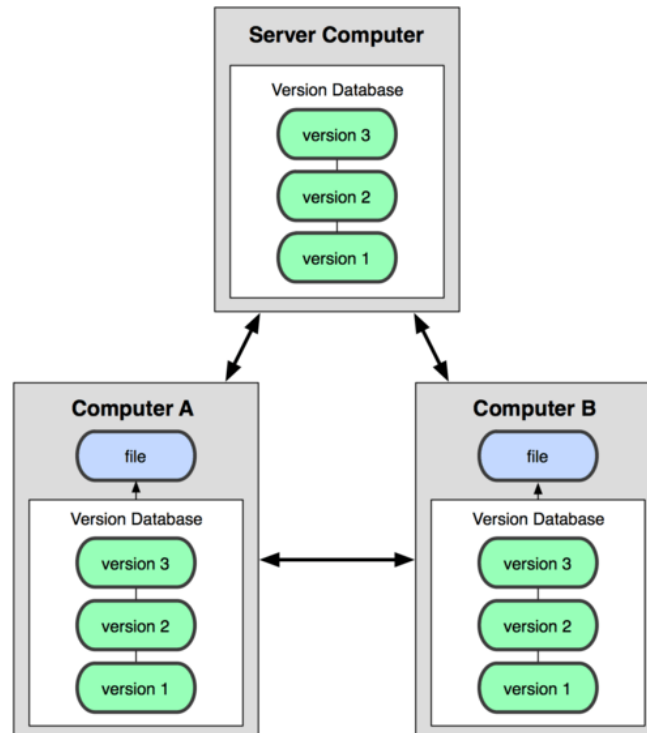
Dentro de las herramientas de control de versiones existen dos arquitecturas, una centralizada y otra distribuida.

En la **centralizada** un servidor almacena el repositorio* y todos los clientes acceden a dicho repositorio:



**Repositorio* : estructura de datos centralizada donde se guardan los archivos del proyecto que van a ser modificados a lo largo del tiempo.

En la **distribuida** cada cliente conserva una copia completa del repositorio de forma que puede trabajar sin conexión, este sistema entre [otras ventajas](#) permite realizar las operaciones habituales de forma más rápida al no necesitar de una comunicación con el servidor



2. Software a utilizar

En el proceso de controlar los cambios en un proyecto de software necesitaremos dos elementos, un servidor que controle todos los cambios y un cliente que le envíe las modificaciones.

Git sigue un modelo distribuido en el cual los clientes almacenan una copia completa del repositorio lo que les permite trabajar y aplicar modificaciones en su copia sin estar conectados al servidor.

Servidor

Para la creación del servidor disponemos de dos opciones, la creación de un servidor propio o la utilización de un servicio externo, vamos a explorar las dos opciones:

Servidor propio:

- **GitBucket:** es un software que podemos instalar utilizando Tomcat que nos proporciona un entorno web similar a BitBucket.
- **GitLab :** proporciona un entorno similar a GitHub. También tiene una plataforma externa en www.gitlab.com.

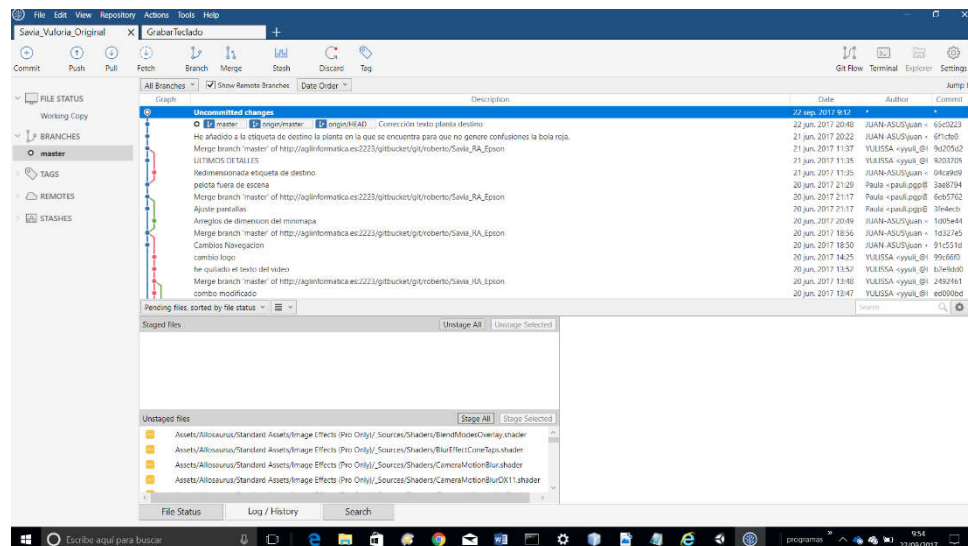
Servidor externo:

- **GitHub.com:** es el estándar de facto para el desarrollo con Git, sustituyendo a SourceForge como el mayor repositorio de software libre,
- **BitBucket.com:** principal alternativa a GitHub, una de sus ventajas es que permite tener repositorios privados de forma gratuita.

Cliente

Existen múltiples alternativas tanto en modo texto como con Interfaces gráficas:

- **SourceTree**: creado por la empresa Atlassian posee uno de los interfaces más atractivos.



- **GitKraken** : otra alternativa gratuita para proyectos no comerciales.
- **GitHub**: la plataforma posee su propio cliente con interfaz gráfica.

Otra posibilidad es utilizar las integraciones que proporcionan la mayoría de los entornos de desarrollo modernos (IDEs). En nuestro caso vamos a utilizar la combinación GitBucket + SourceTree por resultar más adecuada a nuestras necesidades.

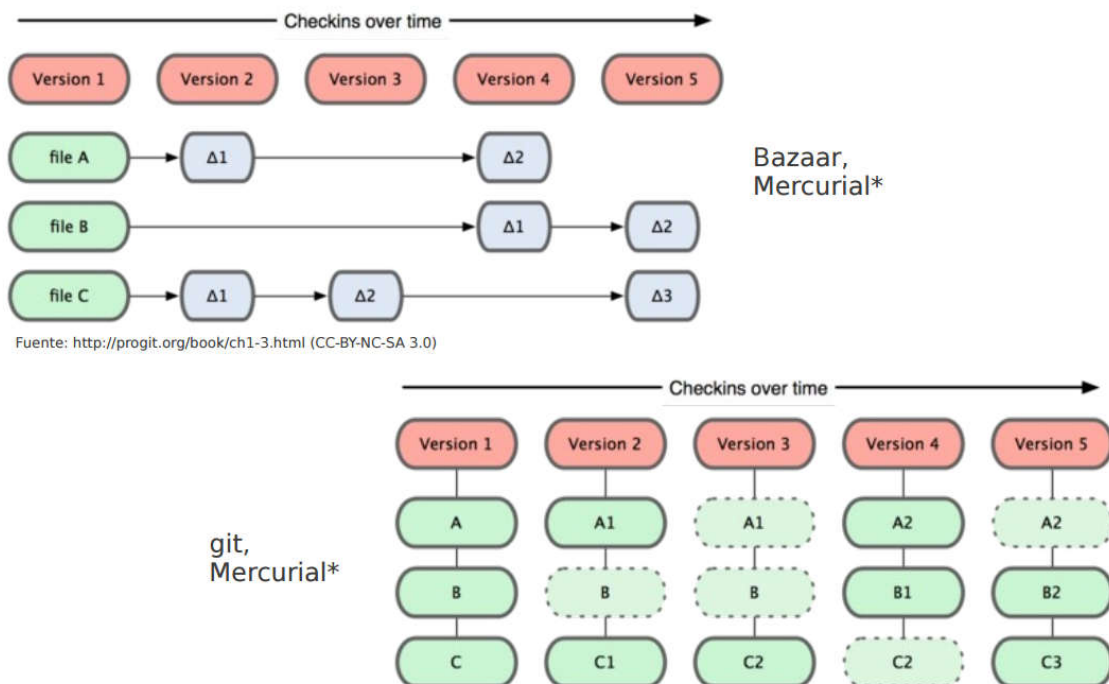
3. Funcionamiento del repositorio

El repositorio es el **almacén central** donde se guardan todos los archivos del proyecto y todos los cambios que van sucediendo a lo largo del tiempo sobre los archivos.

En él repositorio no se guarda cada versión del archivo por separado, si no que se guarda una copia inicial y una serie de cambios incrementales del mismo, lo que redunda es un ahorro importante de espacio. Cada vez que pidamos una versión concreta de un archivo el sistema selecciona la versión inicial y aplica los cambios intermedios hasta la versión que se solicite.

En el caso de git se aplica otra aproximación al problema y se guardan **instantáneas** del archivo cada vez que este cambia, es decir, cada vez que un archivo cambia se guarda de forma completa, aunque no se vuelve a guardar si no se ha modificado desde la última versión del repositorio.

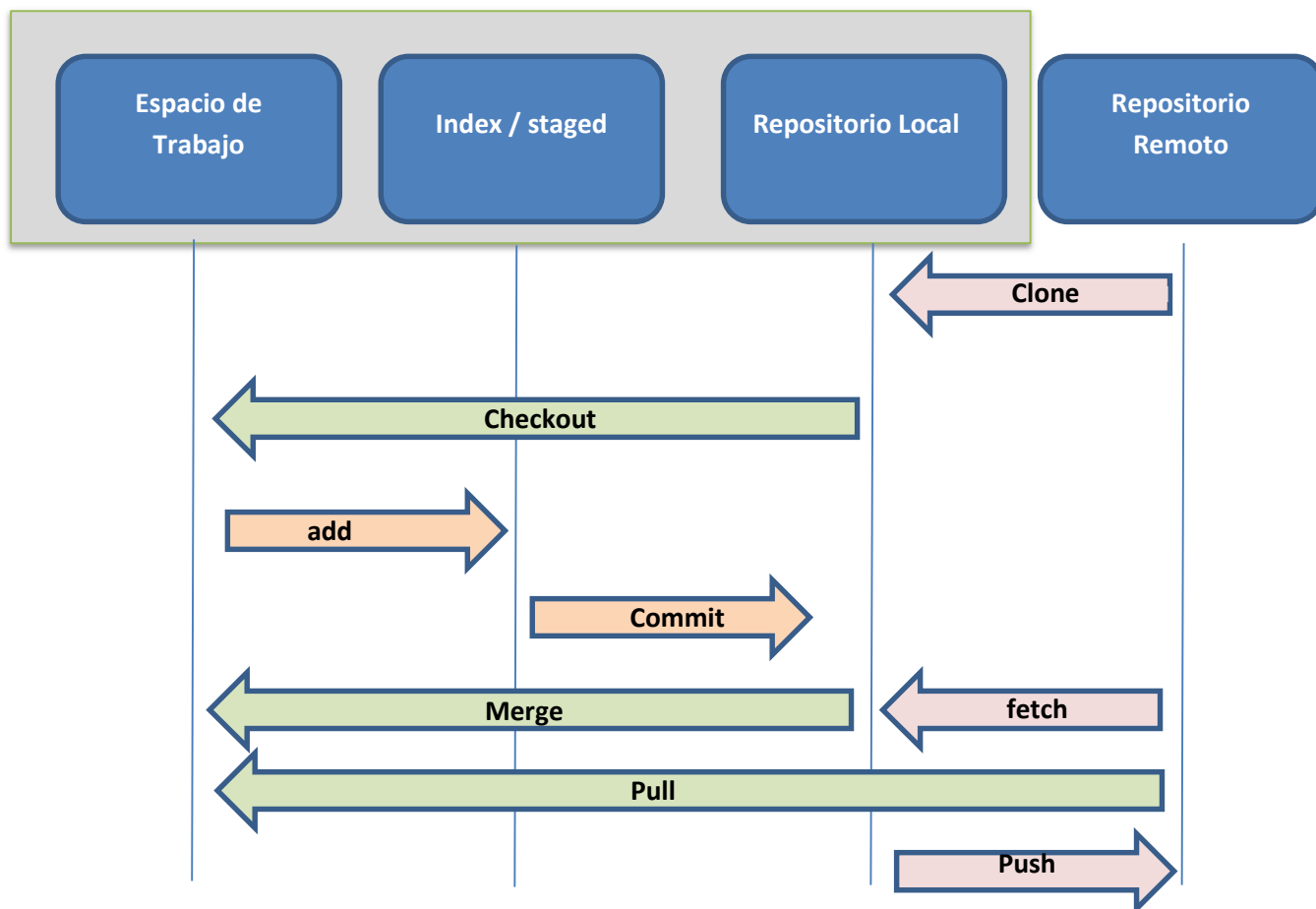
Diferencias versus instantáneas



3.1. Operaciones básicas

- **checkout:** acción de actualizar parte o todo el árbol de trabajo con un objeto árbol o blob desde la base de datos de objeto, además actualiza el índice y la referencia HEAD si se ha cambiado de rama.
- **clone:** obtener una copia local completa de un repositorio git remoto.
- **commit:** información de una revisión dada. Incluye: los padres del objeto, la persona que ha realizado el commit de la revisión, el autor de la revisión, la fecha de la misma, un mensaje asociado, el objeto tree que corresponde al directorio raíz de la revisión.
- **fetch:** obtener la cabeza de una rama (o varias) desde un repositorio remoto, copiando los objetos que falten y moviendo la(s) cabeza(s) remota(s).
- **pull:** hacer un fetch seguido de un merge, con una rama remota dada.
- **push:** enviar los objetos de la rama local que no están en la rama remota a la que hace referencia el pull, y actualizar la cabeza de la rama remota. Es la acción complementaria de pull. Si la cabeza de la rama remota no es un ancestro de la cabeza de la rama local, el push falla*.

De forma gráfica:



3.2.Conceptos

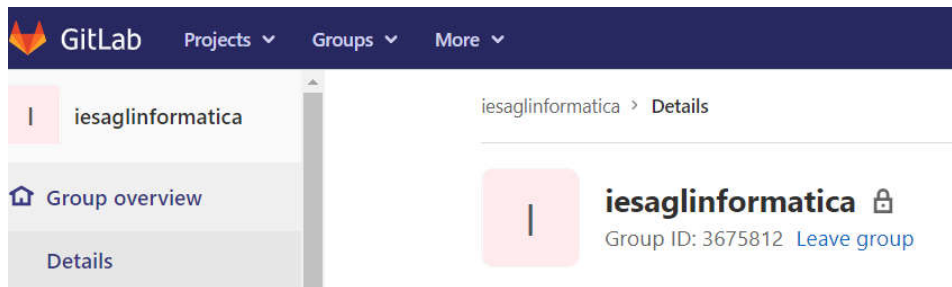
Algunos nombres que veremos en el interfaz de usuario y su significado:

- **cabeza:** una referencia con nombre, que apunta al objeto commit de la punta de una rama.
- **HEAD:** el commit en el cual está situado actualmente el repositorio. Normalmente es el último commit de la rama actual.
- **origin:** nombre por defecto del repositorio remoto principal.
- **master:** nombre de la rama que se crea por defecto en la creación del repositorio, en la mayoría de los casos significa rama principal.

4. Creación del repositorio

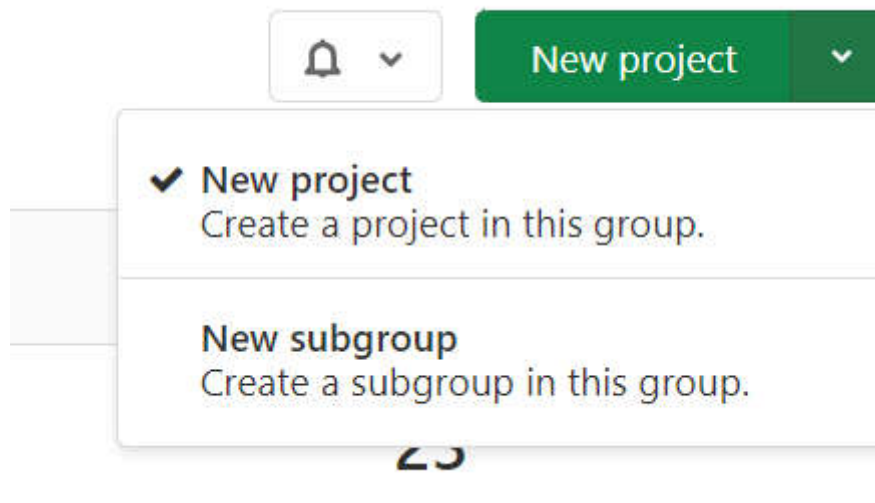
4.1. GitLab

En la plataforma www.gitlab.com disponemos de un grupo donde almacenar repositorios privados a través de una licencia educativa.



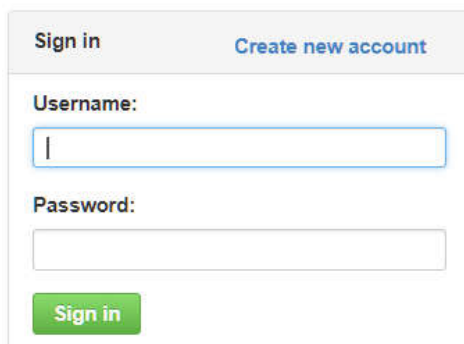
Para registrarse en la página únicamente es necesario un correo electrónico, posteriormente un administrador del grupo puede añadirte al grupo y habilitarte para la creación de grupos y repositorios.

En la parte derecha dispones de dos opciones principales



4.2. GitBucket

El primer paso será la creación de un repositorio en la plataforma **GitBucket**, para ello debes estar identificado en la plataforma, si aún no tienes un usuario puedes hacerlo desde la opción *Create new account*



Sign in [Create new account](#)

Username:

Password:

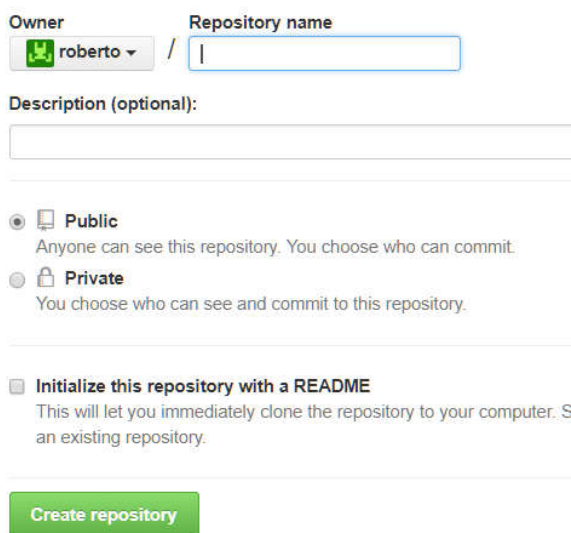
[Sign in](#)

Una vez dentro de la plataforma puedes pulsar sobre el *botón New repository* para la creación del repositorio.



Your repositories 14 [New repository](#)

En esta pantalla estableceremos el nombre del repositorio, una descripción opcional y el tipo de acceso público o privado del mismo.



Owner [roberto](#) / Repository name

Description (optional):

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. S an existing repository.

[Create repository](#)

Una vez creado el repositorio en el servidor remoto, aún solo contiene un fichero README.md con la descripción, vamos a proceder a clonarlo en nuestro equipo.



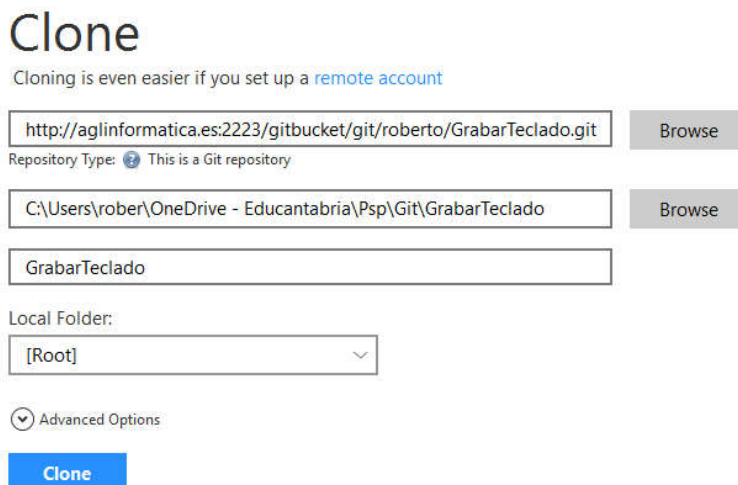
4.3. Clonar el repositorio

Mediante la clonación hacemos una copia del repositorio remoto en nuestro equipo, para clonar un repositorio no necesitamos ningún permiso únicamente debemos indicar la URL del mismo.

En cada repositorio tenemos a la derecha la información con la URL para copiar y la opción de descargar en forma de zip todo el contenido.

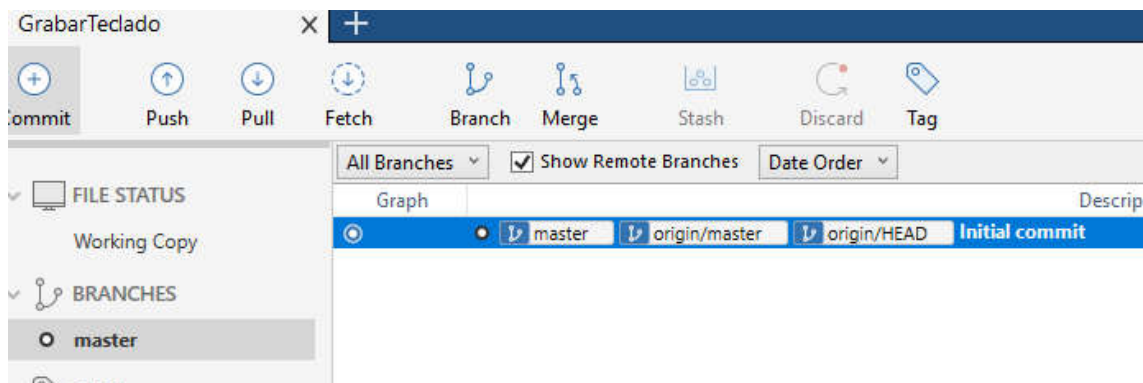


Copiamos la URL y abrimos SourceTree pulsamos en el menú File->Clone y nos solicitará la URL, el directorio destino en nuestro equipo y un nombre descriptivo para el proyecto. También puede solicitarlos las credenciales de nuestro usuario de GitHub.



Si es la primera vez que utilizamos SourceTree también nos puede solicitar un nombre de usuario y un correo que aparecerá como información en nuestros commits.

En este momento ya tenemos una **copia local del repositorio** y SourceTree nos mostrará la información del mismo, por defecto vemos el contenido de la rama master con el commit inicial que solo contiene el fichero Readme.





4.4. Inclusión del .gitignore

Para que git no incluya ficheros de configuración del IDE, temporales de compilación y todo aquello que no creamos necesario controlar de nuestro proyecto, podemos utilizar un fichero .gitignore en la raíz del proyecto. Por ejemplo para un proyecto de Java en Netbeans podría ser algo como lo siguiente :

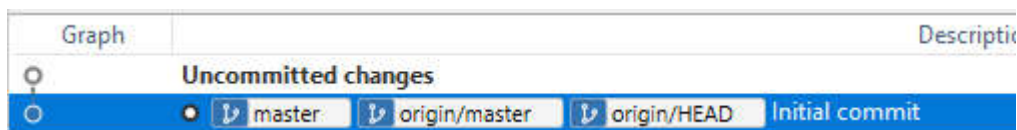
```
nbproject/private/
build/
nbbuild/
dist/
nbdist/
.nb-gradle/
```

Creamos el fichero y lo incluimos en la raíz del proyecto:

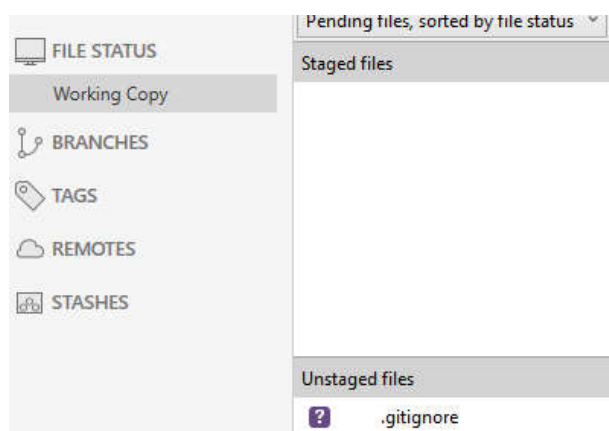
Nombre

 .gitignore
 README.md

SourceTree nos informará que tenemos cambios que no han sido incluidos en un commit :



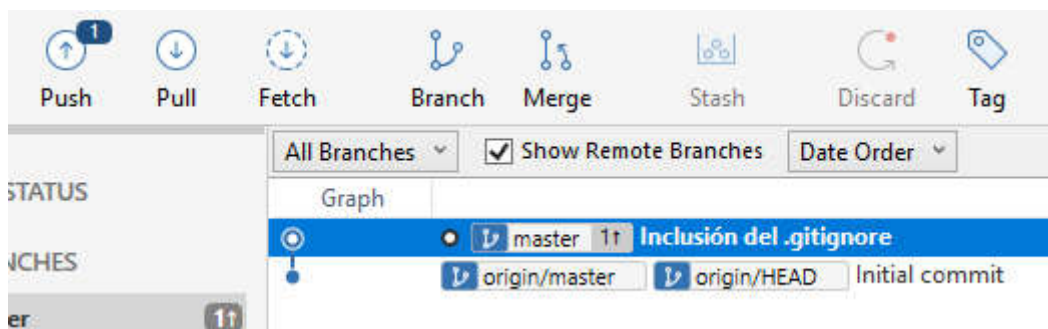
SourceTree para ver dichos cambios pulsamos en la izquierda sobre *File Status* -> *Working copy*. Aquí veremos los ficheros de nuestra copia de trabajo que no están en un commit y por lo tanto tampoco están en nuestro repositorio local. En este caso solo aparecerá el fichero *.gitignore* pues es el único cambio realizado :



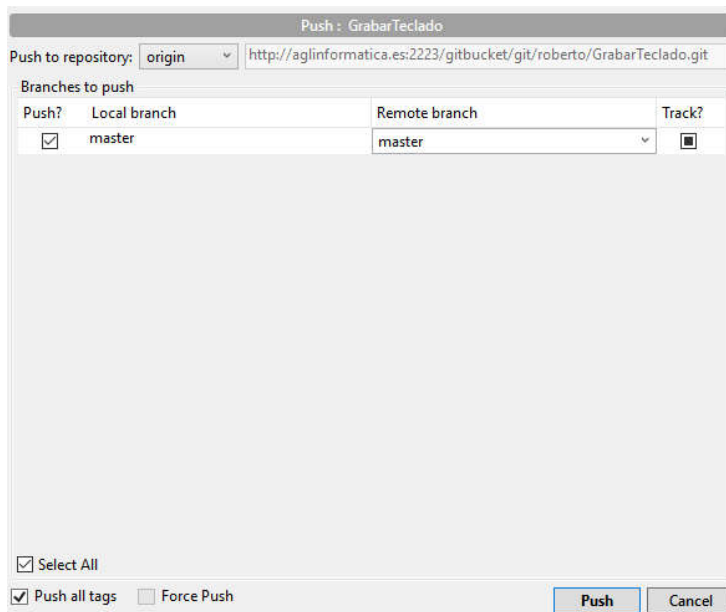
Como vemos aparece en un apartado **Unstaged**, para que un fichero sea incluido en el commit debemos pasarlo al apartado **Staged**, puede que no nos interese para al Staged a todos los ficheros modificados desde el último commit.

Una vez subido algún fichero al estado Staged, incluimos algún **comentario** en el apartado commit, que sea descriptivo de los cambios introducidos y pulsamos sobre commit.

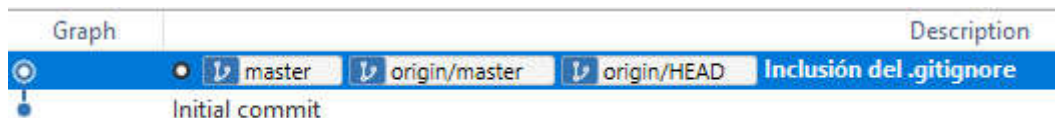
Veamos el estado actual del proyecto:



Nuestra rama master ha avanzado un commit, no así la rama master del repositorio remoto ni la cabecera del mismo y vemos que podemos realizar el **Push** de un commit. Vamos a pulsarlo y enviar los cambios al repositorio remoto.



Ahora la rama local y la remota están en el mismo commit :



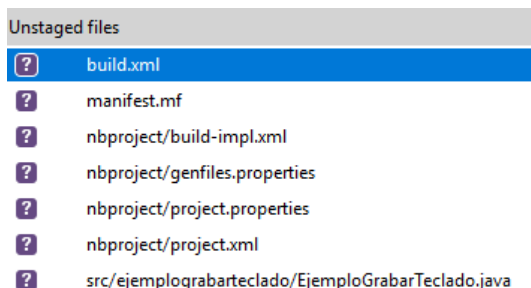
5. Operaciones básicas

5.1. Añadimos los ficheros de un proyecto

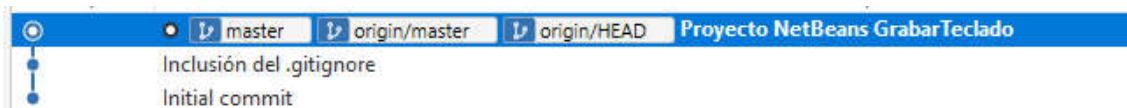
Vamos a copiar en dicho directorio los ficheros de un proyecto Netbeans,

Nombre
nbproject
src
.gitignore
build.xml
manifest.mf
README.md

Repetimos el proceso que seguimos con el fichero .gitignore e incluimos los nuevos ficheros en un commit que enviaremos mediante un push al servidor remoto.



Un nuevo commit aparecerá en los dos repositorios:



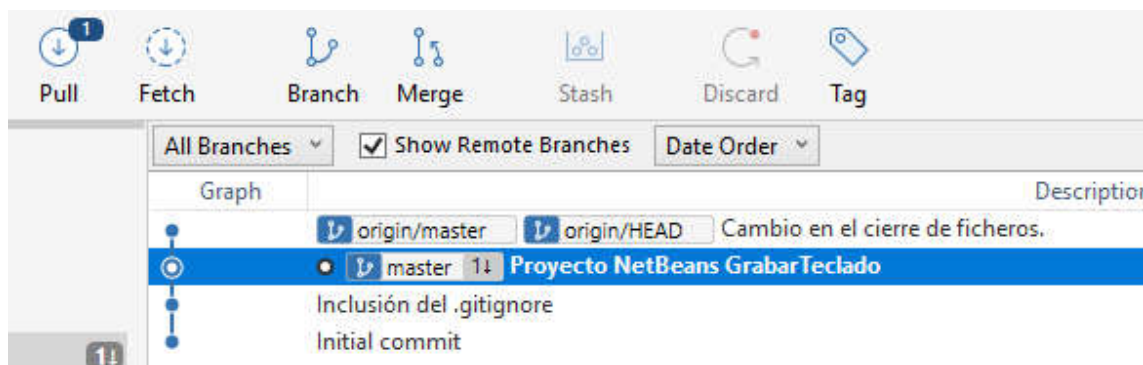
5.2.Descargamos los cambios realizados por otros usuarios

Mientras trabajemos nosotros solos en el repositorio los únicos cambios serán los que nosotros realicemos, si estamos trabajando en equipo en un proyecto y varios programadores colaboran en la creación del repositorio, será habitual que tengamos que incorporar a nuestro repositorio local los cambios que se realicen sobre el repositorio remoto.

Para comprobar si existe algún cambio en el repositorio podemos realizar un fetch y comprobar si existe algún cambio pendiente:



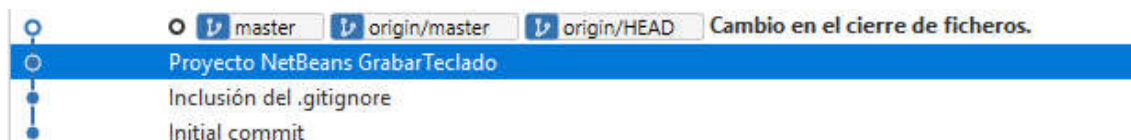
En este caso el resultado es el siguiente :



Vemos que otro usuario ha realizado un cambio de forma que la rama en el repositorio remoto tiene un commit más y en el repositorio local vamos uno

por detrás. Se indicará con un número sobre el botón Pull el número de commits que nos faltan.

Si pulsamos sobre Pull se realizará un fetch + merge de forma que integraremos los cambios del repositorio remoto en el local. Si no hay ningún conflicto el resultado es el siguiente :



De nuevo los dos repositorios están en el mismo commit.

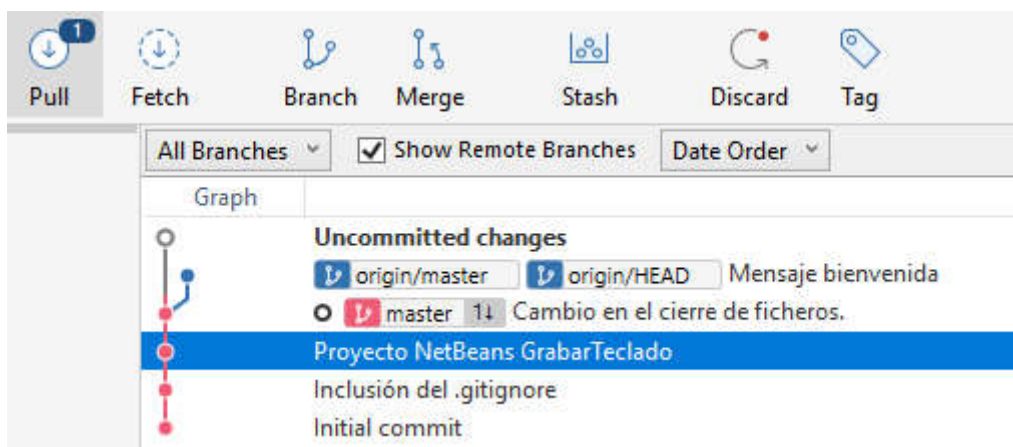
5.3.Resolución de conflictos

Mientras los cambios que sufran los dos repositorios no afecten al mismo fichero los Push y los Pulls se sucederán sin problemas. En el caso que intentemos integrar un fichero sobre el cual nosotros también hemos realizado cambios provocará un conflicto que debemos solventar.

Supongamos que realizo un cambio en mi clase GrabarTeclado, simplemente añadido la siguiente línea en la finalización del programa

```
System.out.println("Fin del programa");
```

Sin nosotros saberlo otro usuario ha subido ya un cambio con un mensaje de bienvenida al comienzo del programa. Veamos lo que sucede en este caso:



Vemos que existe un commit en el repositorio remoto, en el cual se ha incluido un mensaje de bienvenida, Intento realizar el Pull.

```
Pulling
[Redacted]
[ ] Show Full Output

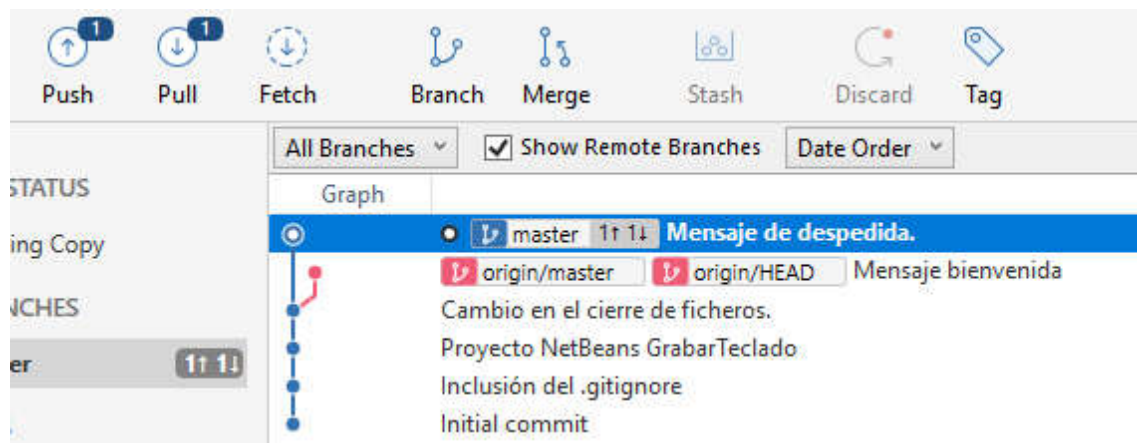
git -c diff.mnemonicprefix=false -c core.quotepath=false fetch origin

git -c diff.mnemonicprefix=false -c core.quotepath=false pull origin master
From http://aglinformatica.es:2223/gitbucket/git/roberto/GrabarTeclado
* branch      master    -> FETCH_HEAD

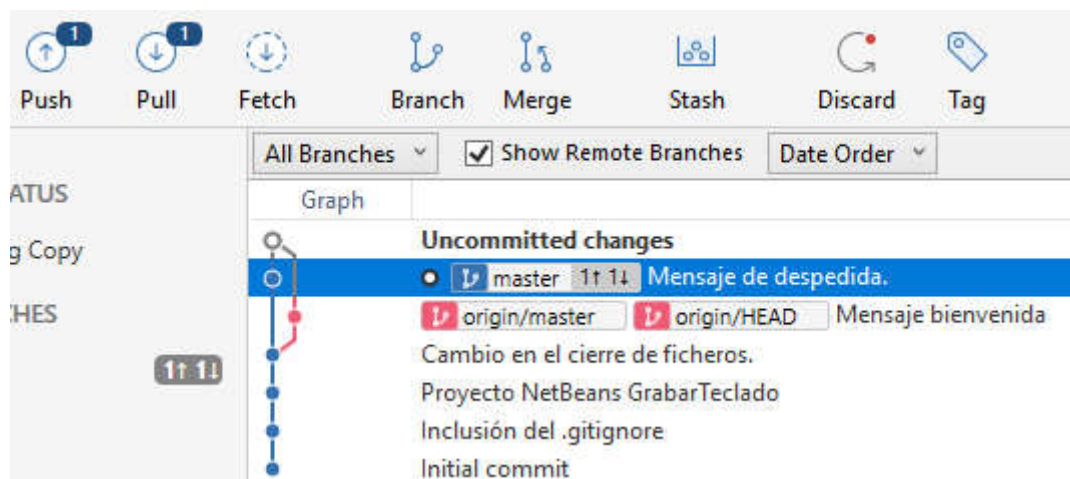
Updating 39e522a..78b6796
error: Your local changes to the following files would be overwritten by merge:
       src/ejemplograbarteclado/EjemploGrabarTeclado.java
Please commit your changes or stash them before you merge.
Aborting

Completed with errors, see above.
```

Se produce un error en el cual se nos avisa que habrá cambios que se sobrescribirán en nuestro proyecto. Vamos por tanto a realizar un commit de nuestros cambios para que no se pierdan.



Ahora tenemos pendiente un Push y un Pull, pues hay cambios en los dos repositorios, primero intentamos el Pull.



Git ha intentado mezclar los dos ficheros pero tenemos que revisarlo manualmente:

```
System.out.println("Bienvenidos al programa");
if (args.length == 0 ) {
```

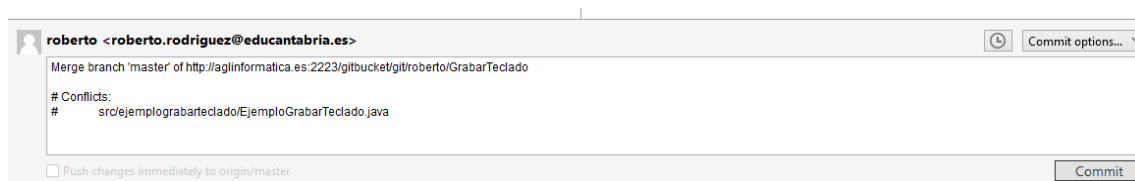
El mensaje de bienvenida se ha integrado perfectamente, pues es un código nuevo, en cambio nuestro código de despedida entra en conflicto con lo que se encuentra en el repositorio remoto (incluso una línea en blanco puede ser causa del conflicto), vemos marcado como <<<<HEAD nuestro código y la línea en blanco del repositorio remoto.

```
    }
<<<<<< HEAD
    System.out.println("Fin del programa");
=====
>>>>>> 78b6796047c59fbe8f499e8bdc0b0a0aab32d8cf
    }
```

Una vez cambiado el código vamos a realizar otro commit para guardar los nuevos cambios,

```
    }
    System.out.println("Fin del programa");
}
```

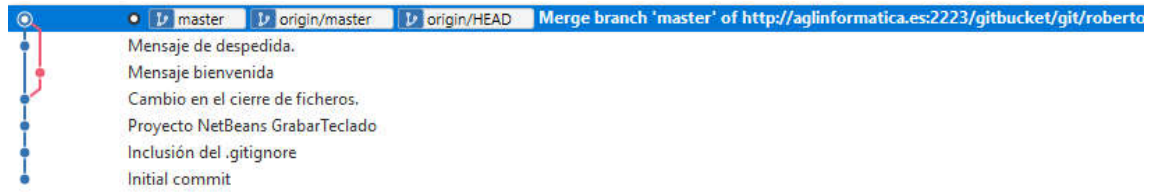
En este caso el mensaje es sugerido por SourceTree:



Ahora tenemos dos commits pendientes de enviar, pero sabemos que no van a entrar en conflicto.



Una vez subido el cambio, vemos la sucesión de commits y como se han vuelto a unir en una misma rama.

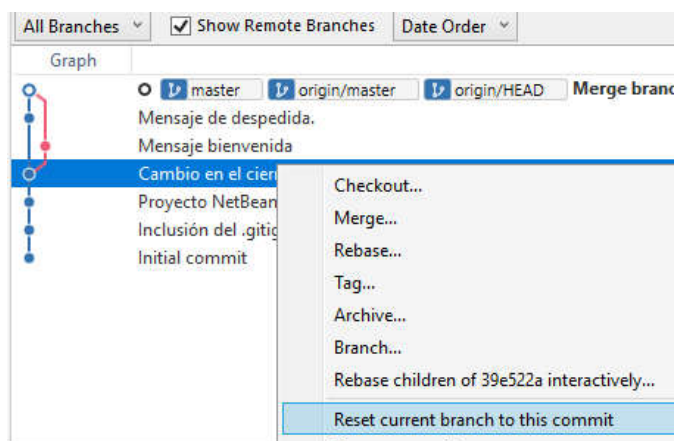


6. Operaciones avanzadas

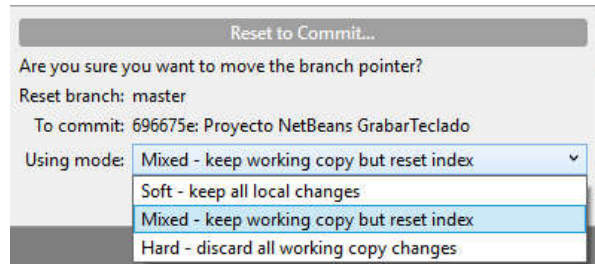
- **rama:** línea activa de desarrollo. El commit más reciente de una rama se denomina la punta de dicha rama. La punta de la rama se referencia por medio de una cabeza. La copia de trabajo está siempre asociada a una rama (la rama "actual" o "checked out") y la cabeza especial "HEAD" apunta a esa rama.
- **merge:** fusionar los contenidos de otra rama (potencialmente desde un repositorio externo) en la rama actual. Si la rama es de otro repositorio, primero se hace un fetch* de la rama y después se fusiona en la rama actual. La fusión puede crear un nuevo objeto commit si una de las ramas no es un ancestro de la otra. Si una es ancestro de la otra, simplemente se mueve la referencia de la cabeza de la rama fusionada (fast-forward merge).

6.1.Cambiar de commit

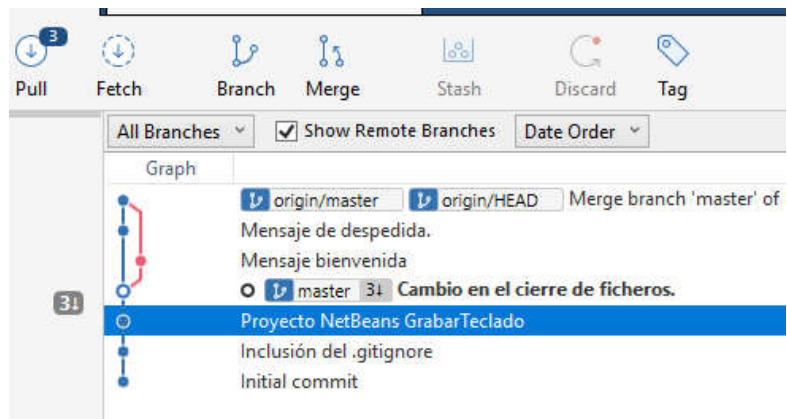
Dado que Git guarda todos los cambios podemos ir a cualquier commit que esté almacenado en el repositorio, supongamos que no deseamos los dos últimos mensajes (bienvenida y despedida) y queremos volver al código en el commit justo anterior, pulsando con el botón derecho sobre dicho commit y seleccionando *Reset current Branch to this commit*:



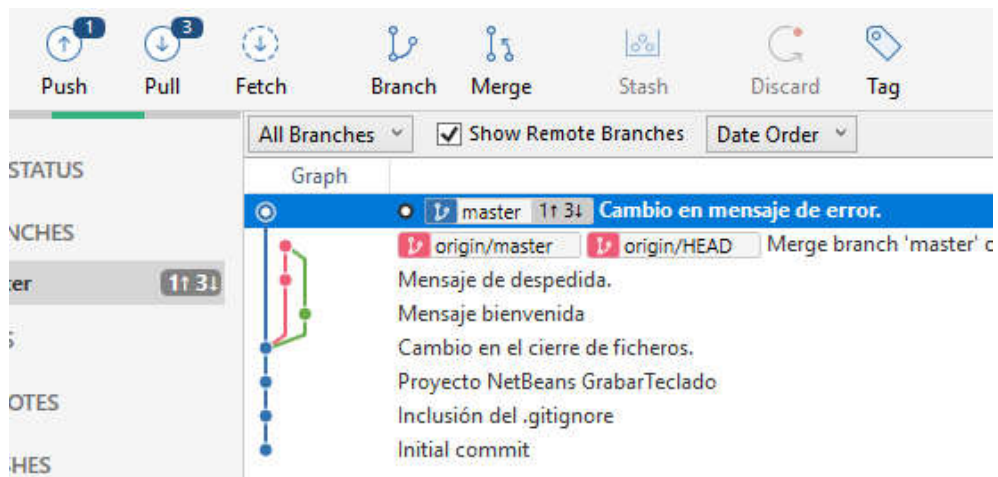
Tenemos tres opciones a elegir en función de si queremos perder o no los cambios locales, en este caso vamos a elegir la opción Hard (el código cambiará y tendremos las fuentes exactas del commit elegido):



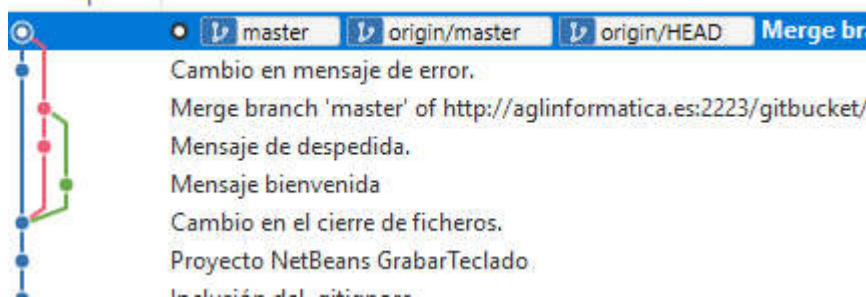
La nueva situación de nuestra rama, estamos tres commits por detrás del repositorio remoto:



En este caso si realizamos algún cambio vemos que estamos en una nueva rama, si deseamos subir nuestros cambios vamos a tener que realizar un merge con la rama master remota.



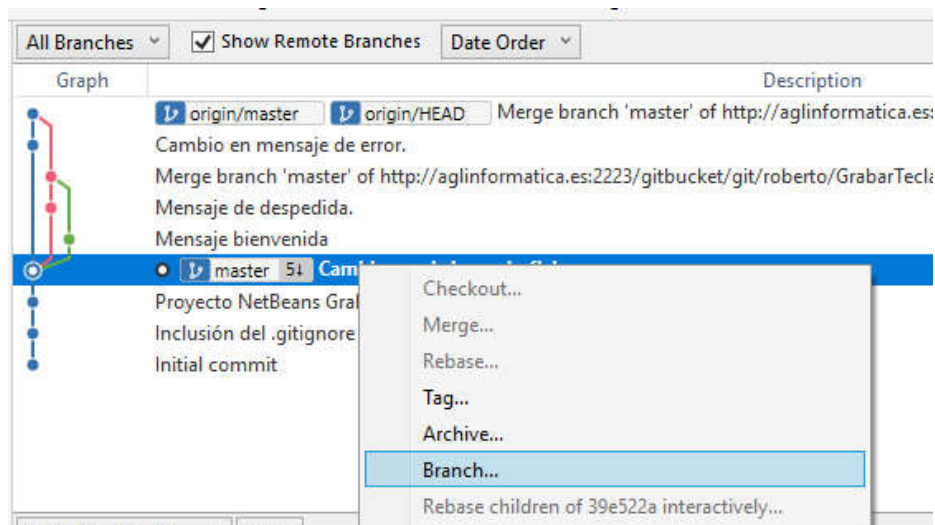
Realizaremos primero el Pull (se integraran los 3 commits) y posteriormente el Push que pasará de 1 a 2 commits para incluir el merge.



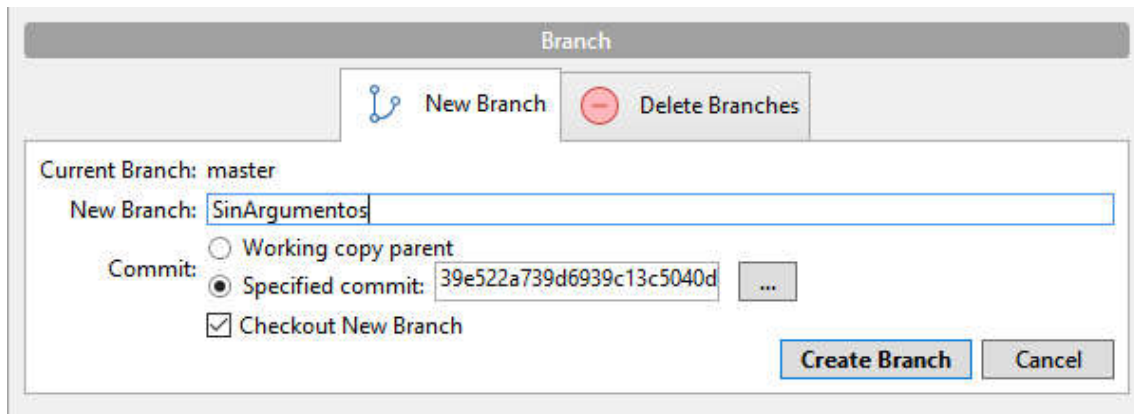
6.2.Creación de una nueva rama

Si deseamos realizar alguna prueba con el código que no sabemos si va a funcionar y no queremos realizarla en la rama principal, tenemos la opción de crear una rama separada en la que desarrollar por separado.

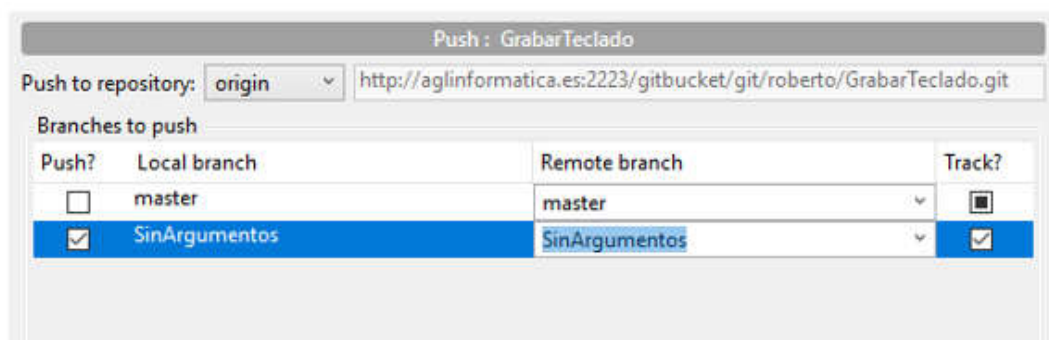
En este caso he vuelto a un commit anterior y pulsado con el botón derecho tenemos la opción de crear una nueva rama *Branch*.



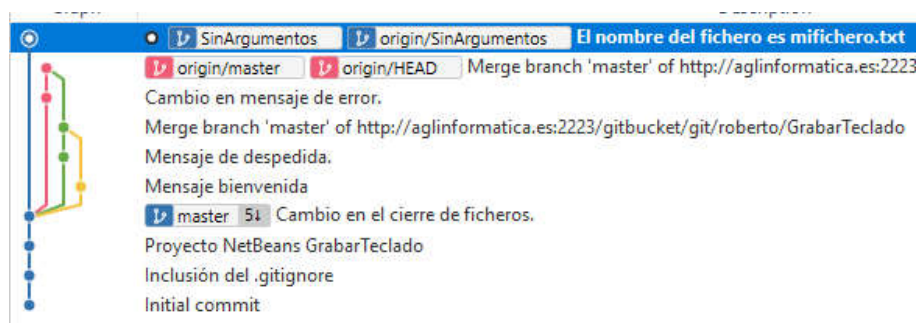
Nos solicitará el nombre de la nueva rama y cuál es el commit de la que partirá.



Tras realizar algún cambio en el código en la nueva rama, deseo enviar los cambios al repositorio remoto (que aún no sabe nada de la nueva rama), le indico de que rama voy a enviar los cambios para proceder a realizar el Push:



Como resultado tengo la nueva rama SinArgumentos en el repositorio remoto (origin).



Ahora cualquier otro usuario puede descargarse dicha rama y trabajar sobre ella.

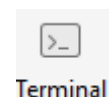
En cualquier momento puedo cambiar de rama a través del interfaz de SourceTree



6.3. Como establecer la rama origin/master a un commit concreto

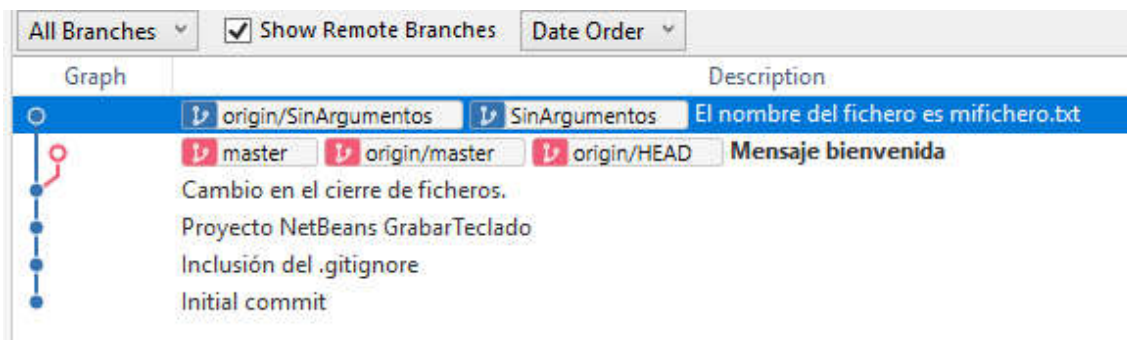
Git incluye muchos comandos que permiten configurarse mediante parámetros, por ello no es difícil encontrar situaciones donde la interfaz gráfica de SourceTree (u otros) no permite realizar la operación que deseamos para ello tenemos a nuestra disposición la consola/terminal donde podremos introducir los comandos de Git.

Supongamos que queremos volver al commit del mensaje de bienvenida, para ello tendremos que ir a la consola y teclear :



```
git checkout master
git reset --hard id_del_commit
git push -force origin master
```

Como resultado :

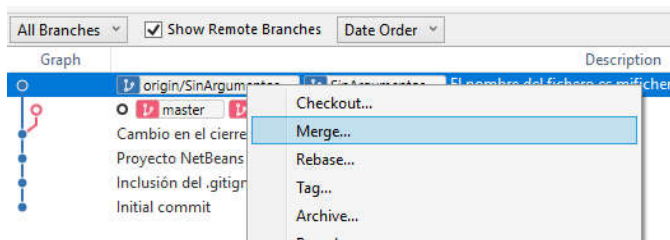


Hay que tener en cuenta que los repositorios locales de otros usuarios verán como cambia la rama master en el repositorio remoto pero tendrán varios push pendientes con sus cambios, si se desea que todos estén en ese commit tendrán que resetear su rama al commit correspondiente.

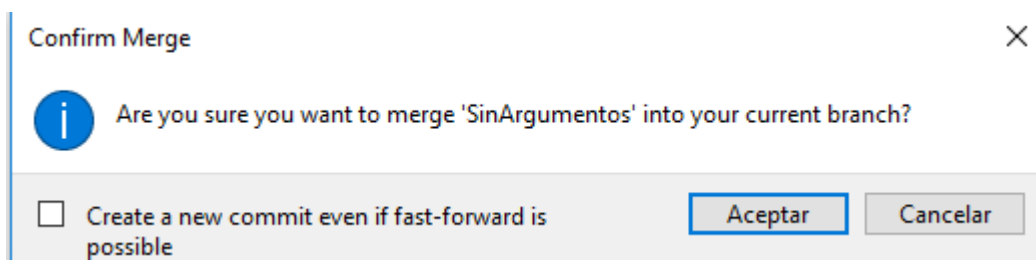
6.4. Merge

Con merge podemos juntar dos ramas en una, sería la operación contraria al branch, vamos a verlo en un ejemplo, unamos las dos ramas que teníamos en una sola.

Nos situamos sobre la rama SinArgumento y con el botón derecho pulsamos Merge:



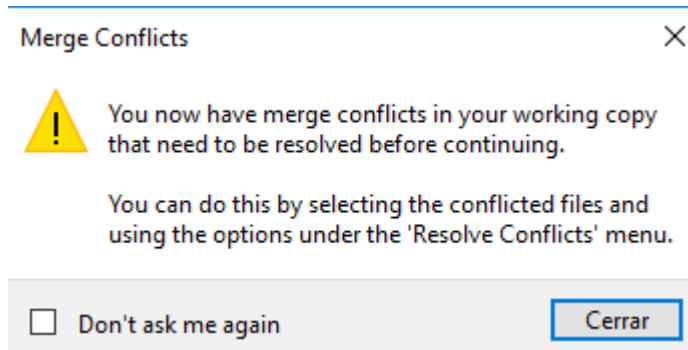
Nos preguntará si deseamos realizar un Merge sobre la rama actual:



Nuestra rama actual es master, para cambiar de rama deberíamos hacer un checkout de la misma.



Al realizar el merge se han generado conflictos, vamos a resolverlos:



```

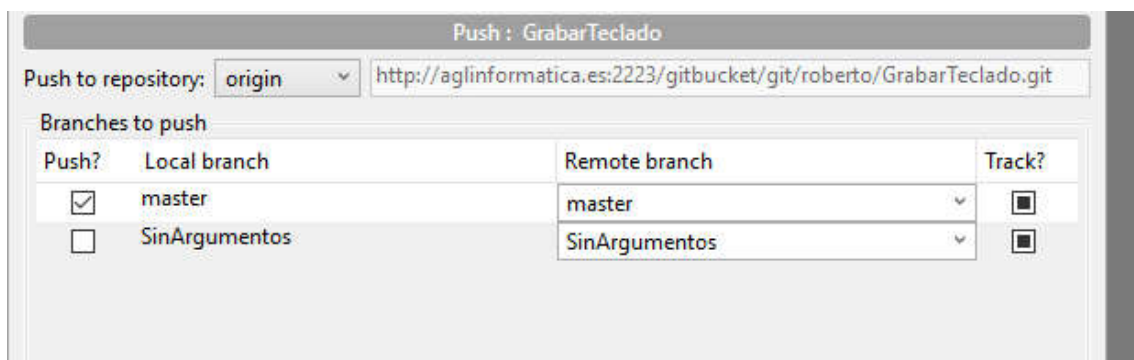
<<<<<<< HEAD
System.out.println("Bienvenidos al programa");

if (args.length == 0 ) {
    System.out.println ("Error. Debes pasar como argumento el fichero "
        + " de salida. No te olvides de expresar la ruta absoluta");
    System.exit(-1);
}

=====
>>>>>>> SinArgumentos
//Objetos File
fileOut = new File("mifichero.txt");
  
```

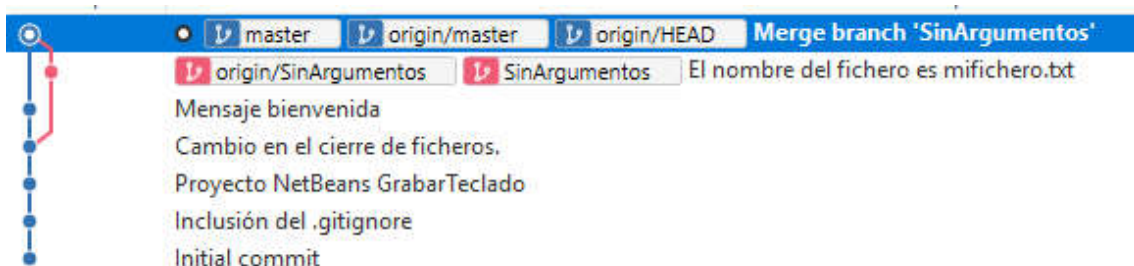
Vemos en verde las líneas que han cambiado, y en azul las que se han insertado sin conflicto.

Tenemos que modificar el fichero y realizar un commit con los cambios.



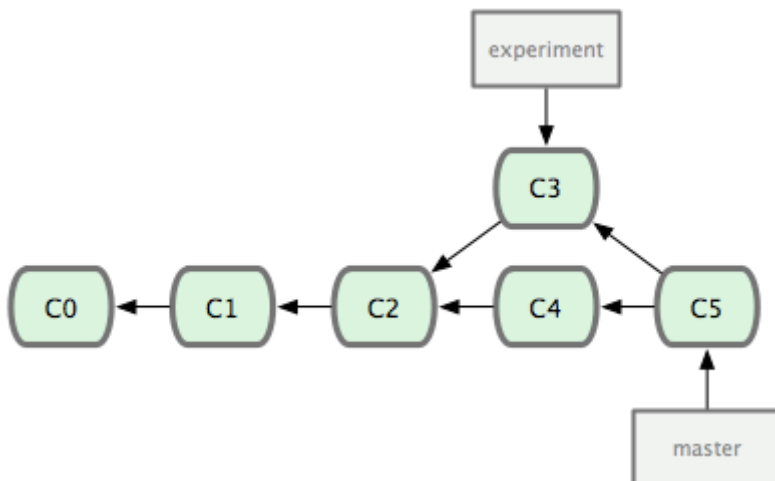
Subimos los cambios mediante un Push al repositorio remoto en la rama que deseemos.

Las ramas están unidas y los dos repositorios están en el mismo commit.

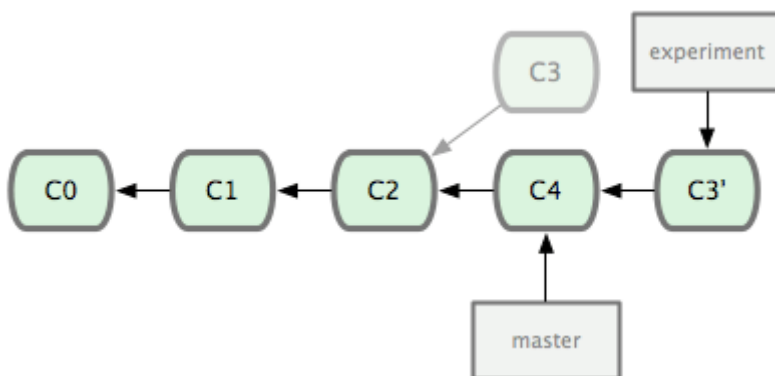


6.5.Rebase

Para fusionar dos ramas podemos hacer un merge, como en la siguiente imagen:



O utilizar un Rebase :



Vemos que en este caso se aplican los cambios de C3 sobre C4 perdiendo en el camino el commit de C3.

7. Para saber más

En este documento hemos repasado algunos conceptos básicos de Git, la casuística que genera la gestión de proyectos en el estado de los archivos es muy alta, y será habitual que nos encontremos con situaciones en las que tengamos que recurrir a la documentación para resolver alguna situación.

Para un conocimiento más exhaustivo de Git recomiendo la lectura del siguiente libro <https://git-scm.com/book/es/v1>, en la web podréis encontrar videos y guías para la utilización de Git.

Algunas imágenes de este documento han sido obtenidas de:

Fuente: <https://git-scm.com/book/es/v1>.

Licencia MIT