

INTRODUCCIÓN

Supongamos que dos programadores trabajan juntos en un mismo código, cada uno en su ordenador. ¿Qué ocurre si los dos tienen que modificar el mismo archivo? ¿Cómo nos podemos asegurar de que nuestro código no se pierda nunca? ¿Y si quiero volver a una versión anterior del código? ¿Y si quisiera saber toda la historia de cambios de esos archivos? ¿Y si se nos rompe el disco duro?

Para solucionar todos estos problemas, existen los **Sistemas de Control de Versiones**.

- Se llama control de versiones a los métodos y herramientas disponibles para controlar todo lo referente a los cambios en el tiempo de un archivo. Un código necesita cambios o modificaciones para corregir errores, modificar su contenido... A medida que el documento cambia existen dos opciones, mantener un historial de cambios o dejar que evolucione sin memoria. El control de versiones es una forma de mantener “esta memoria” haciendo además que sea útil para el desarrollo futuro.
- Es decir, son herramientas software que gestionan los cambios realizados en el código fuente de los proyectos. Gracias a estas herramientas, los desarrolladores pueden consultar y comparar versiones anteriores de sus proyectos.
- Podremos controlar los cambios, recuperar versiones anteriores, saber quien hizo cada cambio y cuando...
- Ejemplos: CVS, Subversion, SourceSafe, ClearCase, Darcs, Bazaar, Git, Mercurial, Perforce..

Los Sistemas de Control de Versiones trabajan con **repositorios**.

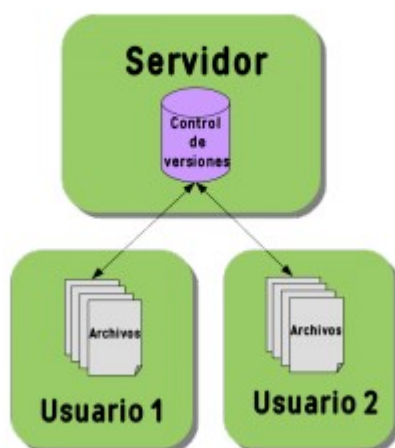
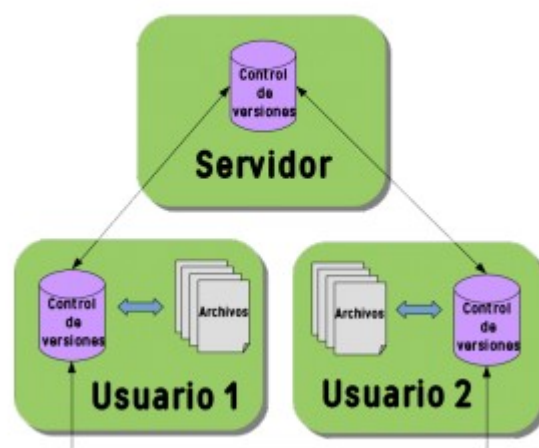
- Un repositorio es un espacio digital centralizado que los desarrolladores utilizan para realizar y administrar cambios en el código fuente de una aplicación. Cada miembro del equipo puede focalizarse en un archivo diferente y trabajar de forma simultánea. Estos sistemas están preparados para resolver conflictos en caso de que más de un desarrollador esté trabajando sobre el mismo archivo.

Sistemas de control de versiones Centralizado

- Existe un único repositorio centralizado.
- Un ejemplo de Sistemas de Control de Versiones Centralizados es Subversión. Partimos de una versión determinada, por ejemplo de la versión 1 y tenemos tres ficheros, A, B y C.
- En la versión 2, almacenamos los cambios, como si fueran esa especie de incrementos. De esta manera, por buscar algún símil, es como si Subversión trabajase con backups incrementales.
- El almacenamiento de datos se realiza como cambios en la base de cada archivo.

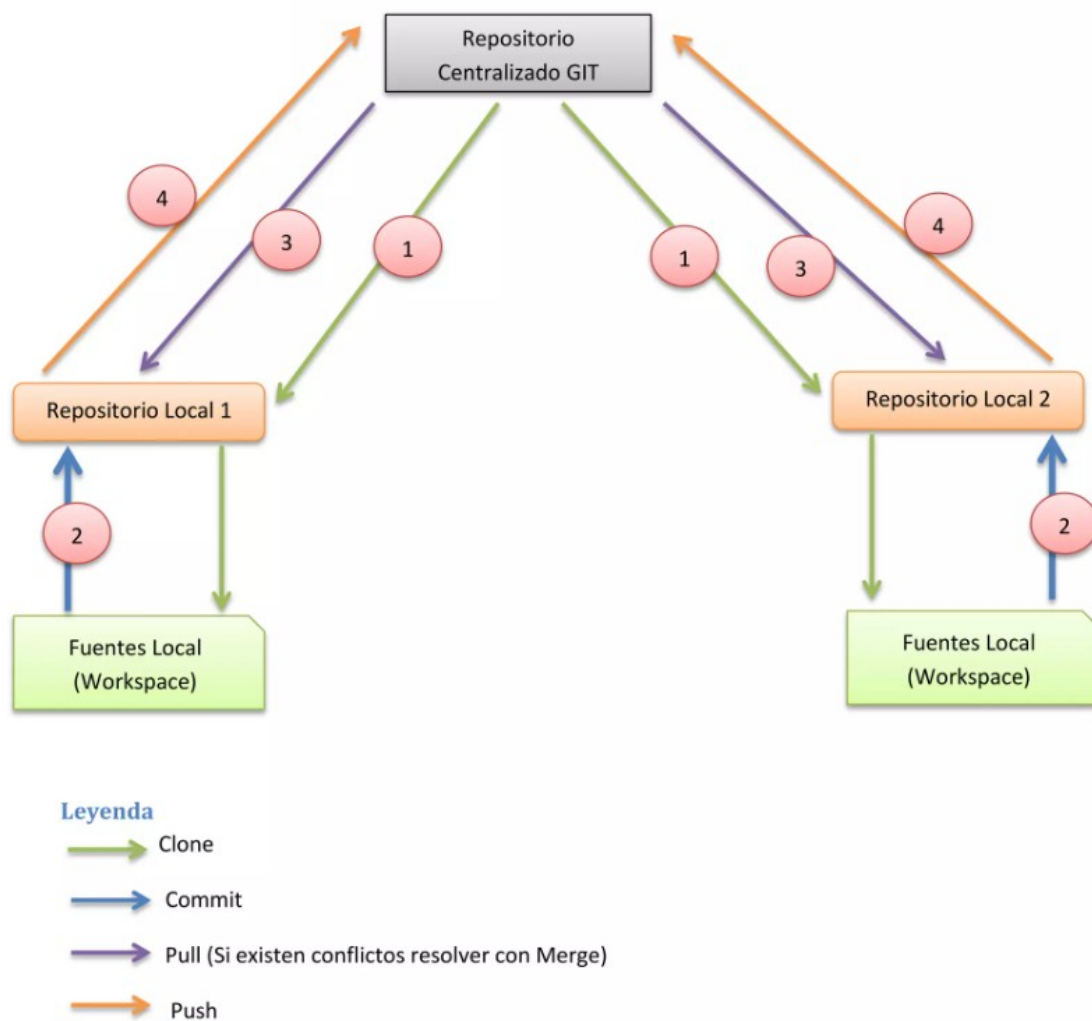
Sistemas de control de versiones Distribuido

- Cada usuario tiene su propio repositorio, tiene un control de versiones propio que a su vez son manejadas por el servidor.
- Si el repositorio remoto se cae, podemos seguir trabajando.
- Han sido diseñados para el uso de ramas tanto en repositorio remotos como locales.

Modelo centralizado**Modelo distribuido****¿Qué es Git?**

- Sistema de control de versiones distribuido frente a los centralizados como por ejemplo SVN.
- Creado por Linux Linux Torvalds (comunidad Linux) en 2005. Desde entonces, su evolución ha sido constante, ha mejorado con cada versión y actualmente es un sistema fiable, robusto, rápido y potente incluso con proyectos de gran tamaño.
- Gestión eficiente de proyectos grandes
- Facilita el acceso a las diferentes revisiones del código.
- Sistema de trabajo en ramas que permite disponer de proyectos divergentes y tener líneas de progreso diferentes a la rama principal.

Introducción Flujo GIT



¿Cómo funciona?

- El código se sube a un repositorio remoto. Si ya estuviera creado, tendríamos que clonarlo en local (**Clone**)
- Bajamos una copia local (**pull**)
- Trabajamos con nuestra copia local y hacemos **commit**, para tener un registro de historio local.
- Después de hacer nuestros cambios en local, actualizamos la copia local (pull)
- Subimos los cambios al repositorio remoto (**push**)

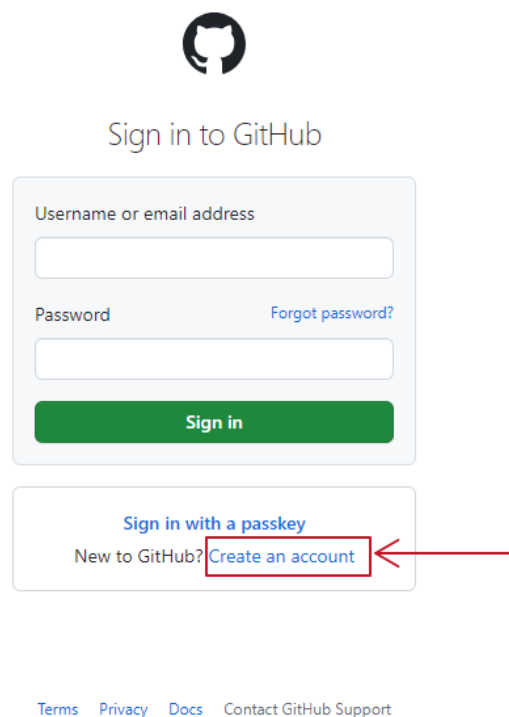
GitLab y GitHub

- Son servicios que ofrecen control de versiones y desarrollo de software colaborativo basados en GIT
- Ofrecen alojamiento de repositorios, gestión documental, sistema de seguimiento de incidencias, ...
- En ambos casos se ofrece la posibilidad de alojar código fuente de forma pública o privada.
- GitLab ofrece la posibilidad de poder disponer de repositorios privados en servidores locales, de forma gratuita.
- GitHub permite disponer de repositorios privados, mediante planes de pago, pero en espacios de su nube.

Instalación

1.- Descargar git: <https://git-scm.com/downloads>

2.- Crear cuenta en gitHub o gitLab: <https://github.com/login>



Sign in to GitHub

Username or email address

Password [Forgot password?](#)

Sign in

[Sign in with a passkey](#)

New to GitHub? [Create an account](#)

[Terms](#) [Privacy](#) [Docs](#) [Contact GitHub Support](#)

3.- **Descargar Tortoise Git:** <https://tortoisegit.org/download/>

OJO: seleccionar conexión ssh git-server.

Crear repositorio en GitHub

Accedemos a nuestra cuenta de gitHub y creamos un repositorio. Indicamos el nombre del repositorio.

El fichero README suele tener descripciones importantes sobre el proyecto. Crear repositorio.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * btudo / Repository name * ED_ProyectoEjemplo
✔ ED_ProyectoEjemplo is available.

Great repository names are short and memorable. Need inspiration? How about [reimagined-meme](#) ?

Description (optional)

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

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

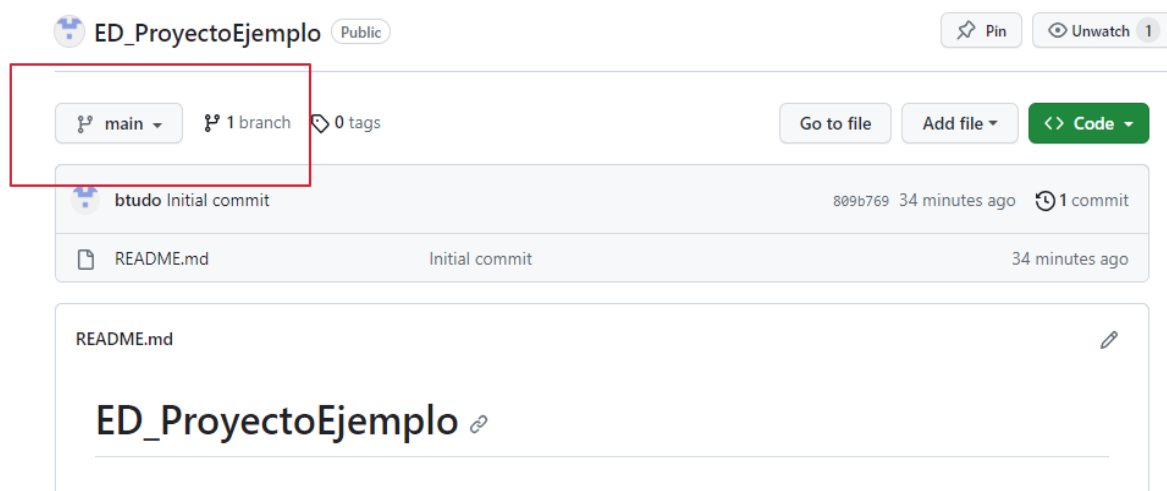
Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

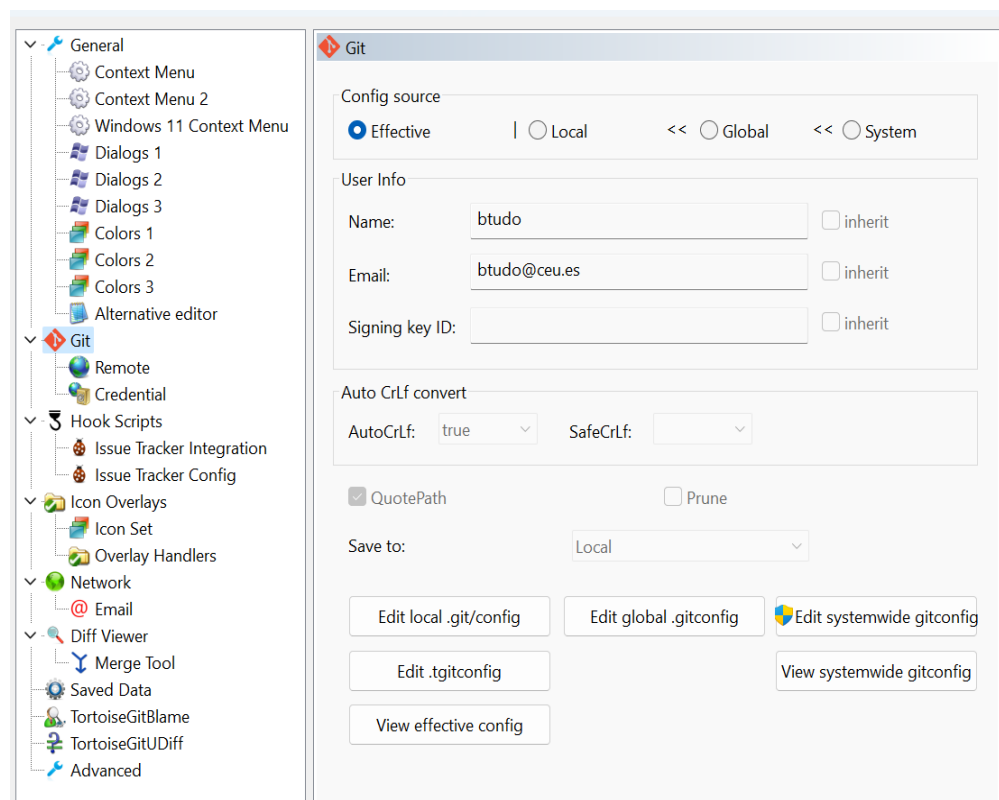
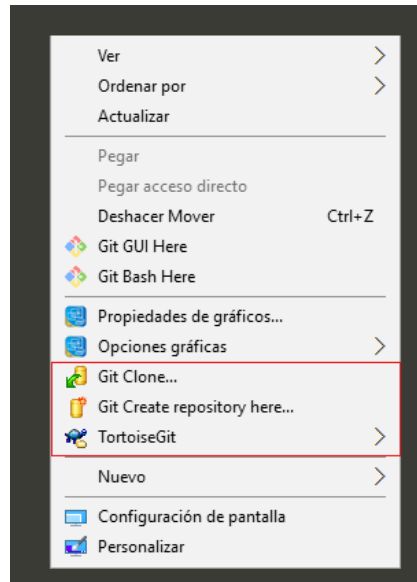
.gitignore template: None

El repositorio se crea por defecto en una rama principal que se llama **main**.



Uso de TortoiseGit

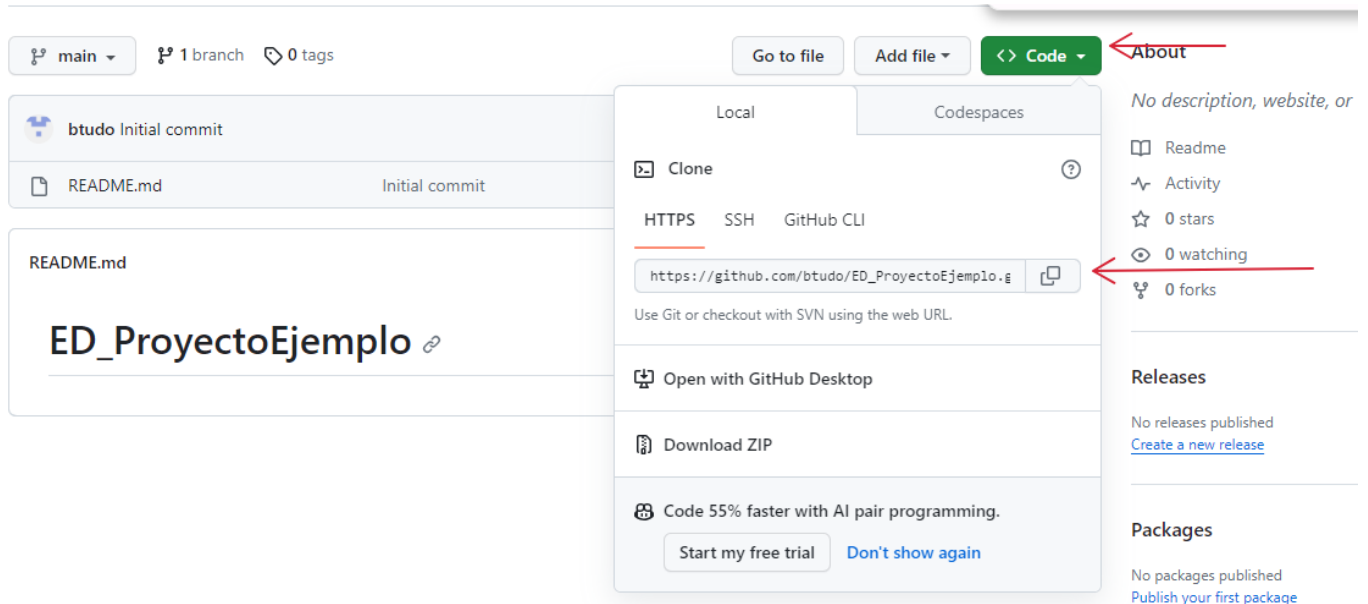
Vamos a configurar nuestro usuario. Si pulsamos botón derecho, sobre TortoiseGit, vamos a settings:



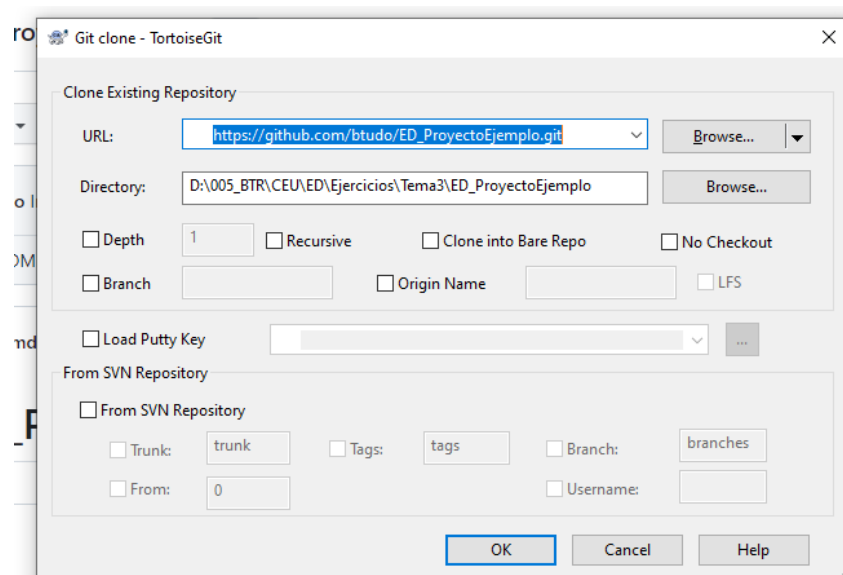
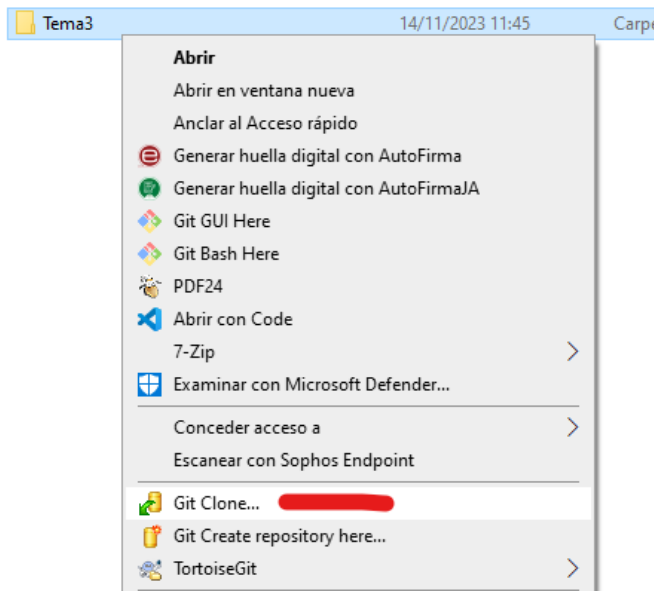
Tenemos que guardar el usuario y correo electrónico asociado a nuestra cuenta de Github.

Cómo clonar un repositorio en local

Sobre nuestro repositorio en github copiamos la url donde se encuentra:



Nos ubicamos en la carpeta de trabajo. Si pulsamos botón derecho, seleccionamos **Git Clone**:

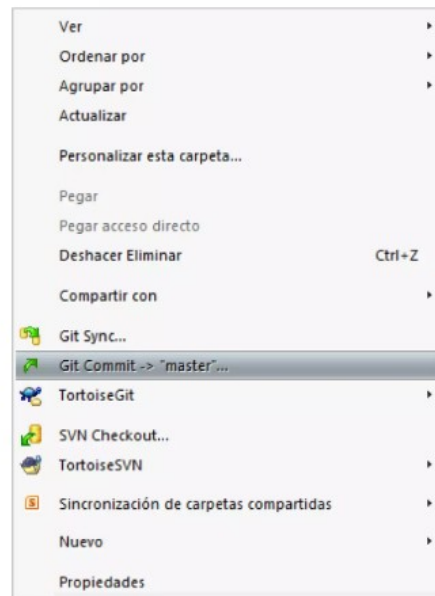


Comprobamos cómo se ha clonado el repositorio en mi carpeta local. Dentro, tendrá un .git y el único archivo que había en el repositorio que es README.md

Vamos a modificar el fichero README, y escribimos alguna descripción en él.

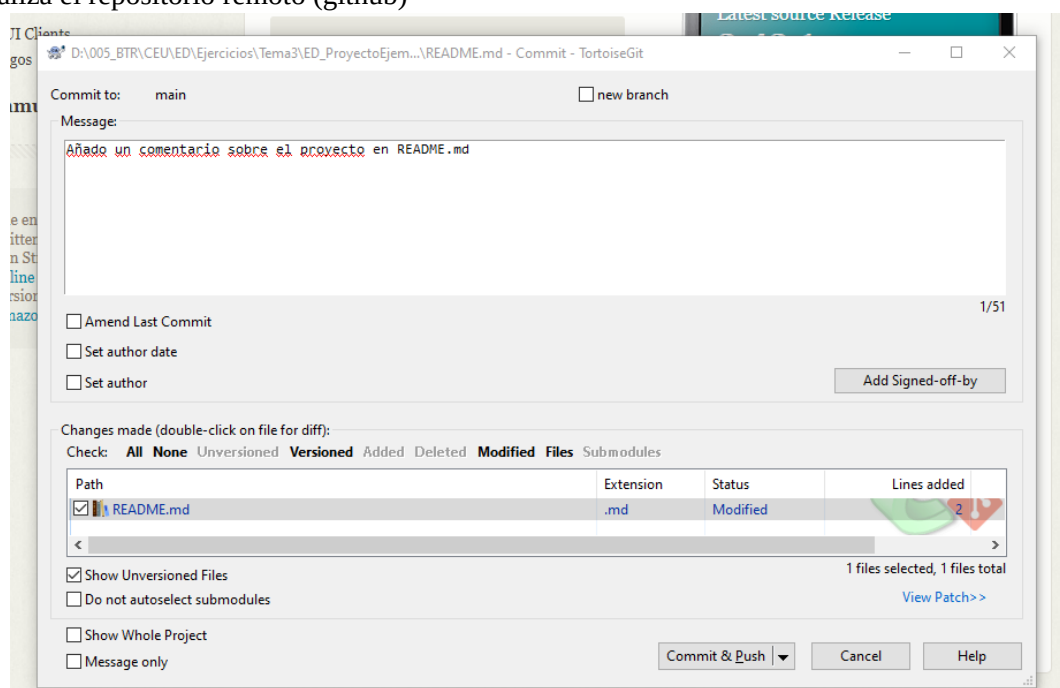
Nombre	Fecha de modificación	Tipo	Tamaño
.git	14/11/2023 12:06	Carpeta de archivos	
README.md	14/11/2023 12:10	Documento MD	1 KB

Pulsamos botón derecho, y seleccionamos: **Git Commit**.



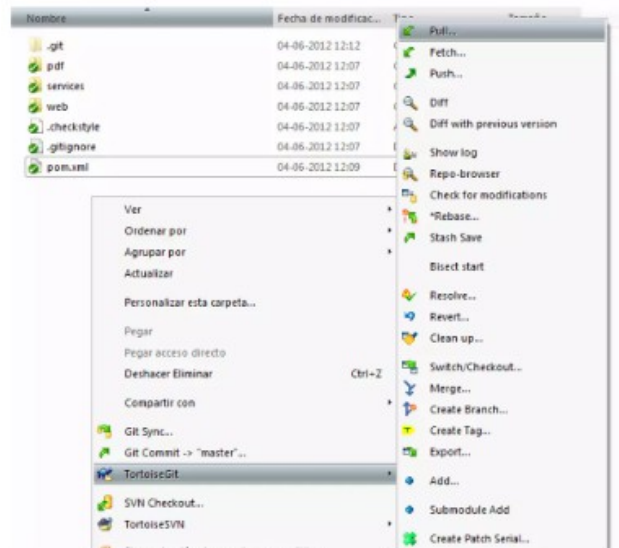
Commit: Actualiza mi repositorio local

Push: Actualiza el repositorio remoto (github)



Después de guardar cambios, en el repositorio podemos comprobar cómo está el fichero README actualizado, junto con los comentarios, fecha de modificación y usuario que modificó el fichero.

IMPORTANTE: Cada vez que tengamos que trabajar con código fuente, debemos estar sincronizados y es necesario hacer un **PULL**.



Ejercicio: En la carpeta de trabajo, vamos a añadir una imagen que nos descargemos. Subirla al repositorio remoto y comprobar que está subida correctamente.

¿Y si quiero eliminar la imagen?

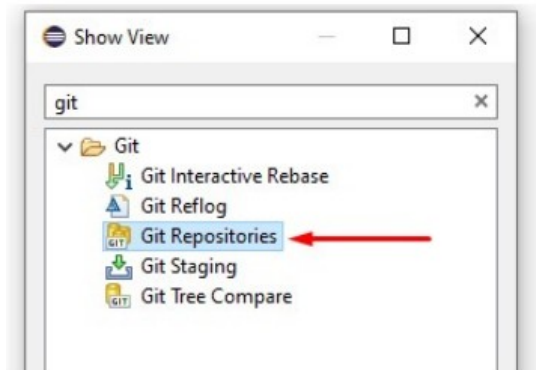
Git desde eclipse

NOTA: Para que desde eclipse nos deje hacer cambios en nuestro repositorio, tenemos que generar un token.

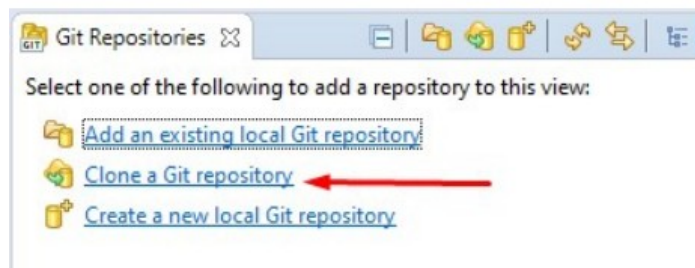
Para ello, en nuestra cuenta de gitHub / Settings / Developer Settings / Personal Access Token / Tokens, generamos un nuevo token. Copiar el token generado.

Ahora vamos a **clonar** nuestro repositorio. Id a gitHub para copiar la url del repositorio.

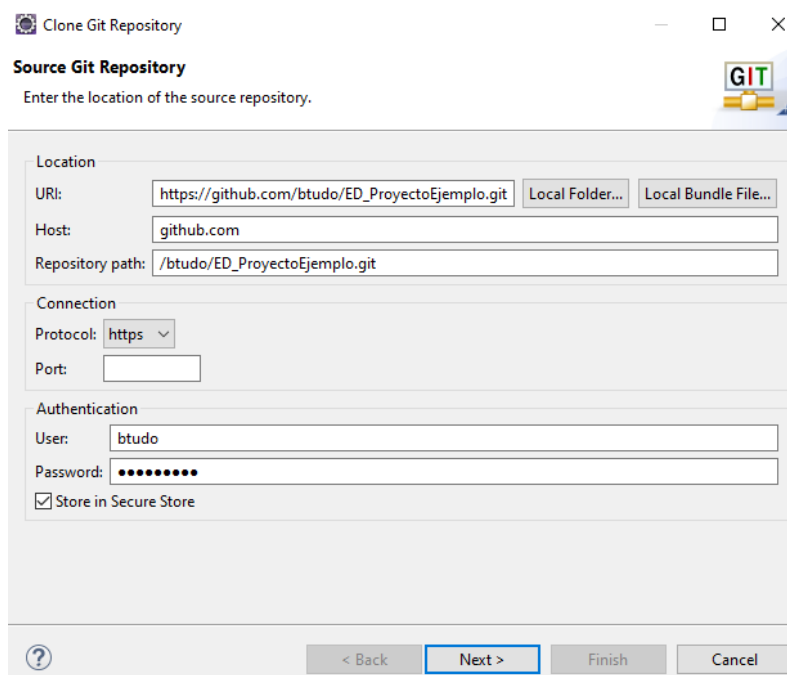
Abrimos eclipse, y en nuestro workspace, seleccionamos: Window / Show View / Git Repositories



Escogemos clonar un repositorio:

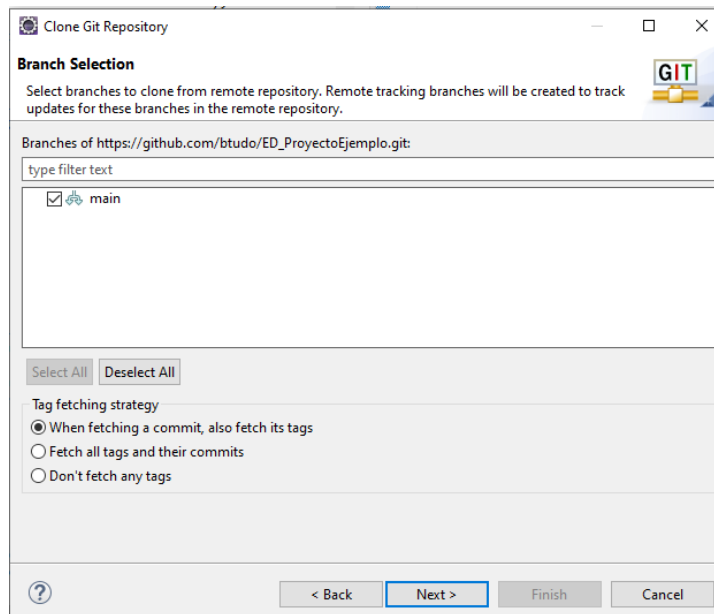


Indicar los datos de nuestro usuario y contraseña de GitHub. Marcar que se almacene para que no tengamos que repetirla cada vez que queramos trabajar con el repositorio:



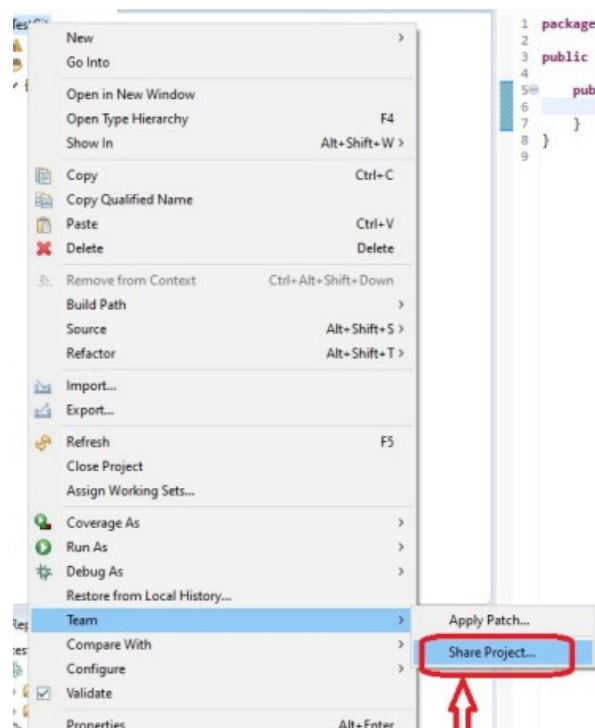
En password ponemos el token que hemos guardado en el primer paso.

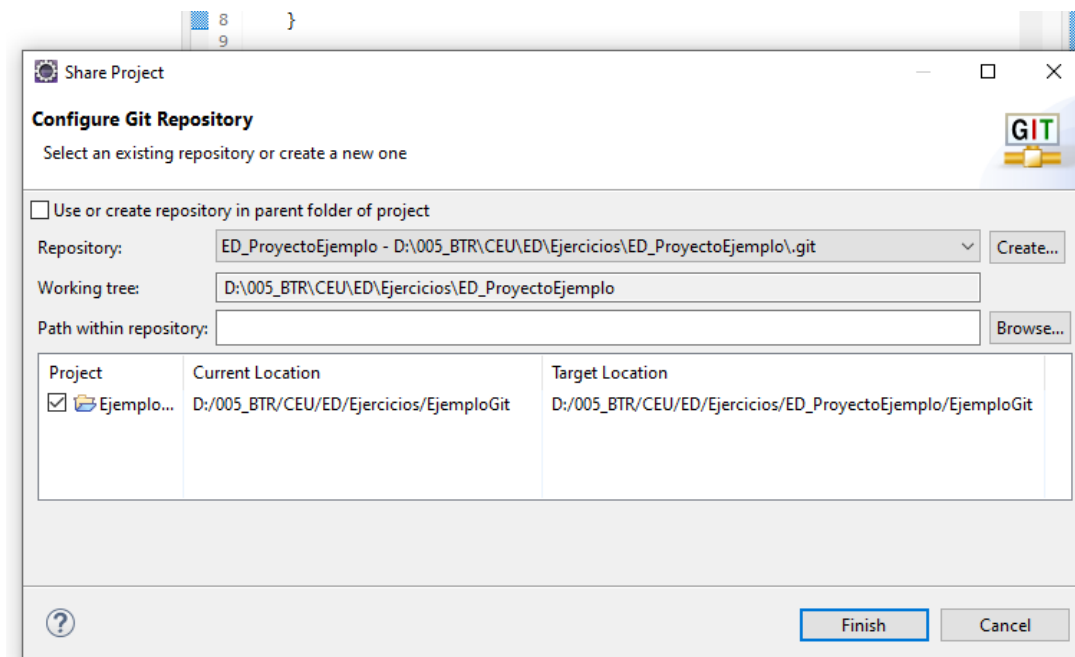
A continuación indicamos qué rama clonar. Seleccionamos main:



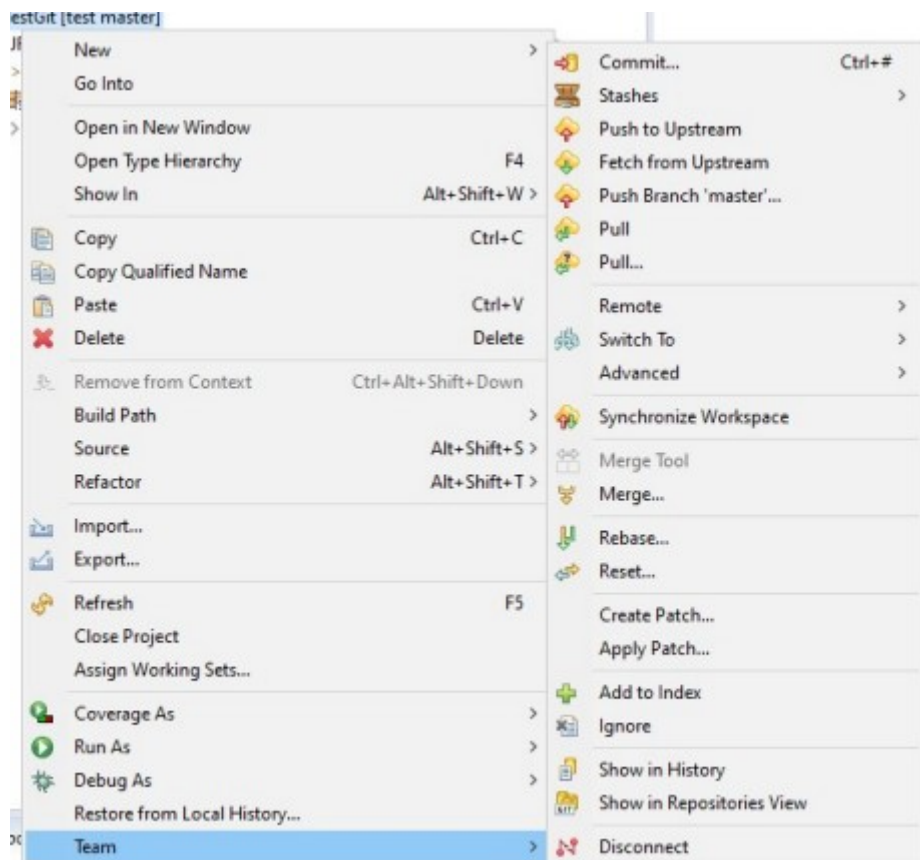
Guardamos dónde queremos copiar el repositorio. Se recomienda dentro del workspace de eclipse.

A continuación nos creamos un proyecto Java en eclipse con una clase MiClase.java, que imprima un mensaje por consola. Para indicarle a Eclipse que nuestro proyecto esté dentro del repositorio GIT, sobre el proyecto, damos a botón derecho: Team / Share Project.



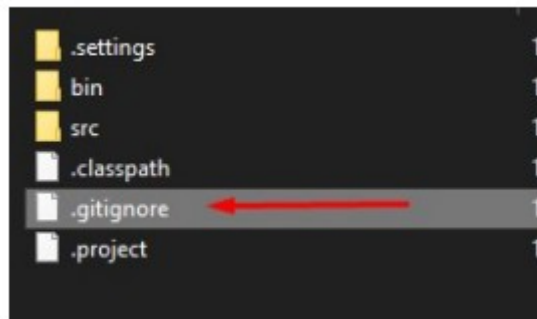


A partir de este momento sobre el proyecto, dando al botón derecho, en Team, podremos hacer cualquier operación sobre el repositorio GIT: push, pull, commit, etc.



Git Ignore

Es un fichero que se encuentra en la carpeta raíz del proyecto, llamado `.gitignore` donde se configura qué carpetas y tipos de ficheros no se suben al repositorio. Ejemplo: Los `.class` que se encuentran en la carpeta `bin` nunca se suben al repositorio.

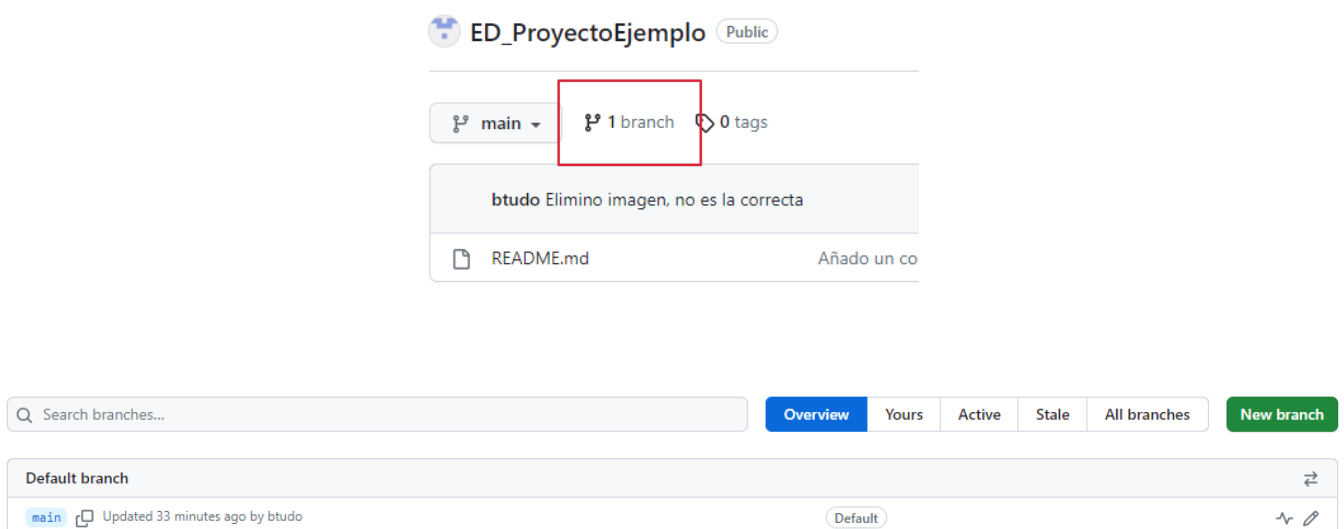


Podemos incluir los ficheros y carpetas que no queremos que se suban al repositorio nunca:

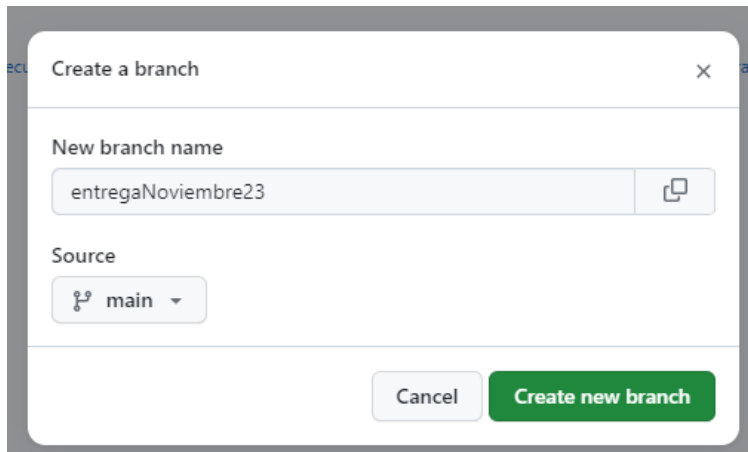
```
1 /bin/
2 .settings
3 .classpath
4 .project
5
```

Trabajar con ramas

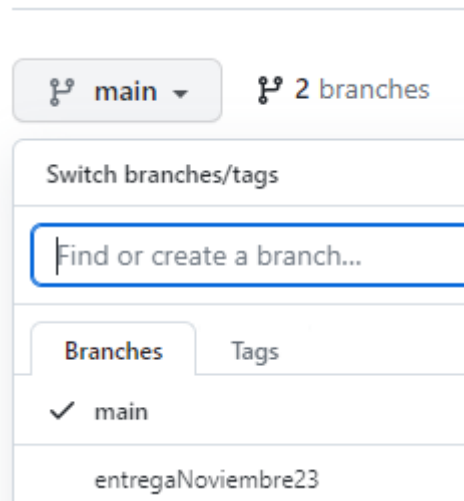
Por defecto, la rama principal de un repositorio en GitHub es `main`. Pero podemos tener varias ramas y poder fusionarlas cuando queramos.



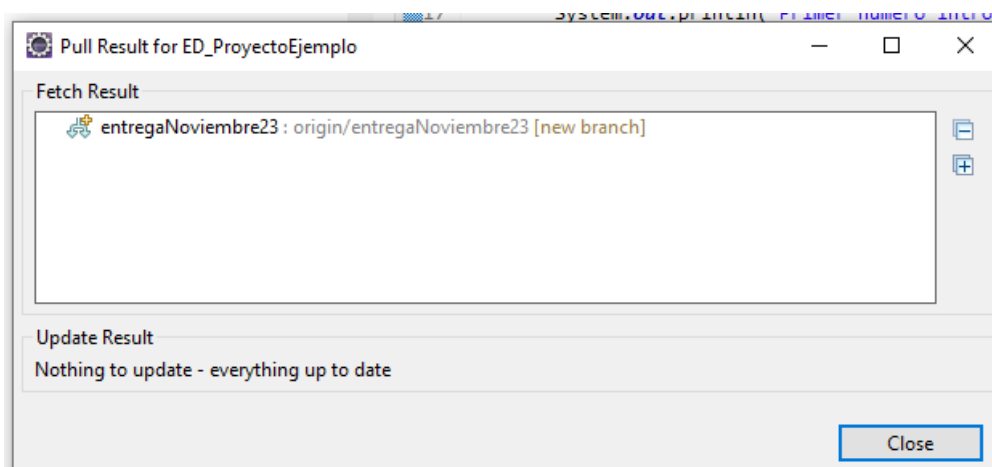
Para crear una rama, se parte de otra, en este caso elegimos main:



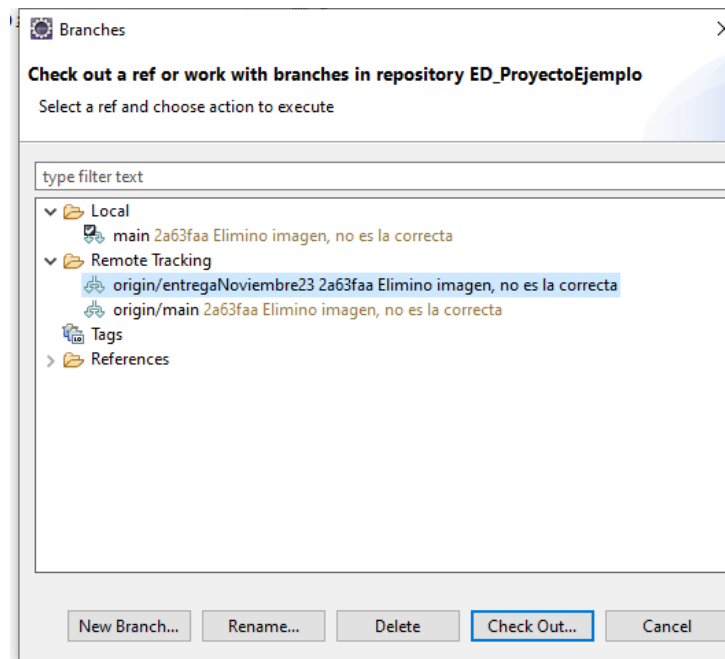
Vemos cómo se ha creado otra rama en el repositorio:



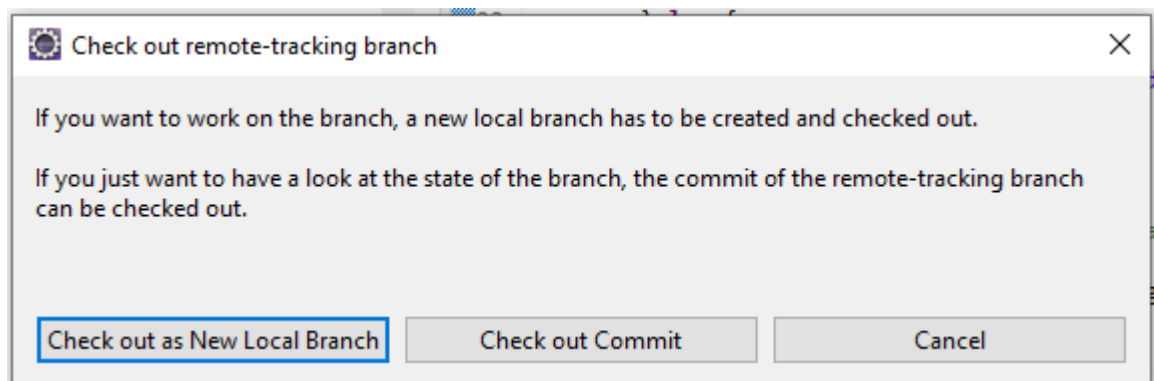
Para traernos la rama nueva, tenemos que hacer **pull** del repositorio en eclipse.



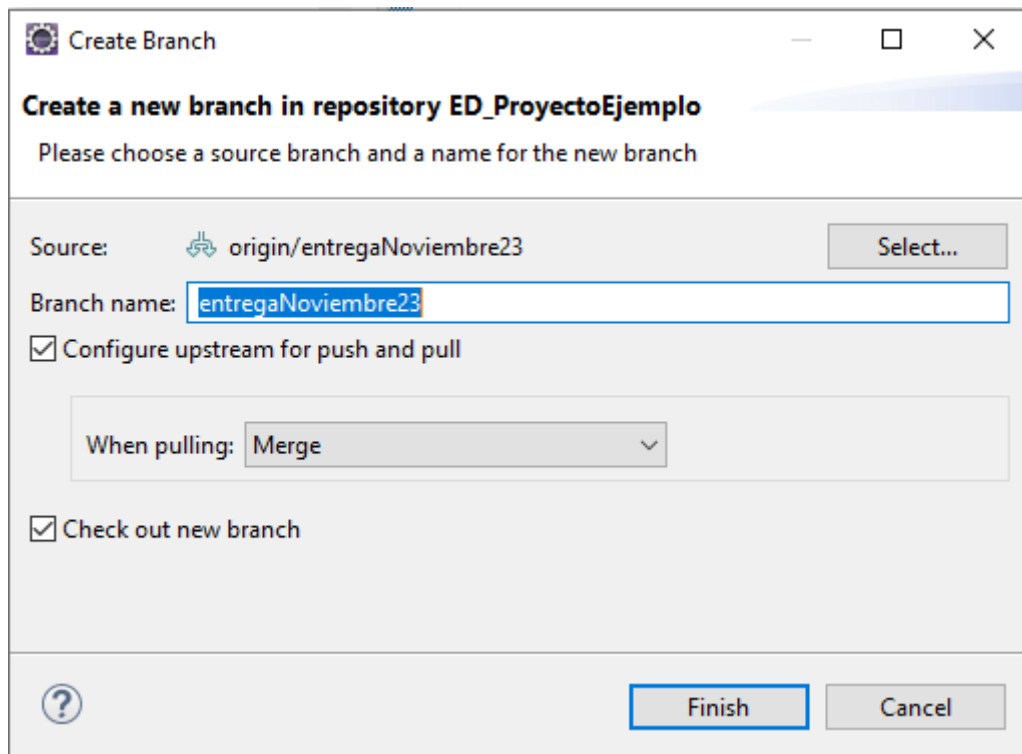
Luego, podremos cambiar en nuestro proyecto para que trabaje con una rama u otra seleccionando sobre el proyecto: Team / Switch to / Other.



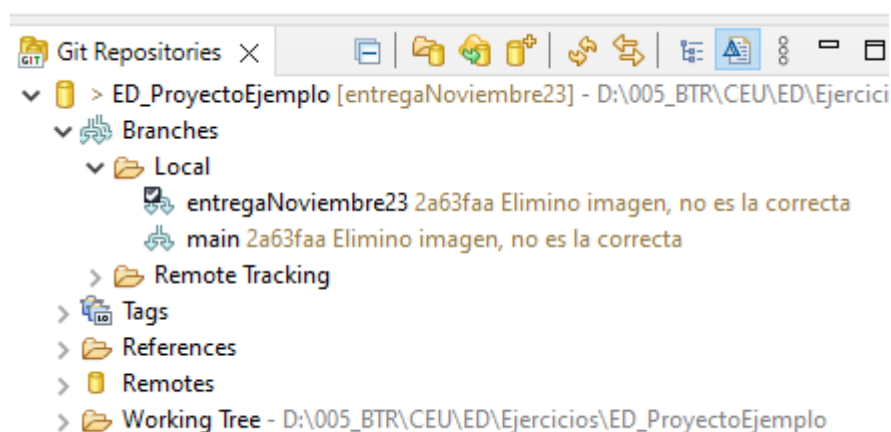
Seleccionamos la nueva rama, y también le indicamos que cree la rama nueva en el repositorio local.



Es recomendable que la rama tenga el mismo nombre en el repositorio local y remoto.



Tras esto, ya tenemos la rama nueva y la podemos ver el visor del repositorio. Se puede cambiar a otra haciendo doble click.



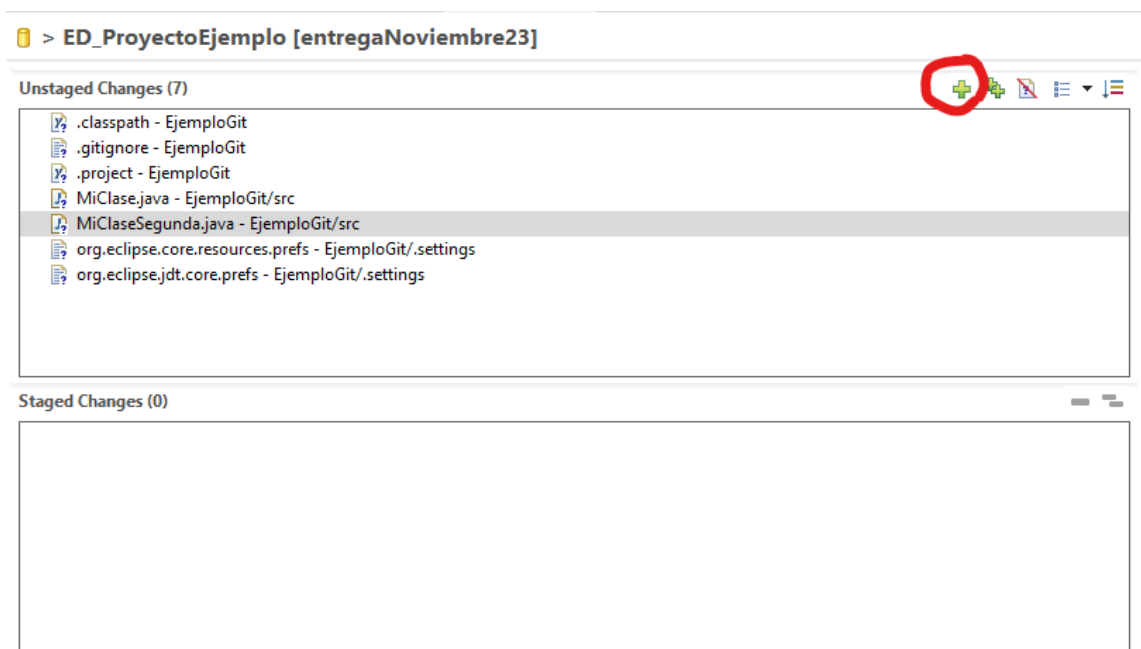
Vamos a subir cambios a la nueva rama.

Fusionar ramas

Sobre nuestro proyecto en eclipse en nuestra nueva rama, vamos a crear otra clase: MiSegundaClase.java, que muestre otro mensaje por consola.

Hacemos un commit: Esto guarda ese nuevo cambio en el repositorio local

Sobre el proyecto, botón derecho: Team / Commit



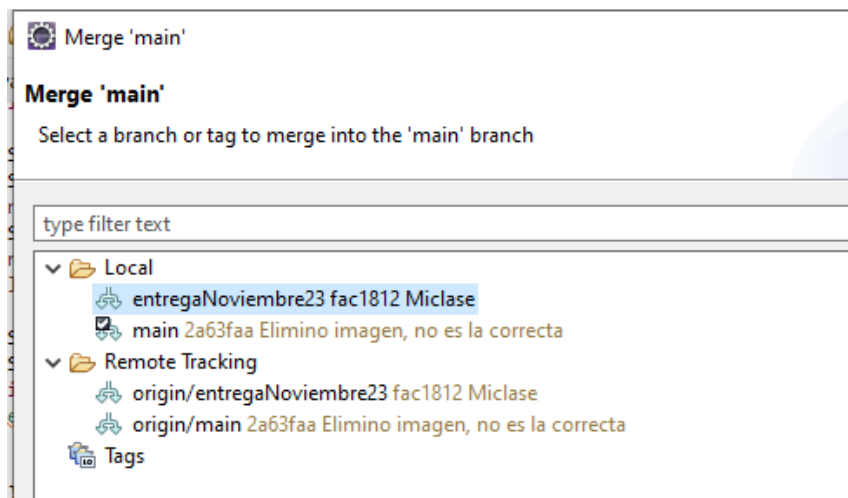
Seleccionamos los ficheros que queremos subir y le damos al icono del +. Una vez que las dos clases esten añadidas en la ventana de Staged Changes, escribimos un mensaje y pulsamos **Commit y Push**

Si sólo queremos actualizarlo en el repositorio local, pulsamos el botón **commit** solamente.

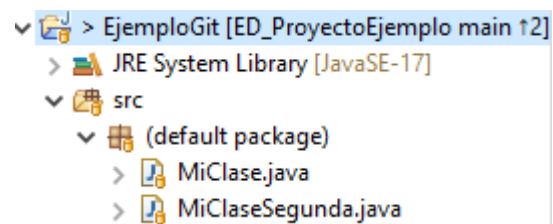
Una vez que tengamos las ramas en nuestro repositorio, podemos fusionarlas, de tal forma que todo lo que tenga la rama nueva la pasemos a la rama main. Para ello, en eclipse, nos posicionamos sobre la rama main. Podemos hacerlo o bien Team / Switch to .. main, o bien dando doble click sobre la rama main en el visor del repositorio.

Una vez posicionados en la rama main, seleccionamos botón derecho, Team / Merge.

Seleccionamos en local, la rama que queremos fusionar:



Vemos cómo en eclipse, ya ha incorporado las clases de la nueva rama en main:



Por último hacemos **push** para que los cambios se actualicen en el repositorio remoto.

