



GIT y GitHub.

Control de versiones





¿Qué es GIT?

Git es una herramienta de control de versiones. Permite controlar el proceso de creación de software llevando un registro exhaustivo de todos los cambios realizados.

Ofrece la posibilidad de crear versiones, de ramificar un proyecto en diferentes flujos e incluso de volver hacia atrás deshaciendo los últimos cambios realizados



¿Qué es GitHub?

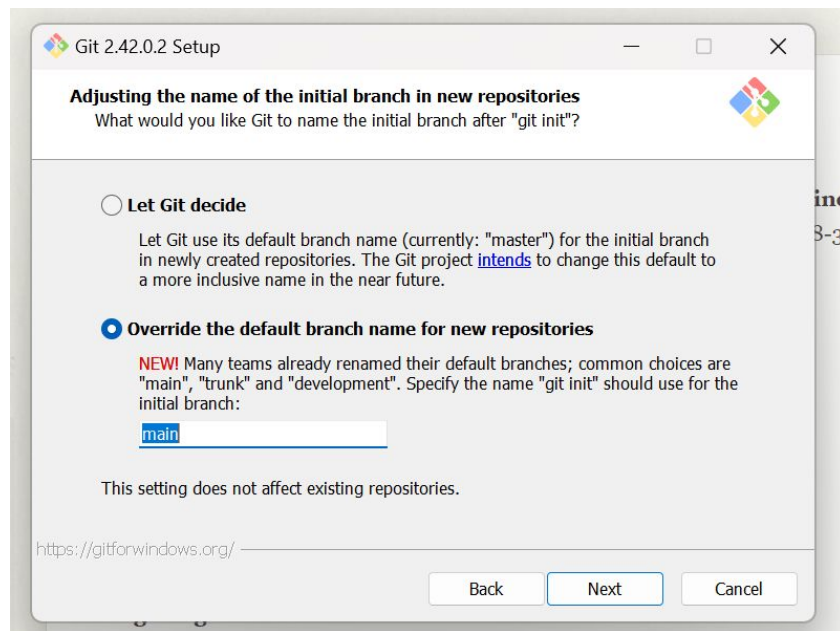
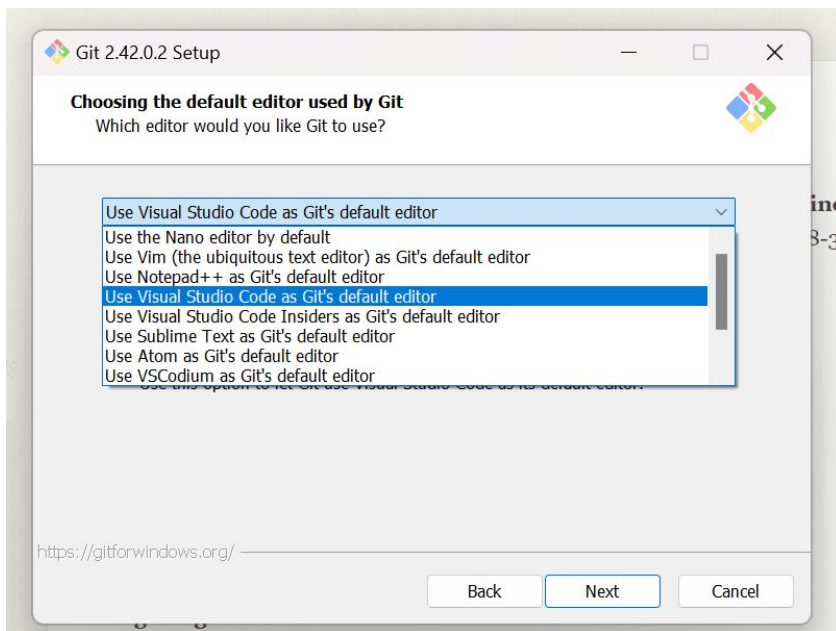
GitHub es una plataforma web que permite alojar proyectos controlados mediante Git. En GitHub tendremos una especie de “espejo” o duplicado de los proyectos que tenemos en nuestro propio ordenador y, además, podremos ver los proyectos de otra mucha gente.



Instalación de GIT

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

Instalación de GIT





Configuración inicial de GIT

La primera tarea que tenemos que realizar con GIT es configurar nuestro nombre de usuario y correo electrónico con el que vamos a trabajar. Para ello ejecutamos los comandos.

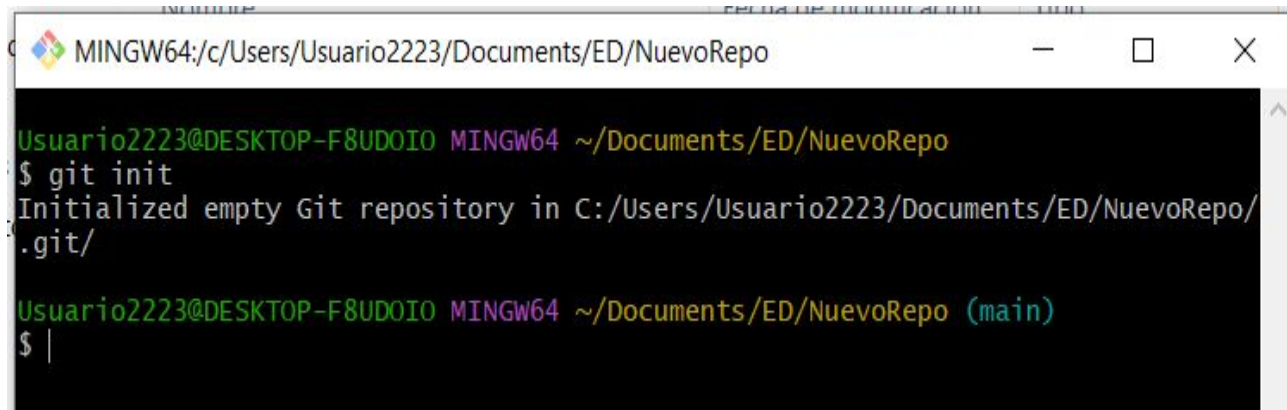
```
git config --global user.name "Tu Nombre"
```

```
git config --global user.email "Tu Email"
```

Crear repositorio GIT

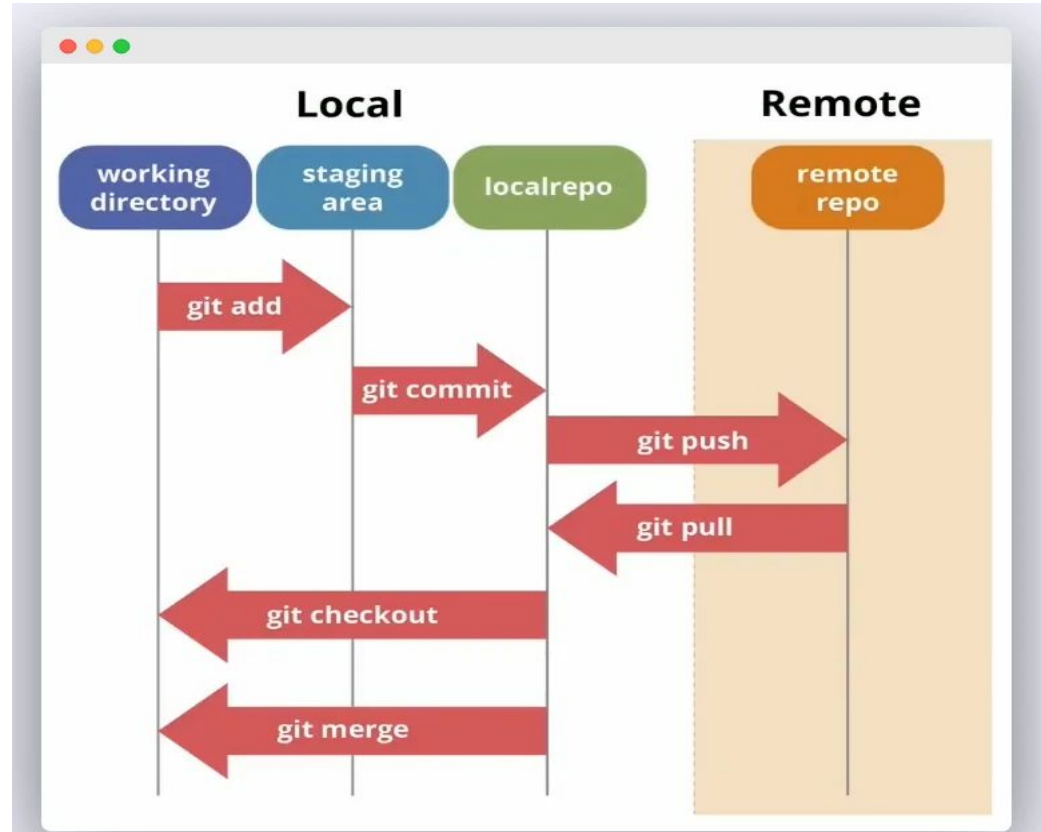
Para crear un repositorio de GIT tendré que navegar hasta la carpeta que quiero convertir en repositorio y ejecutar el comando:

- `git init`

A screenshot of a Windows command prompt window. The title bar shows the path 'MINGW64:/c:/Users/Usuario2223/Documents/ED/NuevoRepo'. The command prompt shows the user 'Usuario2223@DESKTOP-F8UD0IO' in a 'MINGW64' environment at the directory '~/Documents/ED/NuevoRepo'. The user has entered the command '\$ git init', and the output is 'Initialized empty Git repository in C:/Users/Usuario2223/Documents/ED/NuevoRepo/.git/'. The prompt now shows '(main)' and a cursor is ready for the next command.

```
Usuario2223@DESKTOP-F8UD0IO MINGW64 ~/Documents/ED/NuevoRepo
$ git init
Initialized empty Git repository in C:/Users/Usuario2223/Documents/ED/NuevoRepo/.git/
Usuario2223@DESKTOP-F8UD0IO MINGW64 ~/Documents/ED/NuevoRepo (main)
$ |
```

¿Cómo funciona GIT?





Añadiendo mi primer archivo al repositorio

Para añadir mi primer fichero al repositorio tenemos que ejecutar los siguientes comandos:

- **git add *nombreArchivo*** → Añade el fichero “nombre archivo” al staging area
- **git add .** → añade todos los archivos del working directory al staging



Añadiendo mi primer archivo al repositorio

Para pasar el fichero del staging area al repositorio realizamos el primer commit. Un commit es una versión de mi código a la que se le asocia un comentario descriptivo sobre los cambios realizados en dicha versión.

- `git commit -m "Comentario asociado al commit"`



Clonado de repositorios desde GitHub

Clonar un repositorio es hacer una copia idéntica de un proyecto que existe en GitHub y llevártela a tu máquina.

Para clonar un repositorio sólo nos hace falta saber su dirección exacta en GitHub. Vamos a clonar un repositorio de prueba que se encuentra en la ruta:

- <https://github.com/EnriquePicasso/ED2122.git>



Clonado de repositorios desde GitHub

Nos situamos dentro de la carpeta donde queremos clonar el repositorio:

- `cd Documentos/EntornosDesarrollo/Repos`

Clonamos el repositorio con el comando

- `git clone https://github.com/EnriquePicasso/ED2122.git`



Clonado de repositorios desde GitHub

Para actualizar los repositorios que hemos clonado anteriormente utilizamos el comando:

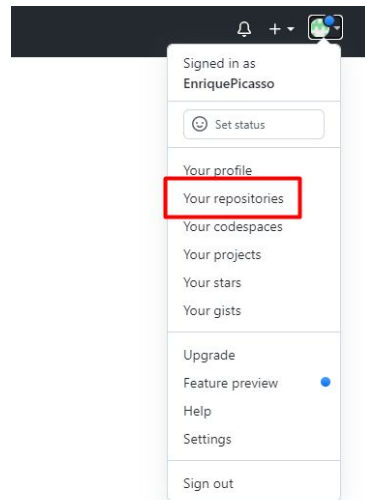
- **git pull**

Tenemos que situarnos en el directorio donde hemos clonado el proyecto previamente.



Creación de repositorios en GitHub

Ya sabemos clonar y actualizar repositorios. Veamos cómo podemos crear nuestros propios proyectos.



Creación de repositorios en GitHub

Create a new repository

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

Owner *

 EnriquePicasso ▾

Repository name *

/ NewRepo ✓

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

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:

[Skip this step](#) if you're importing an existing repository.



Add a README file

This is where you can write a long description for your project. [Learn more.](#)



Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)



Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

This will set  `main` as the default branch. Change the default name in your [settings](#).

Create repository



Creación de repositorios en GitHub

Como hemos marcado la opción de incluir el fichero README.md que contiene por defecto el nombre y descripción del repositorio, ya tenemos algo de contenido en el repositorio, y podemos clonarlo a nuestro equipo. En este paso, el repositorio todavía se encuentra en GitHub y tenemos que clonarlo a nuestra máquina.

Seguimos los pasos que hemos realizado previamente para clonar el repositorio de prueba.



Creación de repositorios en GitHub

Una vez hemos clonado nuestro repositorio de GitHub en nuestra máquina podemos realizar la sincronización entre nuestro repositorio local y el repositorio remoto de GitHub.

Para enviar los cambios realizados en el repositorio local al repositorio remoto utilizamos el comando:

- **git push**



Añadiendo mi primer archivo al repositorio

Para comprobar que mi fichero se encuentra en el estado correcto ejecuto el comando:

- `git status`

```
Usuario2122@DESKTOP-F8UD0IO MINGW64 ~/Documents/ED/GIT/Repo (main)
$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   HolaMundo.txt

Usuario2122@DESKTOP-F8UD0IO MINGW64 ~/Documents/ED/GIT/Repo (main)
$ |
```



Añadiendo mi primer archivo al repositorio

Para comprobar el historial de commits que se han realizado puedo ejecutar el comando

- `git log`
- `git log --oneline`
- `git log --graph`

```
Usuario2122@DESKTOP-F8UD0IO MINGW64 ~/Documents/ED/GIT/Repo (main)
$ git log
commit dee751d2cb327313192beb588a35d1fadb1f3154 (HEAD -> main)
Author: Enrique Martinez <enrique.martinez@iespablopicasso.es>
Date: Tue Nov 2 00:33:33 2021 +0100

Segundo commit

commit 93b98fc6af715e55cb2c2f57416fdc0886860fd1
Author: Enrique Martinez <enrique.martinez@iespablopicasso.es>
Date: Mon Nov 1 23:52:05 2021 +0100

Primer Commit

Usuario2122@DESKTOP-F8UD0IO MINGW64 ~/Documents/ED/GIT/Repo (main)
$ |
```



Revisar modificaciones

Para comprobar las modificaciones que se han realizado entre los dos últimos commits utilizamos:

- `git show`



Navegar entre versiones

Es posible navegar entre los diferentes commits que se han realizado. Para ello ejecutamos el comando:

- `git checkout codigoCommit`

Para volver a la última versión ejecutamos:

- `git checkout main`



Navegar entre versiones

En las nuevas versiones de GIT se está sustituyendo el comando checkout por switch.

- `git switch --detach codigoCommit`

Para volver a la última versión ejecutamos:

- `git switch main`



Deshacer modificaciones

Para deshacer una modificación de un fichero que se encuentra en mi working directory

- `git restore nombreFichero`

Si el fichero está en el staging area

- `git restore --staged nombreFichero`



Eliminar commits

Si queremos eliminar el último commit que hemos hecho tenemos dos opciones:

1. Si no he publicado mis cambios en mi repositorio remoto (GitHub) y quiero mantener los cambios en mi repositorio local
 - a. **git reset --soft HEAD~1** (Con el reset hacemos que la rama actual retroceda a la versión que queremos y con HEAD~1 le decimos que queremos volver a la versión inmediatamente anterior a donde estamos ahora, que es donde apunta HEAD).
2. Si no he publicado mis cambios en mi repositorio remoto (GitHub) y no quiero mantener los cambios en mi repositorio local
 - a. **git reset --hard HEAD~1** (Elimina todos los cambios que teníamos en el último commit. Mucho cuidado al ejecutar este comando).



Eliminar commits

3. Si ya he enviado el commit a mi repositorio remoto (he ejecutado el comando push)
 - a. En primer lugar tenemos que eliminar el commit en mi repositorio local como hemos visto anteriormente

git reset --hard HEAD~1

- b. A continuación, forzamos los cambios al repositorio remoto

git push origin -f



Eliminar commit específico

Si quiero eliminar un commit específico tengo que seguir los siguientes pasos:

1. Ejecuto git log para conocer el id del commit que quiero eliminar.
2. Ejecuto el comando
 - a. **git rebase -i iddelcommit anterior al que quiero eliminar**
3. Se nos abrirá el editor que tenemos por defecto en nuestro sistema y tenemos dos opciones
 - a. Eliminamos el commit que deseamos
 - b. Cambiamos la palabra ***pick*** por la palabra ***drop***



Modificar el último commit

Para modificar el commit más reciente utilizamos el comando

- **git commit --amend**

Te permite combinar los cambios preparados con el commit anterior en lugar de crear un commit nuevo.

También puede usarse para editar el anterior mensaje del commit sin cambiar el contenido del commit. Sin embargo, el comando no se limita a alterar el commit más reciente, sino que lo reemplaza por completo, por lo que el commit corregido será una entidad nueva con su propia referencia.



Comprobar las diferencias entre versiones

Para comprobar las diferencias entre el fichero que se encuentra en mi repositorio (última versión a la que hemos hecho commit) y la que tengo en mi directorio de trabajo (working directory) utilizamos el comando:

- `git diff`

```
Usuario2122@DESKTOP-F8UD0IO MINGW64 ~/Documents/ed/git/repo (main)
$ git diff
diff --git a/HolaMundo.txt b/HolaMundo.txt
index 3e9d430..4cc6b54 100644
--- a/HolaMundo.txt
+++ b/HolaMundo.txt
@@ -1,6 +1,6 @@
 Hola mundo
 Me llamo Enrique
 prueba
-Nueva línea
-Otra línea
+
+Modifico el fichero en mi working directory

Usuario2122@DESKTOP-F8UD0IO MINGW64 ~/Documents/ed/git/repo (main)
$ |
```



Comprobar las diferencias entre versiones

Vamos a utilizar Visual Studio Code para revisar las diferencias entre versiones de una forma más visual. Para ello, tenemos que configurar el fichero `.gitconfig` y añadirle la configuración deseada. Ejecuto:

- `git config --global -e`



Comprobar las diferencias entre versiones

```
[core]
  editor = \"C:\\Users\\Usuario2122\\AppData\\Local\\Programs\\Microsoft VS Code\\bin\\code.cmd\" --wait
[user]
  name = Enrique Martinez
  email = enrique.martinez@iespablocicasso.es
[credential]
  helper = cache --timeout=36000
[diff]
  tool = vscode
[difftool \"vscode\"]
  cmd = code --wait --diff $LOCAL $REMOTE
[merge]
  tool = vscode
[mergetool \"vscode\"]
  cmd = code --wait $MERGED
```



Comprobar las diferencias entre versiones

Otra opción es realizar la configuración directamente con comandos en la consola de GIT. Para eso escribiríamos:

- `git config --global diff.tool vscode`
- `git config --global difftool.vscode.cmd "code --wait --diff $LOCAL $REMOTE"`



Comprobar las diferencias entre versiones

Una vez configurada la herramienta, ejecutamos el comando:

- **git difftool**

Comparamos la versión del fichero que tenemos en el working directory con el último commit realizado sobre el fichero. Si tengo más de un fichero en mi working directory, el comando difftool abre una nueva ventana de VSCode para cada fichero.

Para evitarlo, debo indicar el nombre del fichero que quiero revisar:

- **git difftool HolaMundo.txt**



Ficheros README.md y .gitignore

El objetivo del fichero **README.md** es proporcionar información sobre el repositorio.

Es un fichero escrito en formato **markdown**. Se trata de un lenguaje de marcas parecido a HTML pero mucho más sencillo.

En el siguiente enlace puedes encontrar ejemplos sobre este formato (<https://guides.github.com/features/mastering-markdown/>)

Repositorio de Entornos de Desarrollo

Este es el primer repositorio que hemos creado en GIT

Estamos aprendiendo a trabajar con GIT y para eso hemos creado este repositorio. 🧑🎓

Algunas de las ventajas de GIT son

- Me permite controlar las versiones de mi código
- Puedo trabajar en equipo con mis compañeros
- Aprendo de otros desarrolladores consultando sus repositorios

Flujo de trabajo de GIT

Para añadir un fichero a nuestro repositorio remoto de [GitHub](#) tenemos que seguir los siguientes pasos

```
git add fichero
```

```
git commit -m "Comentario sobre el commit"
```

```
git push
```



Ficheros README.md y .gitignore

Los ficheros que se especifican en .gitignore se ignoran por completo y no se incluyen en el seguimiento.

Resulta útil para no subir al repositorio de GitHub ficheros irrelevantes.

```
.gitignore X
C: > Users > Usuario2122 > Documents > ED > GIT > Repositorios > RepoE
1  #Ignoro todos los ficheros con extensión .bak
2  *.bak
3
4  #Ignoro el fichero IgnoraDocu
5  ignoradocu.txt
6
```

Ramas

Una **rama** es un flujo en el cual se van haciendo cambios en el código (modificando, borrando o añadiendo archivos). Por defecto, cuando se crea un repositorio, el flujo principal se llama rama master, main o rama principal.

Cuando se crea una rama adicional, ese flujo se bifurca, es decir, se puede optar por seguir por la rama principal o seguir por la nueva rama. Es posible saltar de una rama a otra y continuar avanzando por cualquiera de ellas.

Release Branches





Ramas

El primer comando que vamos a utilizar es

- **git branch**

Este comando nos devuelve todas las ramas que tenemos creadas actualmente en nuestro repositorio y nos indica la rama en la que nos encontramos actualmente.

Si quiero crear una nueva rama utilizaré el comando

- **git branch *nombreRama***



Ramas

Para moverme entre ramas utilizo el comando:

- **git checkout nombreRama**

Puedo renombrar una rama con el comando

- **git branch -m nombreAntiguo nombreNuevo**

O eliminar una rama con el comando

- **git branch -d nombreRama**



Ramas

Los cambios que se realicen en cada rama son independientes, y sólo afectarán a esa rama. Puedo añadir ficheros, eliminarlos o modificarlos y el estado de los ficheros en el resto de ramas no se verán afectados.

Para ver el estado general de mi repositorio y comprobar los commits realizados en cada una de las ramas ejecuto el comando:

- `git log --oneline --all --graph`

```
Usuario2122@DESKTOP-F8UD0IO MINGW64 ~/Documents/ED/GIT/reporamas (main)
$ git log --oneline --all --graph
* 903ec51 (dev3) Modifico el fichero Fich1Main
| * 1b3dc45 (dev) Añado Fich2Dev
|/
* 81649b4 (HEAD -> main) Primer commit
```



Ramas. Merge

No obstante, una vez que he terminado de realizar los cambios en las ramas secundarias, esas modificaciones tengo que añadirlas a la rama main, donde tengo la versión “limpia” de mi código. Para ello tengo que realizar una operación de fusión de ramas o **merge**.

Para ello, nos vamos a la rama de destino (git checkout main) y utilizo el siguiente comando:

- **git merge *ramaOrigen***

Se realiza la fusión utilizando una estrategia de **Fast-forward**, que se trata de mover el HEAD que estaba apuntando a la rama de master y lo apuntamos al commit superior de la rama de origen



Ramas. Merge

Otra posible fusión sería realizar el merge de un fichero que tiene modificaciones en la rama main y en otra rama secundaria. No podemos realizar la fusión mediante la técnica de Fast-forward ya que tengo que fusionar cambios de dos ramas. En este caso se utiliza la técnica recursiva en la que GIT realiza un nuevo commit donde intenta añadir automáticamente los cambios de la rama origen en la rama destino.

```
Usuario2122@DESKTOP-F8UD0IO MINGW64 ~/Documents/ED/GIT/reporamas (main)
$ git log --oneline --all --graph
* 903ec51 (dev3) Modifico el fichero Fich1Main
| * 1b3dc45 (HEAD -> main, dev) Añado Fich2Dev
|/
* 81649b4 Primer commit
```



Ramas. Merge

```
Usuario2122@DESKTOP-F8UD0IO MINGW64 ~/Documents/ED/GIT/reporamas (main)
$ git log --oneline --all --graph
* 00320da (HEAD -> main) Merge branch 'dev3'
| \
| * 903ec51 (dev3) Modifico el fichero Fich1Main
| * | 1b3dc45 (dev) Añado Fich2Dev
| /
* 81649b4 Primer commit
```



Ramas. Conflicto

Los conflictos son cambios que se han realizado en los ficheros que GIT no puede resolver de forma automática y nos pide a nosotros que revisemos el fichero y nos quedemos con la versión correcta. Los conflictos más comunes se producen cuando he modificado la misma línea del mismo fichero en dos ramas independientes.

Cuando GIT quiere hacer el merge, no sabe qué versión es la correcta y se produce el conflicto.

```
Usuario2122@DESKTOP-F8UDOIO MINGW64 ~/Documents/ED/GIT/reporamas (main)
$ git merge dev4
Auto-merging Fich1Main.txt
CONFLICT (content): Merge conflict in Fich1Main.txt
Automatic merge failed; fix conflicts and then commit the result.
```



Ramas. Conflicto

Para resolver el conflicto abrimos el fichero y se nos muestra de la siguiente forma

```
sers > Usuario2122 > Documents > ED > GIT > RepoRamas > ≡ Fich1Main.txt
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
Realizo un cambio en la primera línea en la rama main
=====
Ahora modifico la línea en la rama dev4
>>>>>>> dev4 (Incoming Change)

Modifico el fichero en la rama dev3

Modifico el fichero en la rama dev4
```

Ramas. Conflicto

Para resolver el conflicto, eliminamos la parte que no nos interesa para quedarnos con una de las dos opciones. O podemos modificar el fichero añadiendo una nueva versión que mezcle ambas.

```
C: > Users > Usuario2122 > Documents > ED > GIT > RepoRamas > Fich1Main.txt
```

```
1 He modificado la línea pero no me importa en que rama lo hice|
2
3 Modifico el fichero en la rama dev3
4
5 Modifico el fichero en la rama dev4
```

Por último, realizamos un add y commit del fichero y mi conflicto queda resuelto.



Tags. Etiquetas

A medida que se avanza en un proyecto y se van completando hitos o se añaden nuevas funcionalidades, es recomendable asignar etiquetas con un nombre y/o un número de versión. Nos sirven para identificar commits particulares.

Para crear un tag en el commit que nos encontramos actualmente utilizamos el comando:

- `git tag nombreTag`



Tags. Etiquetas

La opción que hemos usado previamente se denomina etiqueta ligera. Existe la posibilidad de crear una etiqueta con más información, incluso añadirle a la etiqueta un mensaje tal y como hacemos con los commits. Para eso, utilizamos el comando:

- `git tag -a nombreTag -m "Mensaje asociado a la etiqueta"`

Para comprobar todas las etiquetas que tenemos creadas hasta el momento utilizamos el comando

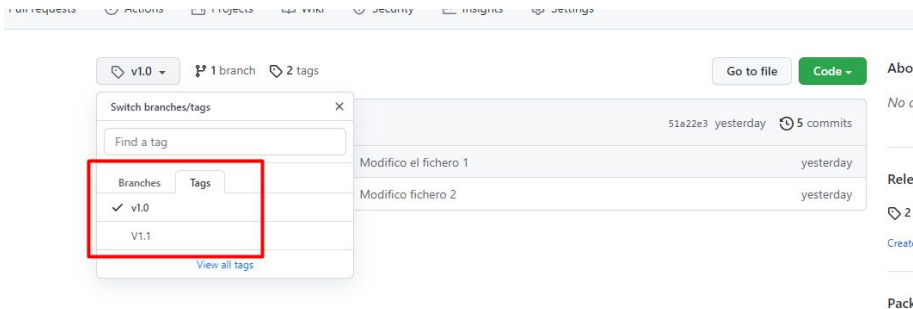
- `git tag -n`

Tags. Etiquetas

Por defecto, el comando git push no actualiza las etiquetas en el repositorio remoto, es necesario indicarlo de forma explícita con la opción **--tags**

- **git push --tags**

Una vez ejecutado el comando, nos aparecen las etiquetas que hemos creado en la parte superior izquierda de nuestro repositorio.





Tags. Etiquetas

Para eliminar un tag ejecuto el comando:

- `git tag -d nombreTag`

```
Usuario2122@DESKTOP-F8UDOIO MINGW64 ~/Documents/ED/GIT/RepoPrueba (main)
$ git tag -d v1.1
Deleted tag 'v1.1' (was 5d28bab)

Usuario2122@DESKTOP-F8UDOIO MINGW64 ~/Documents/ED/GIT/RepoPrueba (main)
$ git log --oneline
51a22e3 (HEAD -> main, tag: v1.0, origin/main) Añado fichero
7e0d6c8 Modifico el fichero 1
b581980 Modifico fichero 2
0ac9384 Añado fichero 2
20ed9d3 Añado fichero 1
```



Desarrollo colaborativo. Fork y Pull request

Una de las características más importantes, si no la que más, que hace atractivo a GitHub es la posibilidad de desarrollo colaborativo.

Imaginad que queremos colaborar en un repositorio remoto que hemos encontrado en GitHub. El primer paso que tenemos que dar es hacer un ***fork*** de ese repositorio en nuestra cuenta de GitHub.

Un ***fork*** consiste en crear una copia de un repositorio remoto de GitHub.

La diferencia entre clonar un repositorio y realizar un fork es que con el fork se crea una copia independiente del repositorio original, con una URL diferente. De esta forma, las modificaciones que se realicen en esa copia no afectarán al repositorio original.

Desarrollo colaborativo. Fork y Pull request

Una vez que realices las modificaciones pertinentes en el proyecto debes crear un Pull request para integrar tus cambios al repositorio original al que hemos hecho el fork.

The screenshot displays a GitHub repository page for a user named 'enriquemfinf'. At the top, it shows the 'main' branch with 3 branches and 2 tags. A status bar indicates 'This branch is 1 commit ahead of EnriquePicasso:main.' and includes a 'Contribute' dropdown menu, which is highlighted with a red box. Below this, a list of files is shown, including 'Directorio', 'Fich1.txt', 'Fich2.txt', 'Fich3.txt', and 'README.md'. A modal dialog is open, displaying the message 'This branch is 1 commit ahead of EnriquePicasso:main. Open a pull request to contribute your changes upstream.' and a prominent green 'Open pull request' button, also highlighted with a red box. The right sidebar contains sections for 'About', 'Releases', and 'Packages'.

Desarrollo colaborativo. Fork y Pull request

Al propietario del repositorio original le aparece toda la información referente a las modificaciones que queremos incorporar al repositorio (Archivos modificados, commits realizados...).

Para aceptar esos cambios pinchamos en “*Merge pull request*”

Update Fich3.txt #2



enriquemfinf wants to merge 1 commit into EnriquePicasso:main from enriquemfinf:main



Conversation 0



Commits 1



Checks 0



Files changed 1



enriquemfinf commented 7 minutes ago

First-time GitHub contributor



Esta es mi contribución al repositorio



Update Fich3.txt

Verified

8d54745

Add more commits by pushing to the `main` branch on `enriquemfinf/RepoPrueba`.



Continuous integration has not been set up

[GitHub Actions](#) and [several other apps](#) can be used to automatically catch bugs and enforce style.



This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request

You can also [open this in GitHub Desktop](#) or [view command line instructions](#).

Eclipse con GIT. EGIT

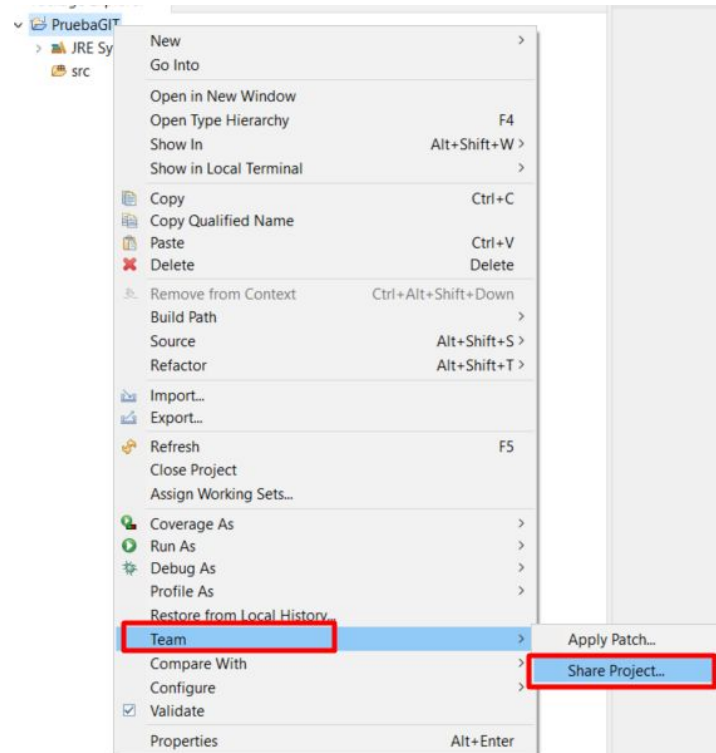
Tenemos la posibilidad de utilizar GIT directamente con Eclipse. De forma que nos resulte más fácil realizar el control de versiones sobre mi proyecto, ya que no sería necesario ejecutar los comandos de consola que hemos visto hasta ahora.



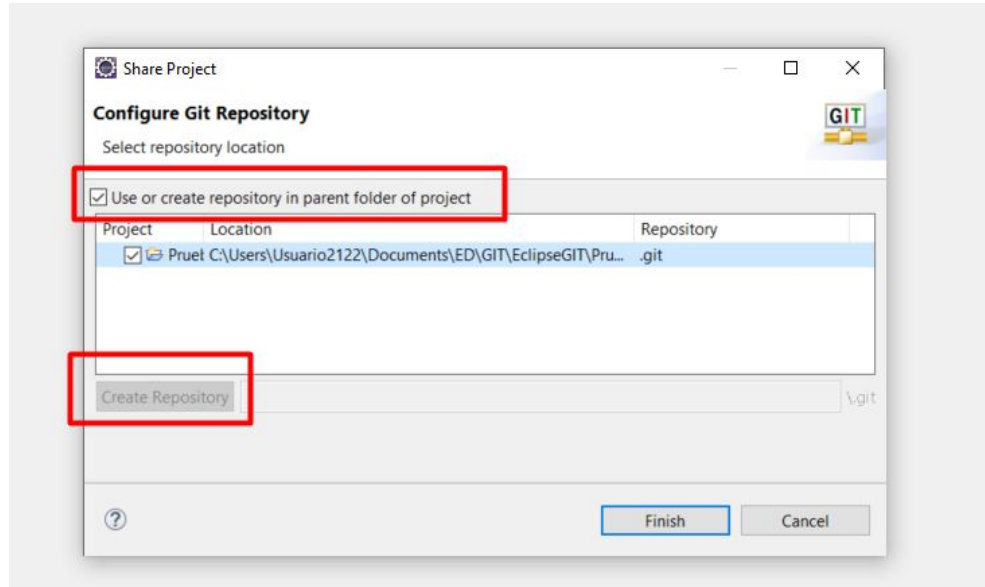
Eclipse con GIT. EGIT

Para crear nuestro primer repositorio, utilizando la carpeta del proyecto que acabamos de crear pulsamos sobre el proyecto con el botón secundario del ratón y seleccionamos **Team/Share Project**.

En el caso de que nos aparezca la opción de Teams, tendremos que actualizar Eclipse o buscar en el Marketplace de Eclipse el plugging para trabajar con GIT.

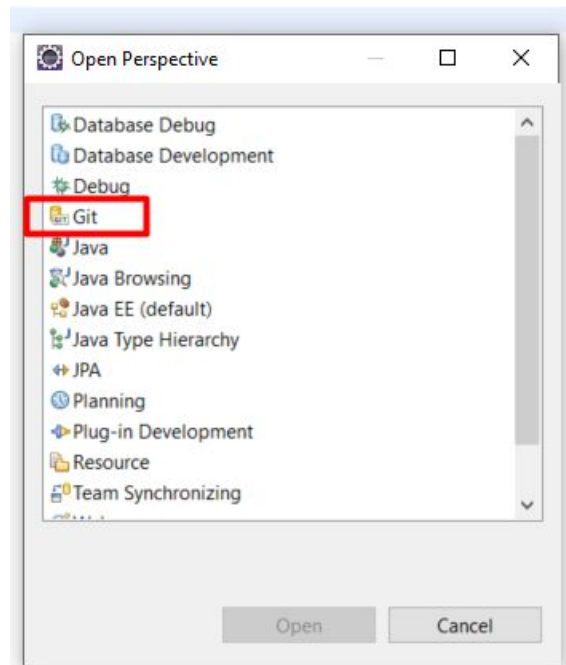
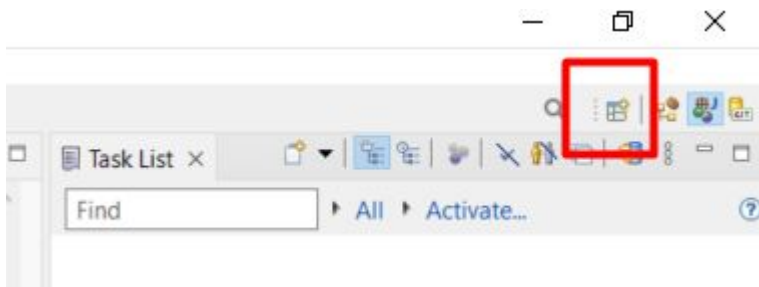


Eclipse con GIT. EGIT

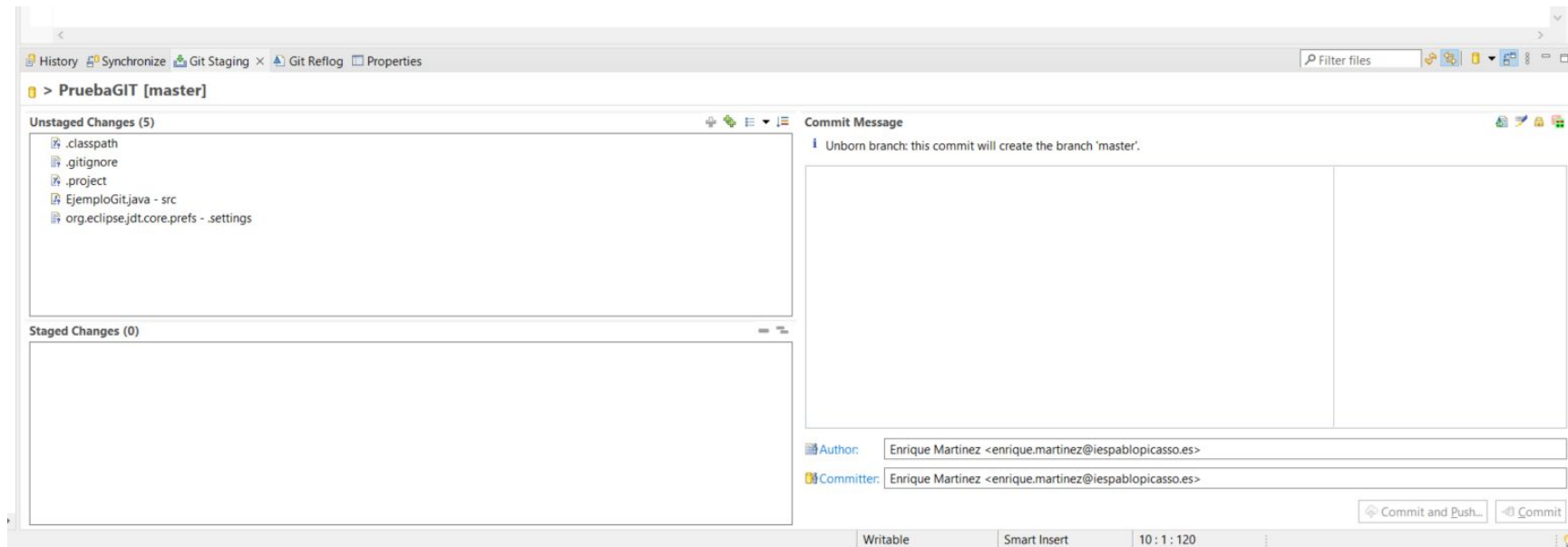


Eclipse con GIT. EGIT

Para trabajar con GIT tenemos que activar la perspectiva correspondiente en Eclipse. Para ello vamos a la parte superior derecha de nuestro IDE y buscamos la perspectiva correcta.

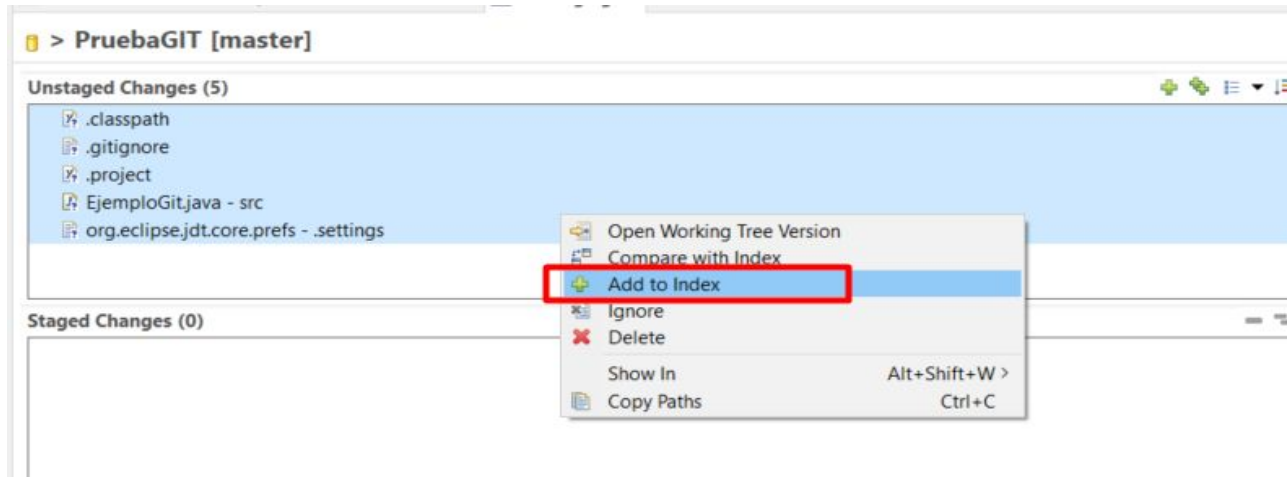


Eclipse con GIT. EGIT



EGIT. Primer commit

Para realizar el primer commit lo primero que tendré que realizar es pasar los ficheros al área de Staging. Para eso, selecciono los ficheros que quiero incluir en mi commit y pulsamos sobre **“Add to index”**



EGIT. Primer commit

Mis ficheros ya se encuentran en área de Staged, con lo que nos quedaría realizar el commit añadiéndole un comentario al mismo.



EGIT. Historial

History × Synchronize Git Staging Git Reflog Properties

Repository: PruebaGIT

Id	Message	Author	Authored Date	Committer	Committed Date
4699db4	master HEAD Añado saludo	Enrique Martinez	79 seconds ago	Enrique Martinez	79 seconds ago
6c7861e	o Agregamos variable nombre	Enrique Martinez	3 minutes ago	Enrique Martinez	3 minutes ago
69cf847	o Añado mis primeros ficheros	Enrique Martinez	4 minutes ago	Enrique Martinez	4 minutes ago

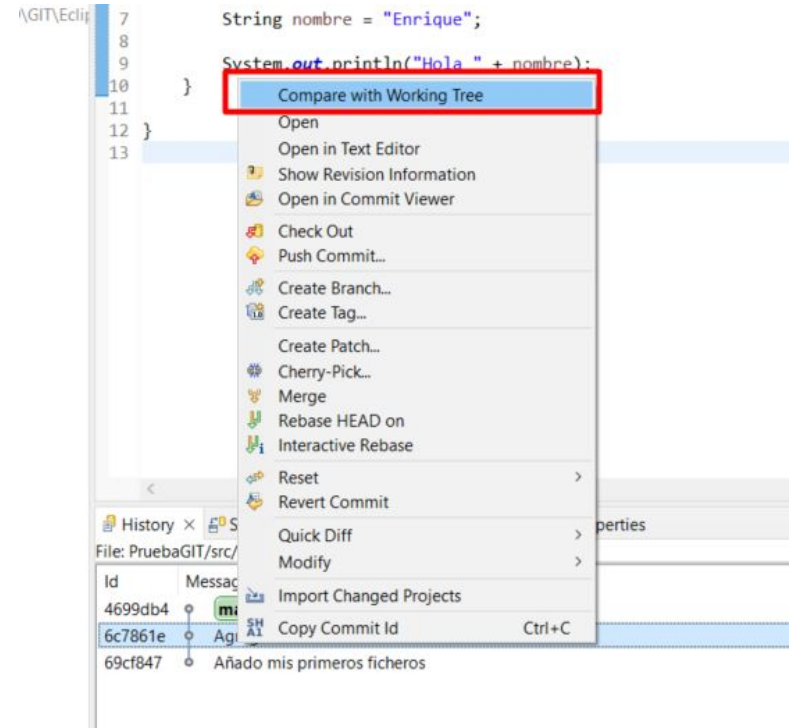
commit 4699db466aa13afdf0d983fbccfeb800c5cd536c
Author: Enrique Martinez <enrique.martinez@iespablocicasso.es> 2021-11-22 18:45:43
Committer: Enrique Martinez <enrique.martinez@iespablocicasso.es> 2021-11-22 18:45:43
Parent: [6c7861e2da838117ff429de8b6f3ff9516b6fbf1](#) (Agregamos variable nombre)
Branches: [master](#)

Añado saludo

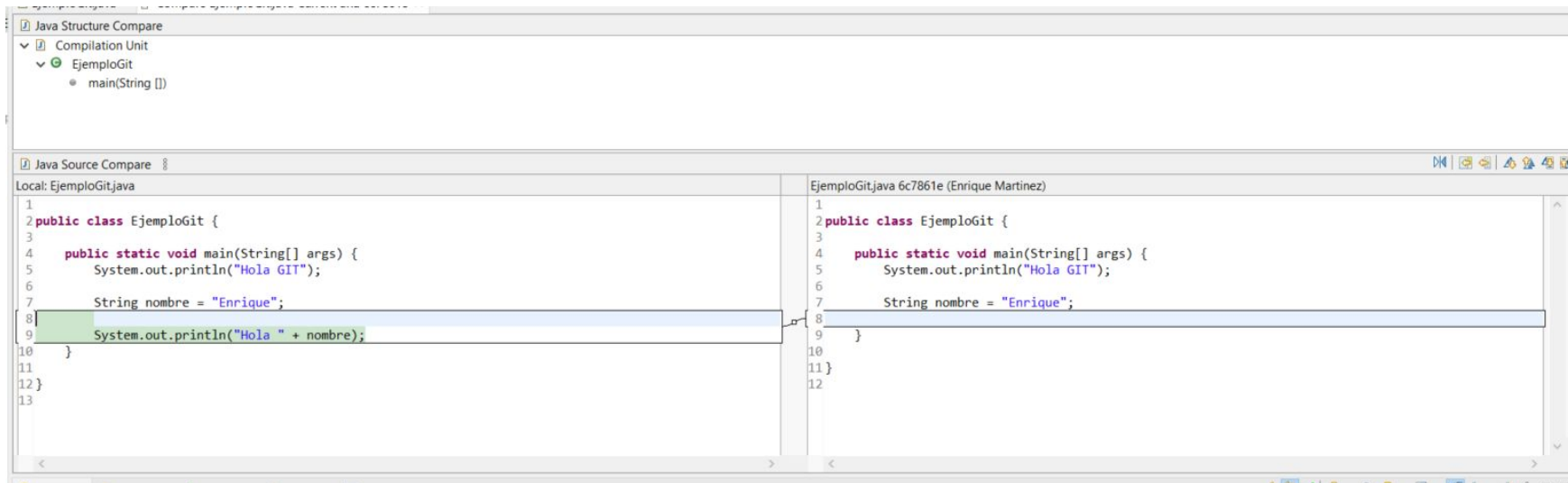
src/EjemploGit.java

EGIT. Comparar versiones

Si quiero ver el estado de mi fichero en uno de los commits anteriores pulso sobre el commit con el botón secundario del ratón y selecciono **“Compare with working tree”**



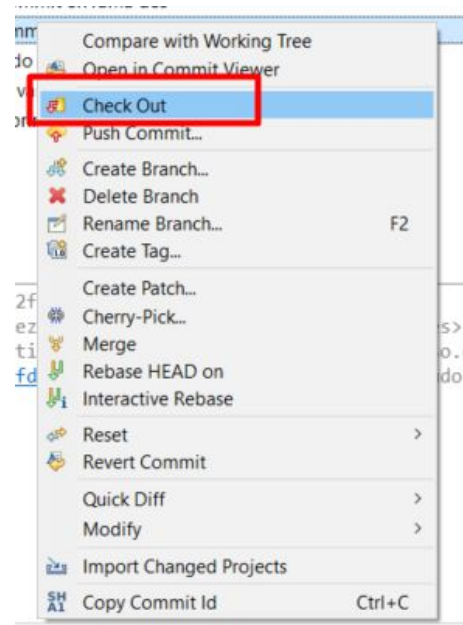
EGIT. Comparar versiones



EGIT. Checkout

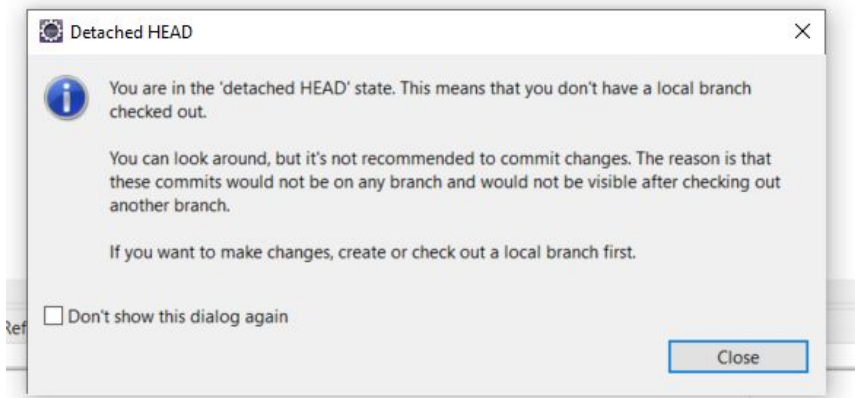
Si quiero volver a una versión anterior de mi código tendré que realizar un checkout. Botón secundario sobre el checkout al que quiero volver y pulso **Checkout**.

Para volver a mi último commit, donde se encuentra el HEAD, realizo el checkout sobre ese último commit.



EGIT. Ramas.

Al realizar el checkout me muestra este mensaje.
¿Qué nos indica?

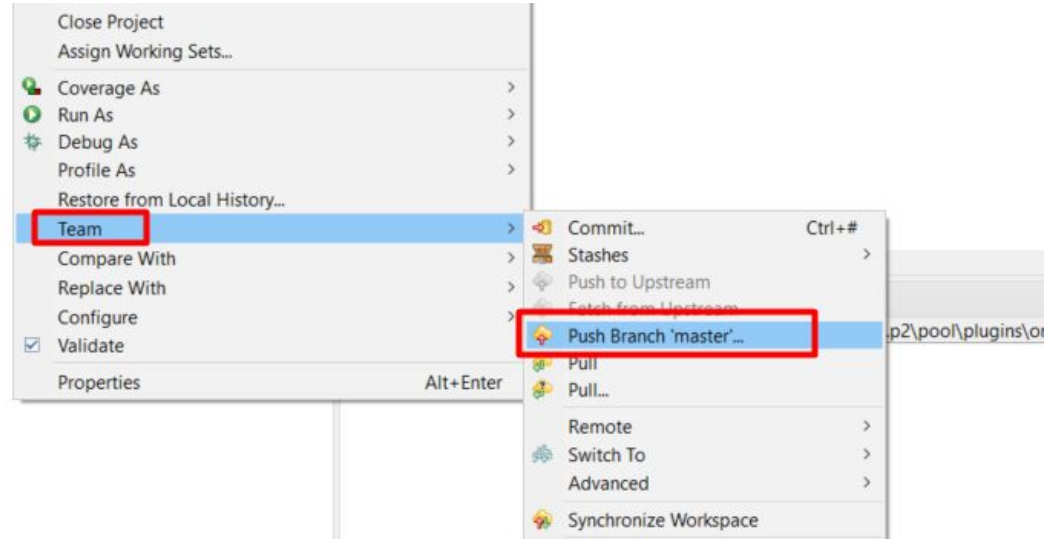



Actividad

- Investiga cómo crear una nueva rama con eGIT
- Crea varios commits en esa rama
- Realiza un merge con la rama main

EGIT. GitHub

Tenemos la posibilidad de subir mi repositorio local a un repositorio remoto de GitHub. Para ello nos vamos al menú **Team** del proyecto y seleccionamos la opción “**Push Branch ‘master’**”



 Push Branch master

Destination Git Repository

Enter the location of the destination repository.

Remote name:

origin

Location

URI:

https://github.com/EnriquePicasso/eGIT.git

Local Folder...

Host:

github.com

Repository path:

/EnriquePicasso/eGIT.git

Connection

Protocol:

https

Port:

Authentication

User:

enrique.martinez@iespablopicasso.es

Password:

••••••••••

☐ Store in Secure Store


?

< Back

Preview >


Push

Cancel



 Push Branch master

Push to branch in remote

Select a remote and the name the branch should have in the remote.



Source:

 master  c776529 Merge branch 'Rama_des'

Destination:

Remote:


Branch:

☒ Configure upstream for push and pull

When pulling:

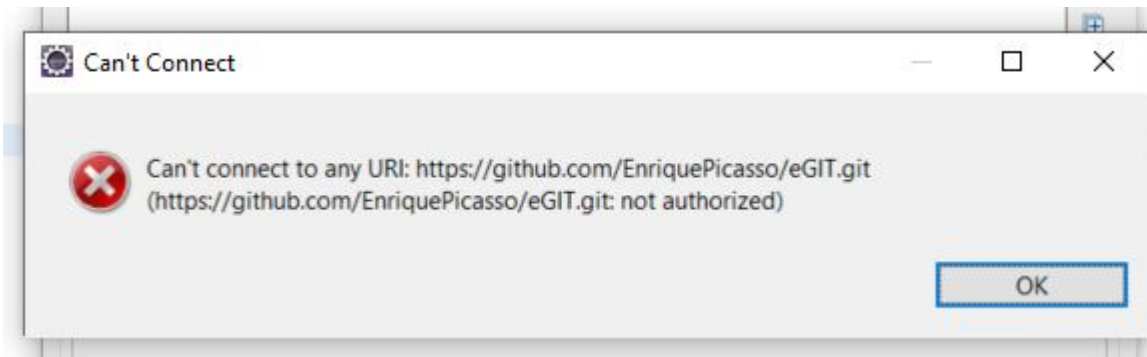
☒ Force overwrite branch in remote if it exists and has diverged

Show [advanced push](#) dialog



EGIT. GitHub

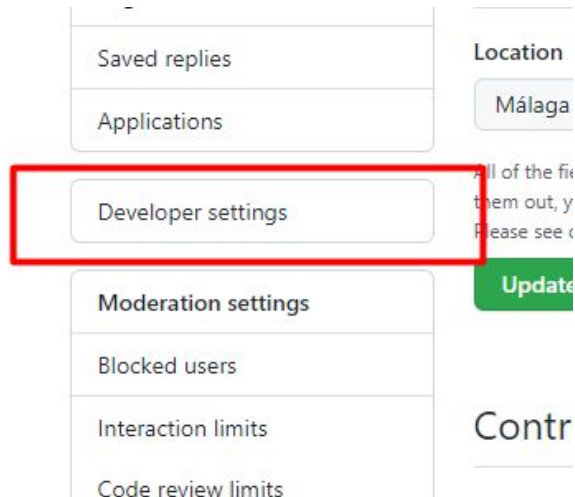
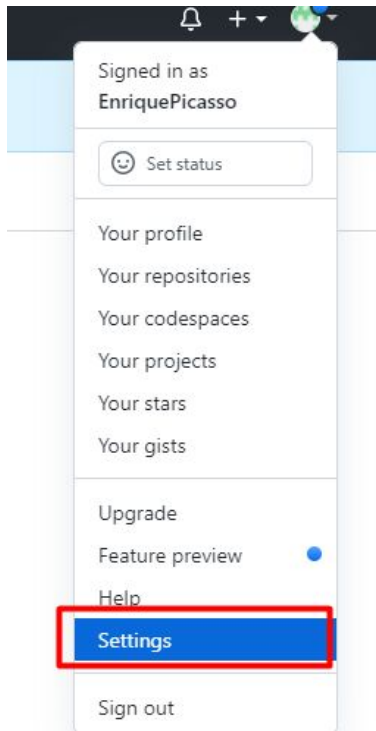
Al intentar conectar con GitHub utilizando nuestro usuario y contraseña nos devuelve este error



EGIT. GitHub

Para solucionarlo tenemos que configurar un token de acceso.

Los tokens de acceso personal (PAT) son una alternativa al uso de contraseñas para la autenticación en GitHub cuando utilizas la API de GitHub





EGIT. GitHub

GitHub Apps

OAuth Apps

Personal access tokens

Personal access tokens

Generate new token

Need an API token for scripts or testing? [Generate a personal access token](#) for quick access to the [GitHub API](#).

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

New personal access token

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

Eclipse

What's this token for?

Expiration *

90 days



The token will expire on Sun, Feb 20 2022

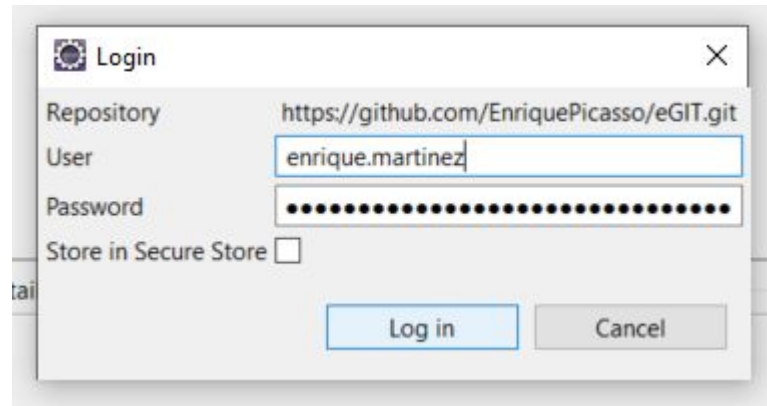
Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

- | | |
|---|--------------------------------------|
| <input checked="" type="checkbox"/> repo | Full control of private repositories |
| <input checked="" type="checkbox"/> repo:status | Access commit status |
| <input checked="" type="checkbox"/> repo_deployment | Access deployment status |
| <input checked="" type="checkbox"/> public_repo | Access public repositories |
| <input checked="" type="checkbox"/> repo:invite | Access repository invitations |
| <input checked="" type="checkbox"/> security_events | Read and write security events |

EGIT. GitHub

Una vez generado el token, lo utilizamos en nuestra ventana de login como la contraseña del usuario con el que queremos conectar a GitHub.





Push Branch master



Push Confirmation

Confirm following expected push result.



master → master [new branch]



Message Details

Repository <https://github.com/EnriquePicasso/eGIT.git>

☐ Cancel push if result would be different than above because of changes on remote

☐ Show dialog with result only when it is different from the confirmed result above




< Back


Preview >


Push

Cancel

EGIT. GitHub

 master ▾


 1 branch






 0 tags

Go to file

Add file ▾

Code ▾

 EnriquePicasso Merge branch 'Rama_des' ... c776529 3 hours ago ⌚ 6 commits

 .settings	Añado mis primeros ficheros	3 hours ago
 src	Merge branch 'Rama_des'	3 hours ago
 .classpath	Añado mis primeros ficheros	3 hours ago
 .gitignore	Añado mis primeros ficheros	3 hours ago
 .project	Añado mis primeros ficheros	3 hours ago

Help people interested in this repository understand your project by adding a README.

Add a README



EGIT. GitHub

Actividad

- Añade un fichero Readme desde GitHub y actualiza tu repositorio local desde Eclipse para descargar ese nuevo fichero que has añadido