

# Control de versiones

Para mí después de llevar un año trabajando con sistemas de control de versiones resulta absolutamente Imprescindible en cualquier desarrollo incluir este tipo de software. El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante. En mi caso uso GitHub, aunque en el mercado existen otras alternativas similares como Mercurial, Subversion, CVS, Perforce, Bazaar...

Parece imprescindible introducir algunos conceptos previos que iré empleando a lo largo de este apartado.

Además de proveer seguridad ante una posible pérdida de datos o vuelta atrás a una versión anterior de código (sin las copias y generación de .zip de distintos estados que hemos hecho todos...), posibilita el trabajo en equipo, al permitir que varios desarrolladores trabajen sobre el mismo código. En mi empresa se usa un sistema de control de versiones llamado "Rational", anteriormente se usaba SVN, por lo que GIT es relativamente nuevo para mí, lo uso para mis proyectos personales.

Hay mucha terminología al respecto, y huelga decir que este tipo de software tiene muchas más posibilidades de las que por las restricciones propias de este trabajo no es posible recoger, limitándome a realizar una introducción y uso básico.

## Terminología

---

### Repositorio

El **repositorio** es el lugar en el que se almacenan los datos actualizados e históricos de cambios

### Revisión ("version")

Una **revisión** es una versión determinada de la información que se gestiona.

### Línea base ("Baseline")

Una revisión aprobada de un documento o fichero fuente, a partir del cual se pueden realizar cambios subsiguientes.

### Rama ("branch")

Un módulo puede ser **branched** o **bifurcado** en un instante de tiempo de forma que, desde ese momento en adelante se tienen dos copias (ramas) que evolucionan de forma independiente siguiendo su propia línea de desarrollo.

### Integración o fusión ("merge")

Una **integración** o **fusión** une dos conjuntos de cambios sobre un fichero o un conjunto de ficheros en una revisión unificada de dicho fichero o ficheros.

### Desplegar ("Check-out", "checkout", "co")

Un despliegue crea una copia de trabajo local desde el repositorio.

### "Publicar" o "Enviar" ("commit", "check-in", "ci", "install", "submit")

Un **commit** sucede cuando una copia de los cambios hechos a una copia local es escrita o integrada sobre el repositorio.

## Conflicto

Un conflicto ocurre en las siguientes circunstancias:

1. Los usuarios X e Y despliegan versiones del *archivo del mismo archivo X*, de manera que *ambos modifican una o varias líneas de manera simultánea*.
2. El usuario X envía cambios (entre ellos las líneas comunes).
3. El usuario Y no actualiza el *archivo A* tras el envío del usuario X.
4. El usuario Y realiza cambios en el mismo archivo (afectando a las mismas línea que ya había modificado el usuario X).
5. El usuario Y intenta posteriormente enviar esos cambios al *archivo A*. Se produce un conflicto, cómo veremos más adelante en un ejemplo gráfico.

El sistema es incapaz de fusionar los cambios. El usuario Y debe *resolver* el conflicto combinando los cambios, o eligiendo uno de ellos para descartar el otro.

## Actualización ("sync" ó "update")

Una **actualización** integra los cambios que han sido hechos en el repositorio (por ejemplo por otras personas) en la **copia de trabajo** local.

Git tiene la particularidad de ser distribuido, que no tienen otros sistemas de este tipo. Esto significa que hay una copia del repositorio en local. Hay que tener algo de cuidado en ir haciendo commits con cierta periodicidad ya que las actualizaciones de código de hacen contra nuestro repositorio local y no contra el código del proyecto.

## ¿Cómo funciona un sistema de control de versiones?

---

Lo habitual es que cada programador realice los cambios necesarios en el código fuente para la tarea que se le ha encomendado. Una vez que dichos cambios están listos, los envía al servidor, de manera que el resto pueda recibirlos en cualquier momento, y así trabajar sobre dichos cambios cuando tengan que realizar cualquier otra tarea. Se puede dar el caso de que varios programadores trabajen sobre el mismo fichero o ficheros, en cuyo caso el sistema lo detectará, y actuará para evitar posibles conflictos:

- Los programadores han trabajado en porciones de código diferentes: En principio, no se han pisado las líneas en las que han trabajado, así que es probable que sea suficiente efectuar ambos cambios sobre el fichero, sin más. Casi todos los sistemas de control de versiones detectan esta situación y realiza la unión de los cambios de forma automática.
- Los programadores han trabajado en líneas de código comunes, modificando, eliminando o añadiendo líneas en la misma porción de código: En estos casos, el sistema suele señalar que hay un conflicto entre ambos cambios, y habitualmente genera un fichero intermedio convenientemente marcado para que se puedan revisar ambos cambios de forma simultánea, y así quedarse con uno, con el otro, o con una combinación de los dos, realizando la unión a mano y eliminando lo que sobra (merge).

## Ventajas

---

Puedes volver a cualquier punto del desarrollo para ver qué aspecto tenía un determinado fichero de código, o volver a una versión donde todo funcionaba antes de haber metido la pata.

Varios programadores pueden trabajar sobre el mismo proyecto, en distintas características de forma simultánea, guardando los cambios en cada una de ellas, y uniéndolos al desarrollo principal.

Puedes crear una nueva versión para probar un experimento, o corregir un bug.

En todo momento se sabe quien realizó un determinado cambio y cuándo.

## A modo de resumen, ¿Cómo funciona Git?

---

Cada vez que confirmas un cambio, o guardas el estado de tu proyecto en Git, él básicamente hace una foto del aspecto de todos tus archivos en ese momento, y guarda una referencia a esa instantánea. Para ser eficiente, si los archivos no se han modificado, Git no almacena el archivo de nuevo, sólo un enlace al archivo anterior idéntico que ya tiene almacenado.

Para verlo en funcionamiento vamos a crear un repositorio y almacenar un proyecto de ejemplo.

Haremos algunas pruebas para ver cómo se puede actualizar el código en el repositorio, volver a una versión anterior o colaborar entre varios desarrolladores.

Evidentemente todo a nivel básico por la extensión limitada de este apartado y del proyecto en general. Los sistemas de control de versiones tienen más potencia de la aquí vista y permiten muchas más opciones como los branch, merges, baselines...

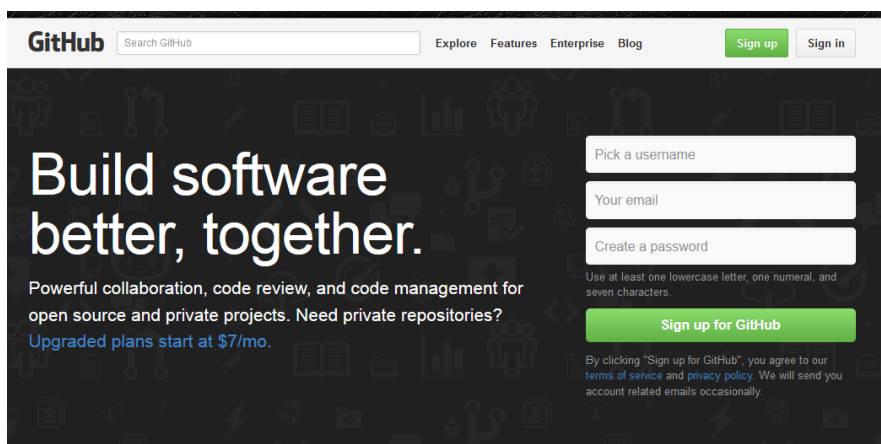
Git funciona con dos repositorios, uno local dónde ir haciendo los commits y el repositorio “general” dónde hacer los “pull”, de manera que si perdemos el repositorio local siempre se puede recupera de GitHub.

Vamos a ver un ejemplo algo más concreto para ver cómo funciona Git.

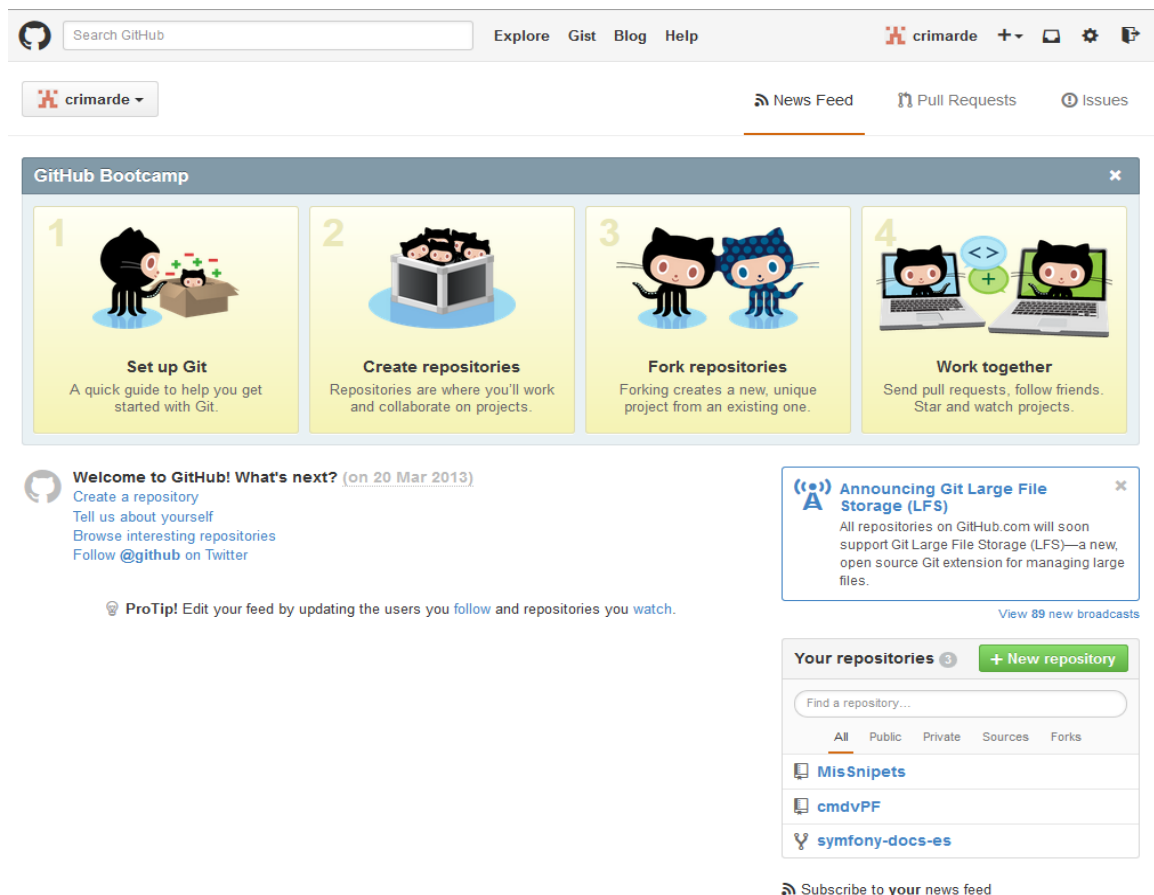
## Crear un repositorio

Vamos a mostrar los pasos para crear u utilizar un repositorio Git y utilizarlo como control de versiones.

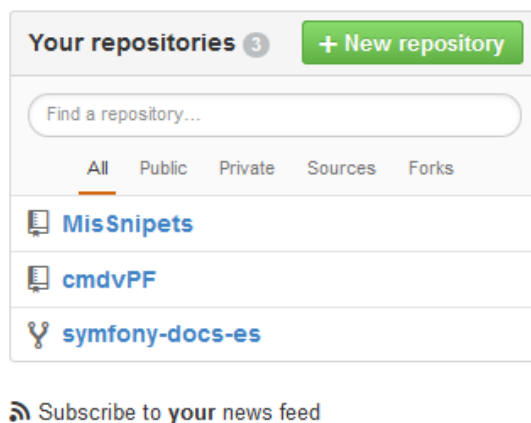
Accedemos a la web de gitHub <https://github.com/>





Accedemos a nuestra cuenta o creamos una nueva.



Una vez hemos accedido seleccionamos crear un nuevo repositorio.



**Owner**  **Repository name**

 **crimarde** /

Great repository names are short and memorable. Need inspiration? How about **turbo-meme**.

**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. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None** ⓘ

**Create repository**

GitHub nos informa que se ha creado el repositorio correctamente y nos ofrece la url para acceder a él. La copiamos porque más tarde la tendremos que insertar en eclipse para acceder al repositorio y sincronizar nuestro código.

**Quick setup — if you've done this kind of thing before**

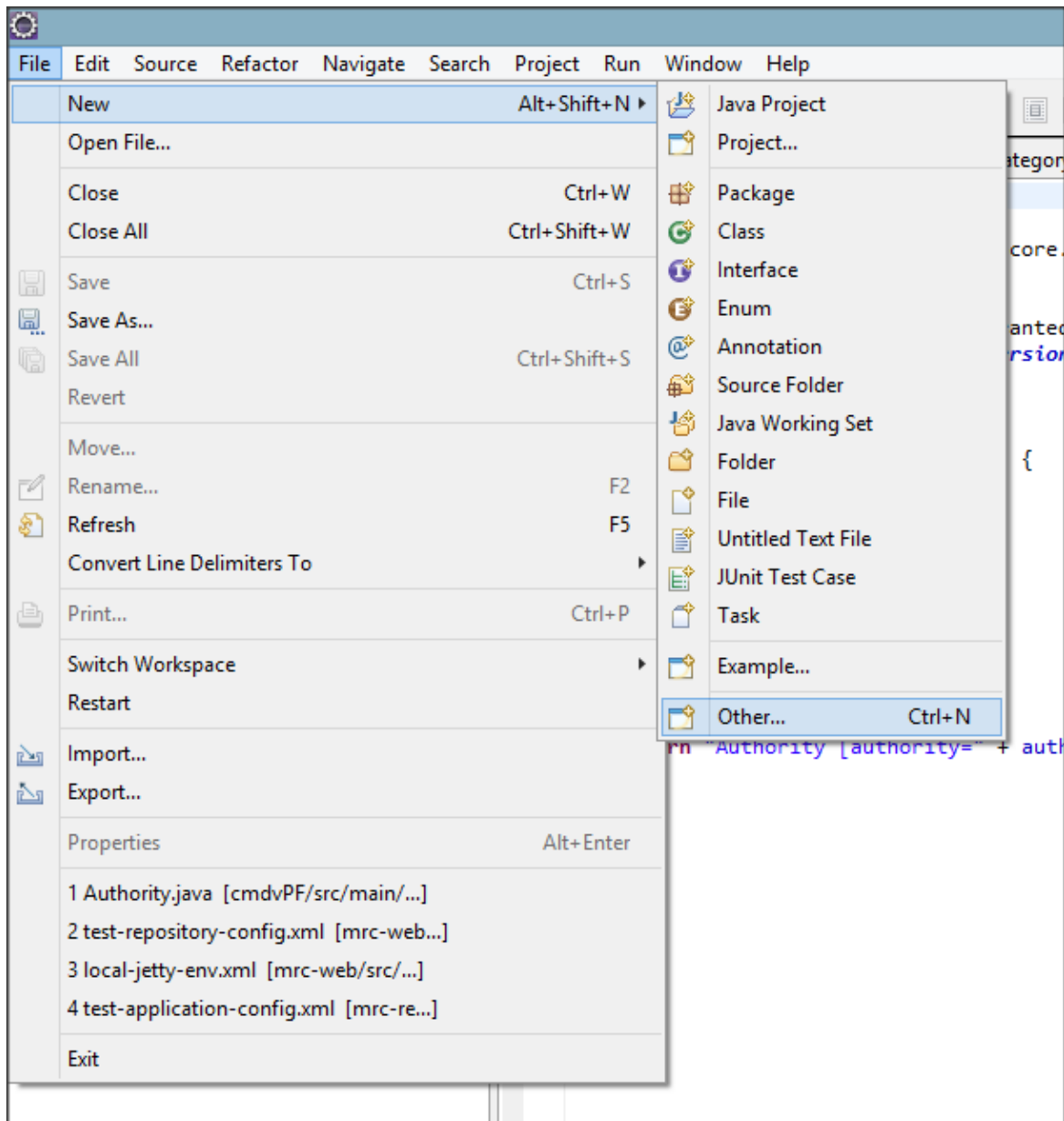
 **Set up in Desktop** or **HTTPS** **SSH**  

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

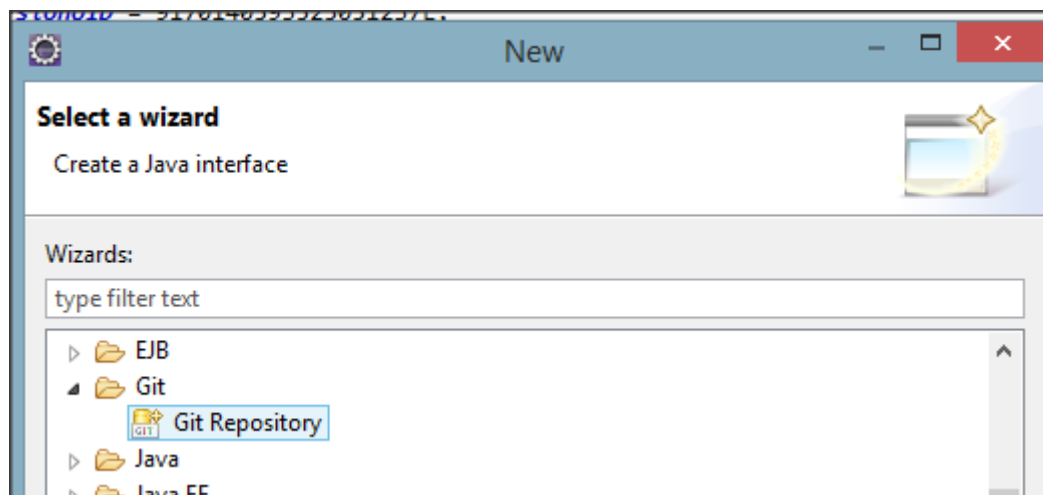
<https://github.com/crimarde/GitHub.git>

## Creación del repositorio en Eclipse

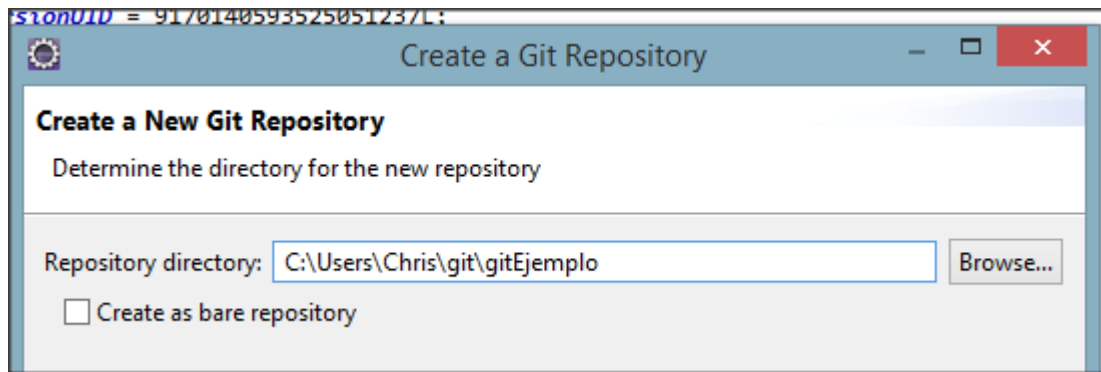
Creamos un nuevo repositorio de Git para ello seleccionamos File > New > Other > Git > Git Repository.



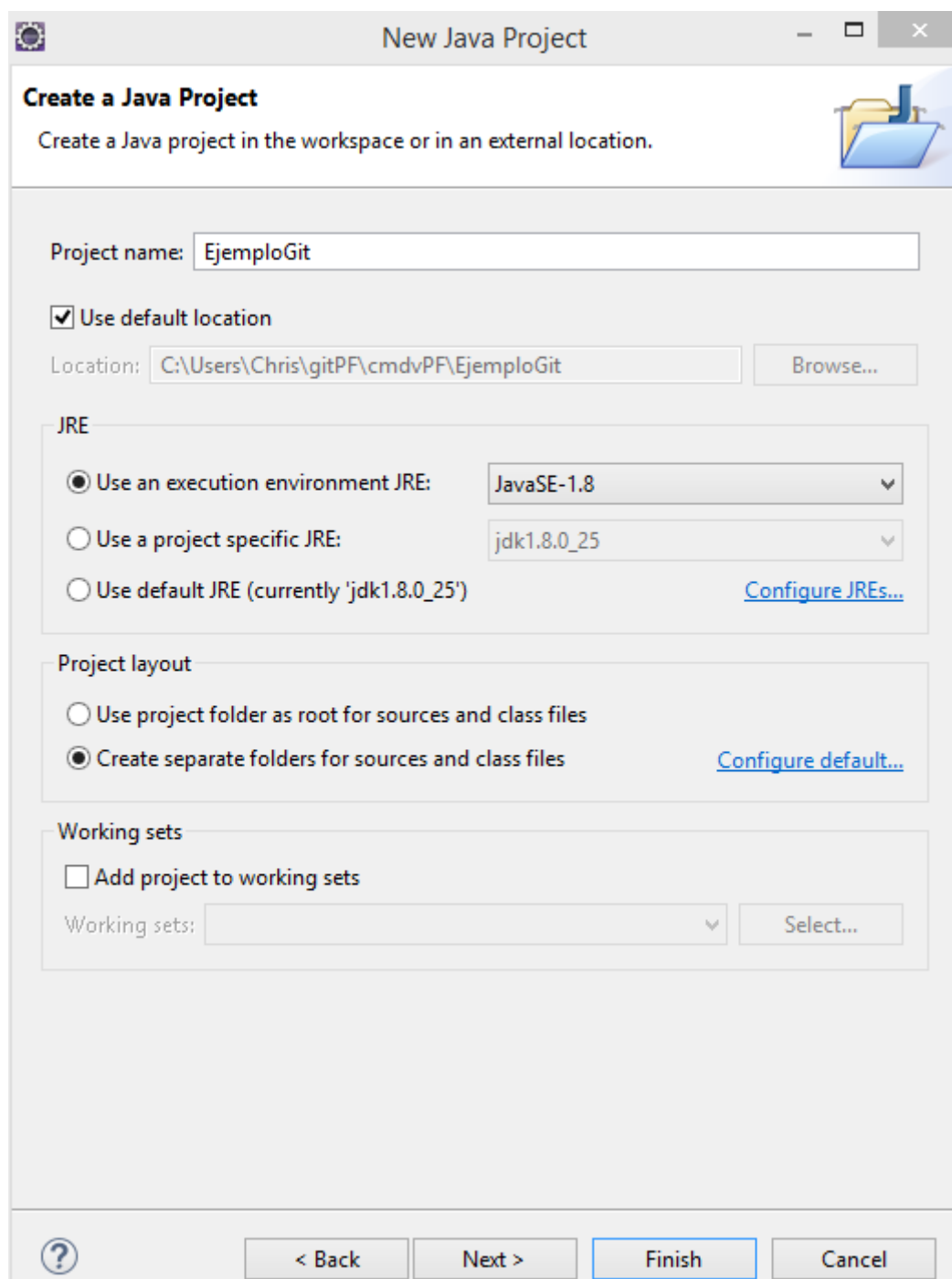
Seleccionamos Git Repository



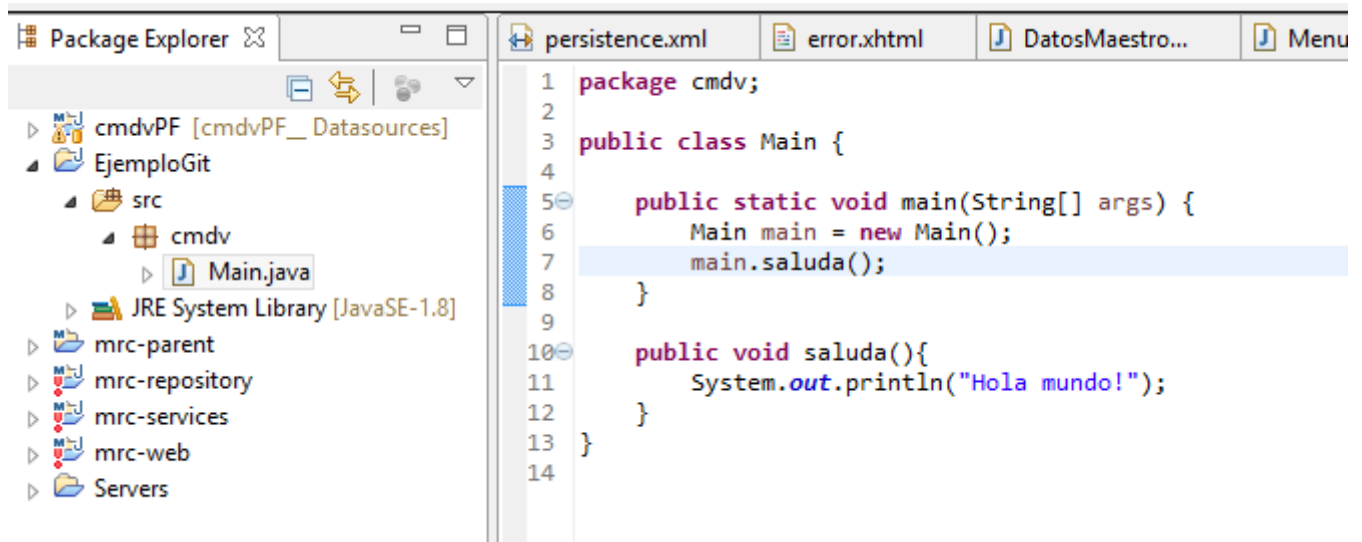
Seleccionamos la ubicación del repositorio en nuestro disco duro:



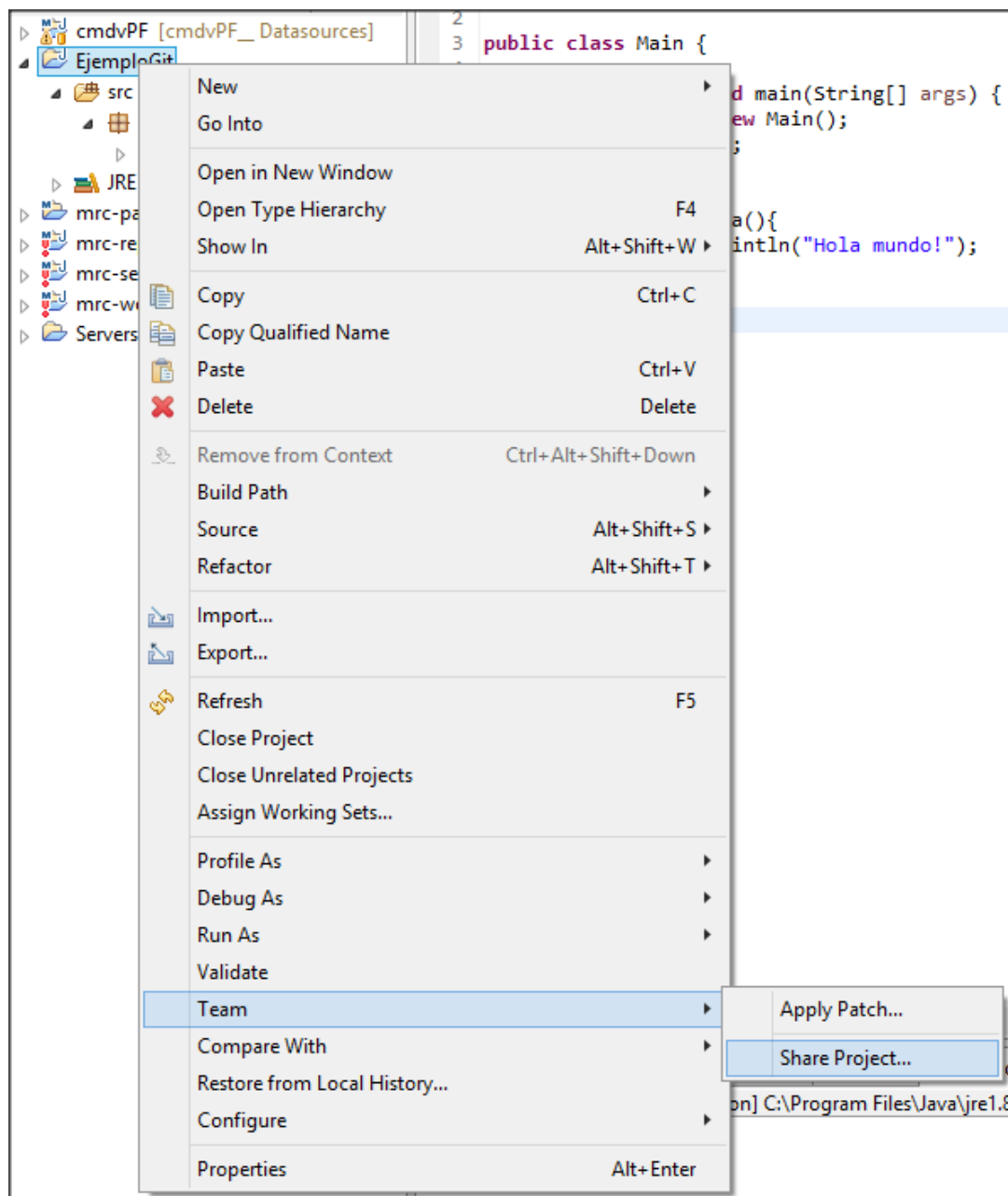
Crearemos un nuevo proyecto y lo subiremos a GitHub, para simplificar será un proyecto java ordinario, pero el procedimiento es similar para cualquier tipo de proyecto.



El código será un simple HolaMundo de java:

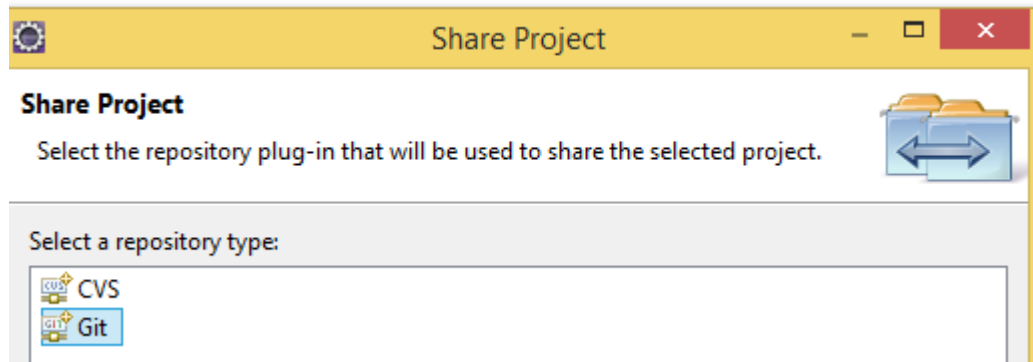


El siguiente paso es subir el proyecto al repositorio, seleccionamos share Project, para ello ponerse sobre el proyecto y pulsar el botón derecho del ratón:

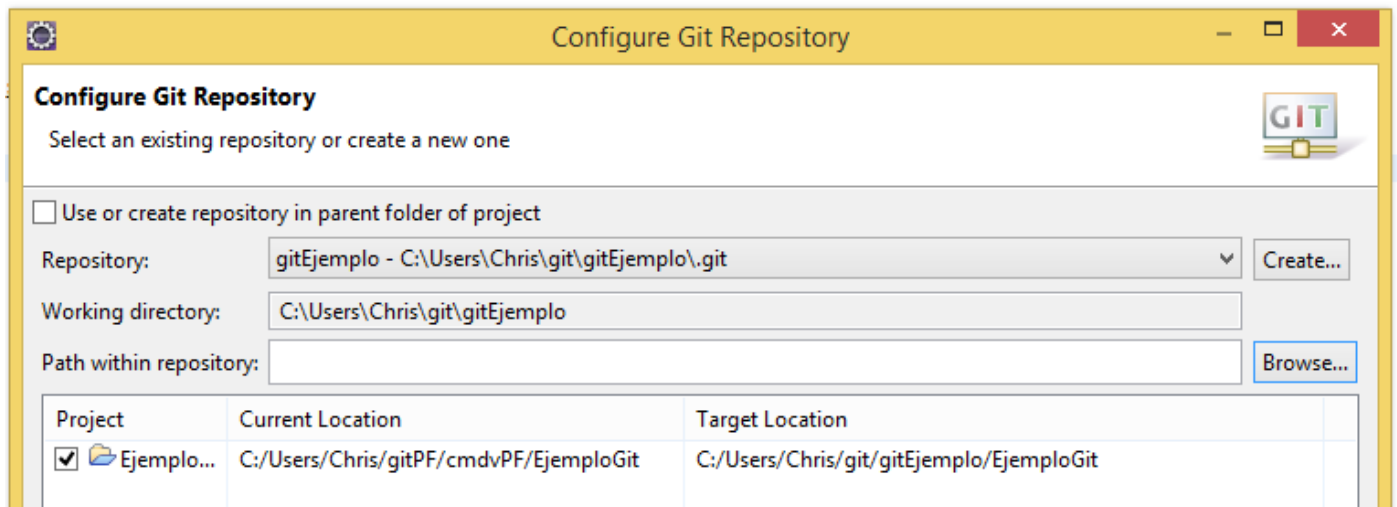




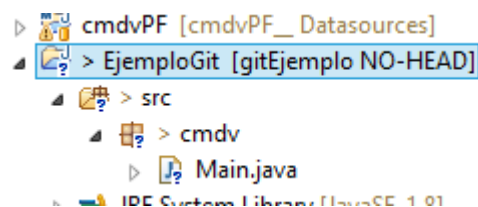
Seleccionamos Git



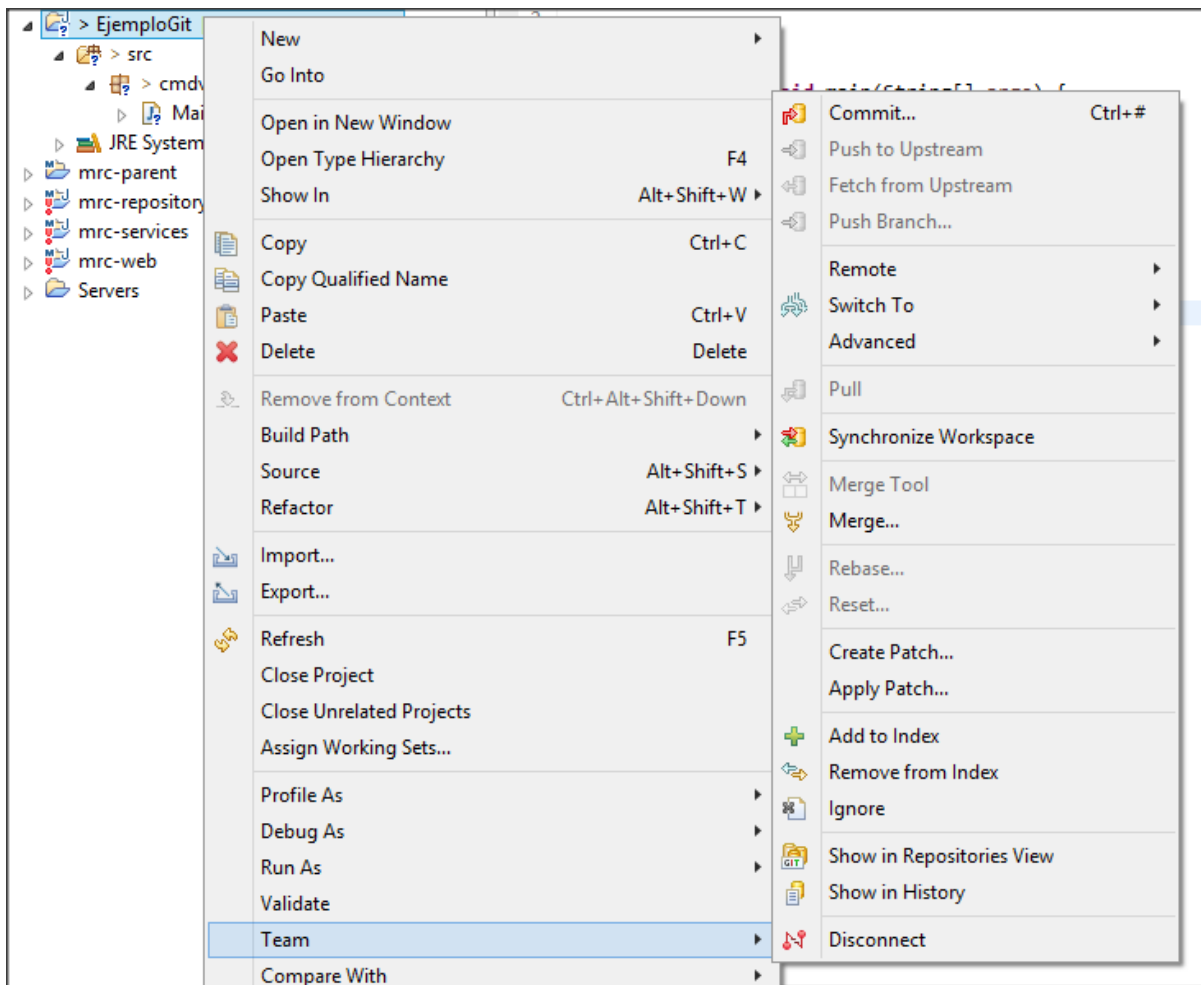
Asignamos la carpeta al repositorio (que también se puede crear desde aquí)



Vemos estos cambios en el proyecto

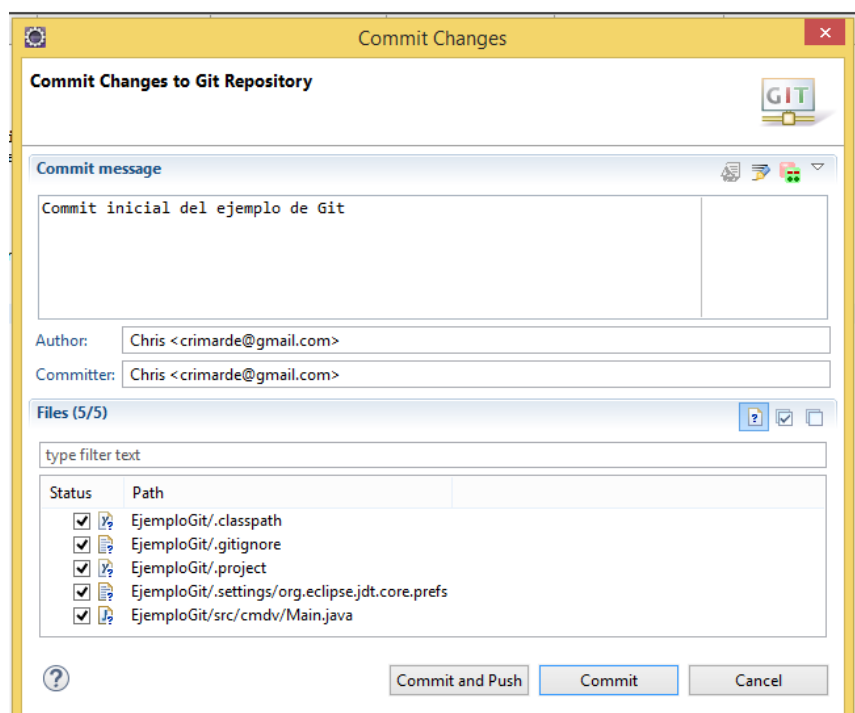


Aparecen nuevas opciones en la opción Team



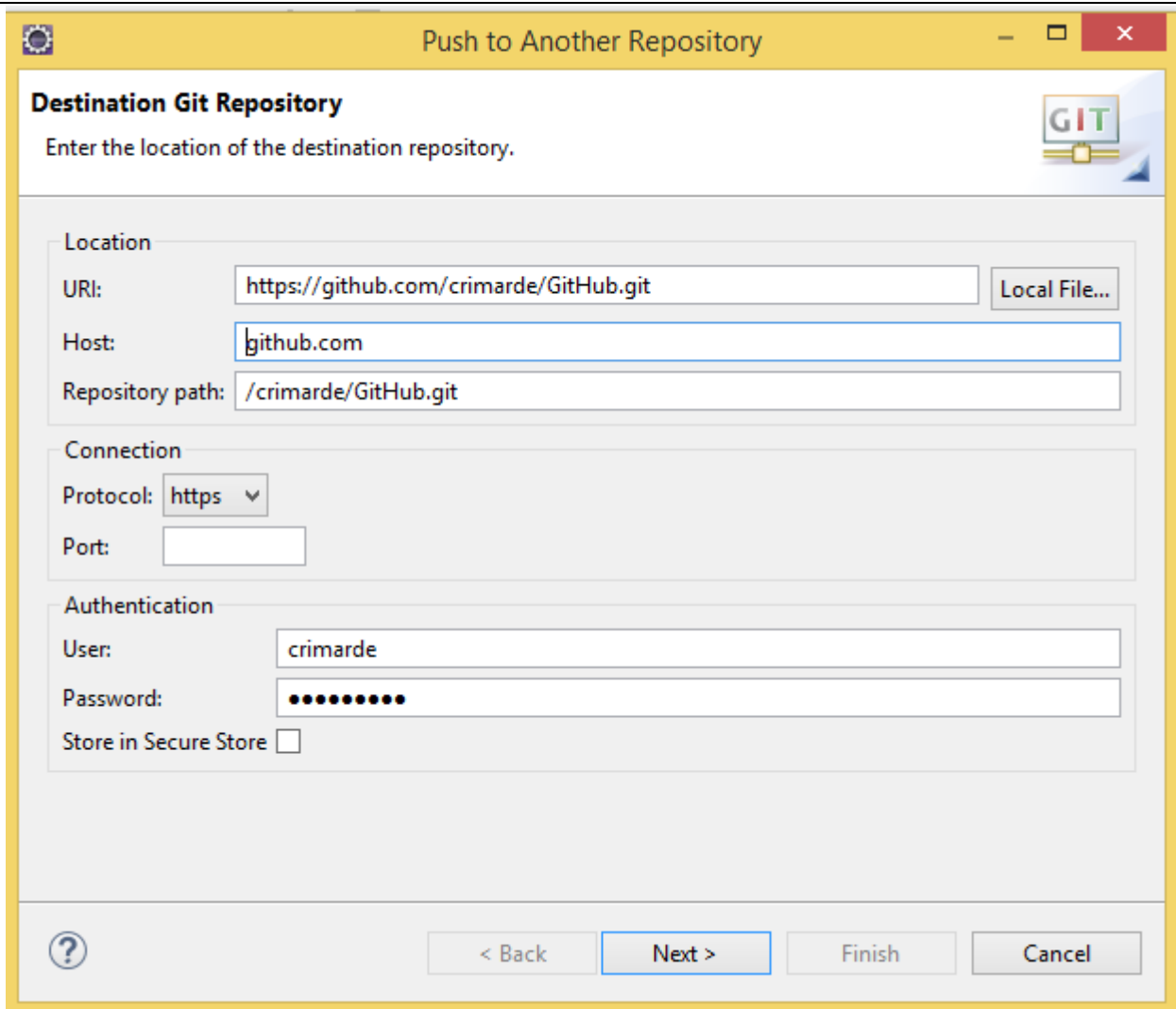
Pulsamos sobre commit, introducimos un mensaje para el commit y seleccionamos los archivos a sincronizar, de esta manera almacenamos los cambios del código, de manera que siempre podamos volver a esta versión.

Pulsamos commit and push



Podemos ver la relación entre la url que habíamos copiado antes y los datos necesarios para configurar el repositorio <https://github.com/crimarde/GitHub.git>

Simplemente copiando la url en el campo URI se autocompletan la mayor parte de los campos de la siguiente pantalla.



**Push to Another Repository**

**Destination Git Repository**  
Enter the location of the destination repository.

**Location**

URI:

Host:

Repository path:

**Connection**

Protocol:

Port:

**Authentication**

User:

Password:

Store in Secure Store ☐

<https://github.com/crimarde/GitHub.git>

La url se descompone en los tres parámetros que necesitamos:

protocolo -> https

host -> github.com

repository -> crimarde/GitHub.git

Push to: <https://github.com/crimarde/GitHub.git>

### Push Ref Specifications

Select refs to push.

**Add create/update specification**

Source ref:  Destination ref:

**Add delete ref specification**

Remote ref to delete:

**Add predefined specification**

**Specifications for push**

Mode	Source Ref	Destination Ref	Force Update	Remove
+ Update	refs/heads/*	refs/heads/*	<input type="checkbox"/>	
+ Update	refs/tags/*	refs/tags/*	<input type="checkbox"/>	

Vemos el mensaje de confirmación

Push Results: <https://github.com/crimarde/GitHub.git>


### Pushed to <https://github.com/crimarde/GitHub.git>

master: master [new branch]

**Message Details**

Repository <https://github.com/crimarde/GitHub.git>

La magia está hecha, ahora si vamos a nuestro repositorio de github tendremos disponible el código del proyecto.

Un

**Description**

Short description of this repository

**Website**

Website for this repository (optional)

Save

 or [Cancel](#)

 [GitHub / +](#) 


Commit inicial del ejemplo de Git

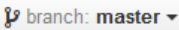

 **crimarde** authored 11 minutes ago latest commit d97d89c20c 

 [EjemploGit](#) Commit inicial del ejemplo de Git 10 minutes ago



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




Unwatch 1 ★ S

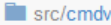
 [GitHub / EjemploGit / +](#) 


Commit inicial del ejemplo de Git


 **crimarde** authored 12 minutes ago latest commit d97d89c20c 


..

 [.settings](#) Commit inicial del ejemplo de Git 11 minutes ago

 [src/cmdv](#) Commit inicial del ejemplo de Git 11 minutes ago

 [.classpath](#) Commit inicial del ejemplo de Git 11 minutes ago

 [.gitignore](#) Commit inicial del ejemplo de Git 11 minutes ago

 [.project](#) Commit inicial del ejemplo de Git 11 minutes ago

Entramos dentro del paquete src/cmdv y podemos ver el contenido de la clase java.

branch: master

GitHub / EjemploGit / src / cmdv / Main.java



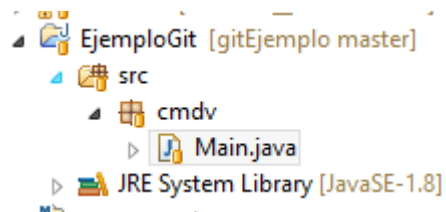
crimarde 11 minutes ago Commit inicial del ejemplo de Git

1 contributor

14 lines (10 sloc) | 0.204 kb

```
1 package cmdv;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Main main = new Main();
7         main.saluda();
8     }
9
10    public void saluda(){
11        System.out.println("Hola mundo!");
12    }
13 }
```

Vemos que aparece el símbolo de sincronización, en Eclipse, lo que indica que el código está sincronizado con el repositorio.



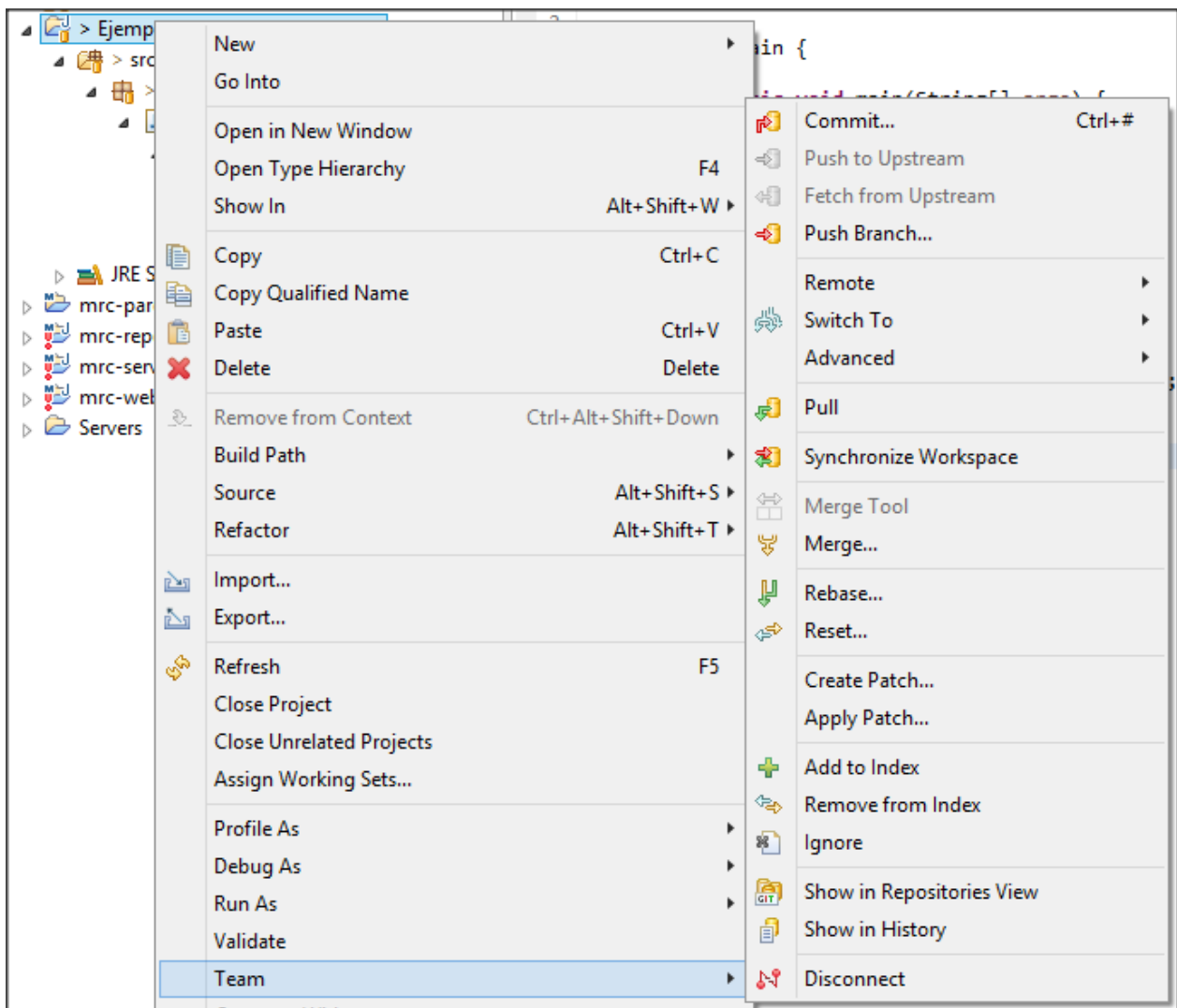
## Hacer cambios y sincronizarlos con el repositorio

Aquí tenemos dos opciones almacenar los cambios en el repositorio local (commit) o subirlos al repositorio de git (push).

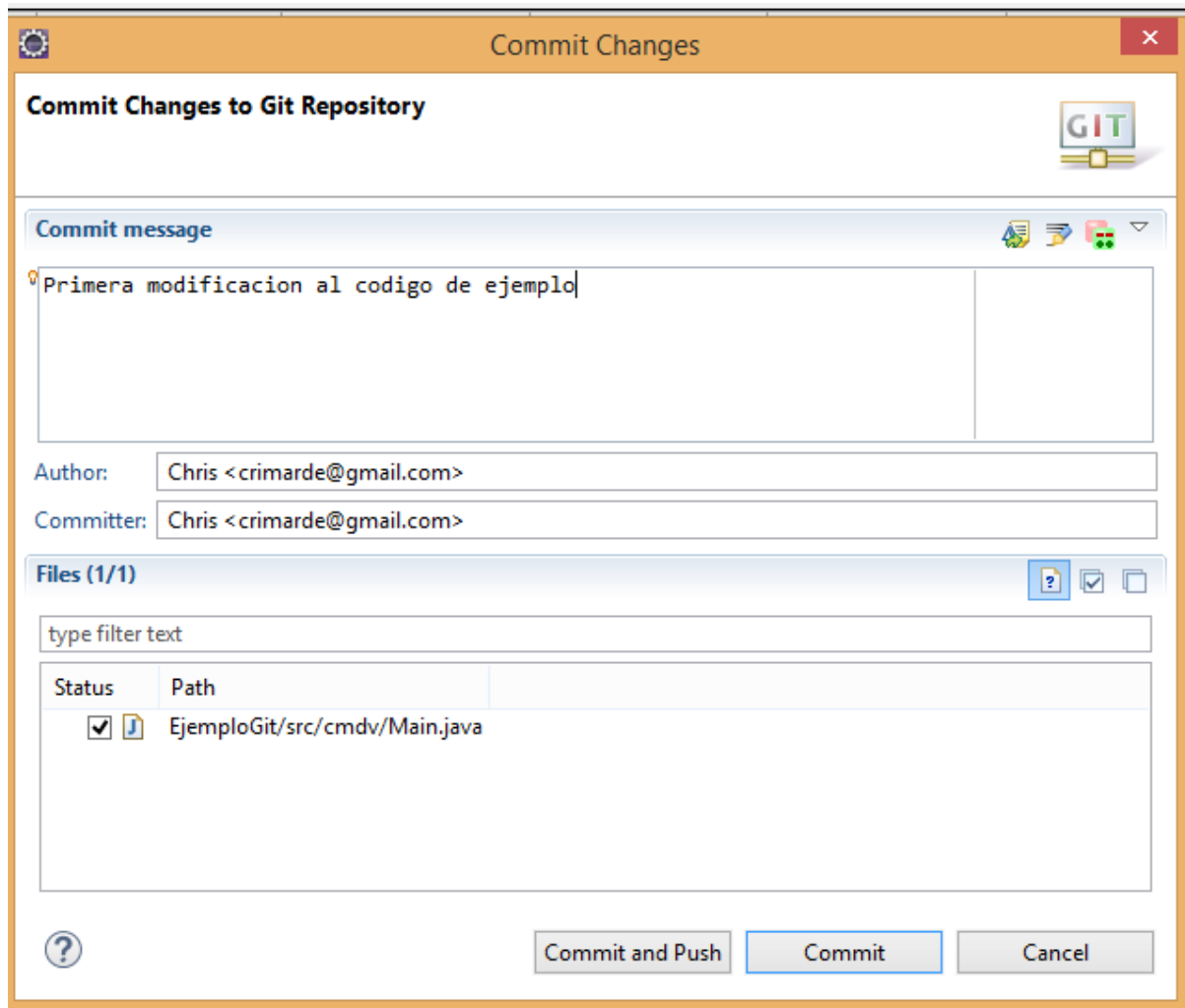
Hacemos algunos cambios al código:

```
package cmdv;  
  
public class Main {  
  
    public static void main(String[] args) {  
        Main main = new Main();  
        main.saluda();  
    }  
  
    public void saluda(){  
        // Modificación  
        System.out.println("Hola mundo!");  
    }  
  
    public void nuevoMetodo(){  
        System.out.println("Soy un nuevo método");  
    }  
}
```

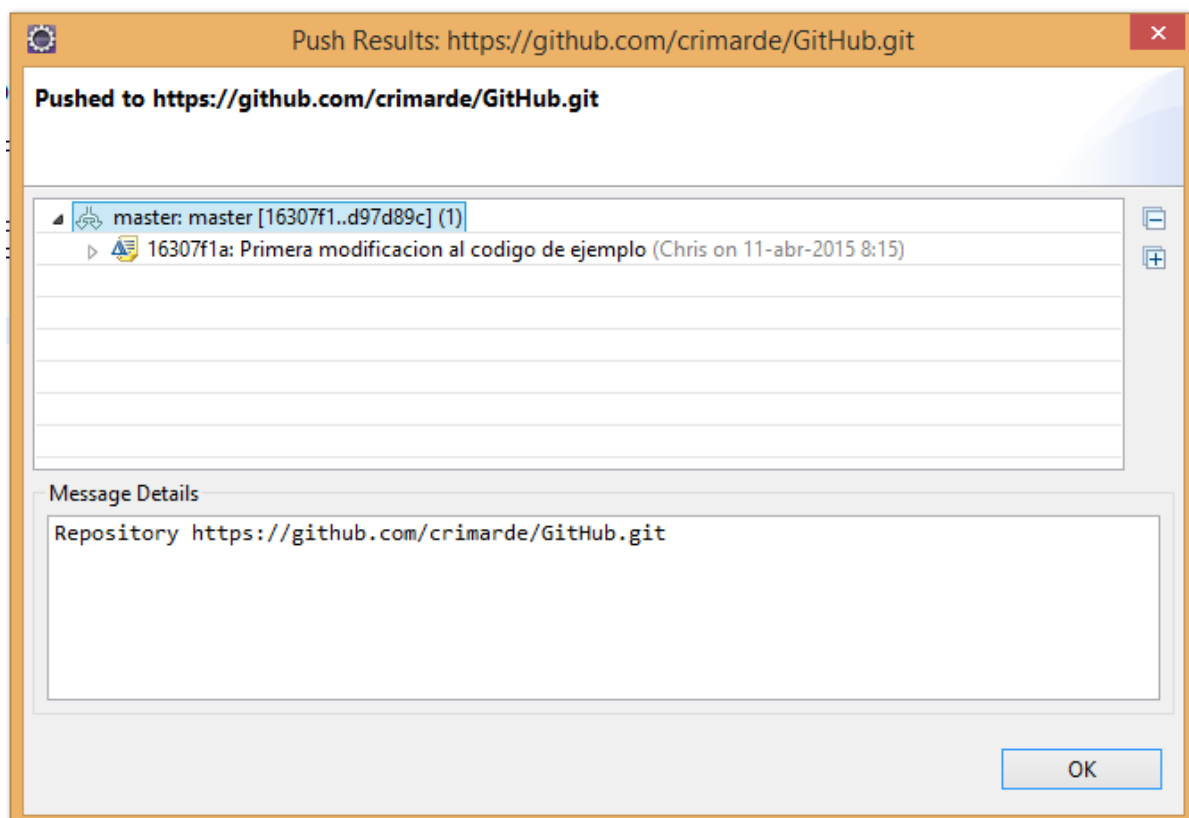
Hacemos commit



Introducimos el prescriptivo mensaje de commit para identificar los cambios posteriormente:



Pulsamos sobre commit and push y podemos ver la pantalla con el mensaje de confirmación.






Si vamos a nuestro repositorio de GitHub podremos ver los cambios realizados



branch: master

GitHub / EjemploGit / src / cmdv / Main.java

 **crimarde** 2 minutos ago Primera modificacion al codigo de ejemplo

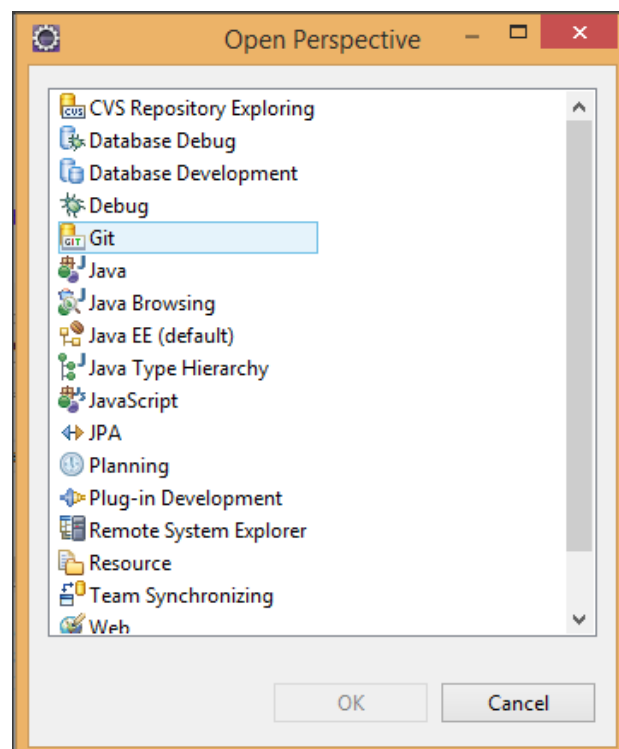
1 contributor

19 lines (14 sloc) | 0.305 kb

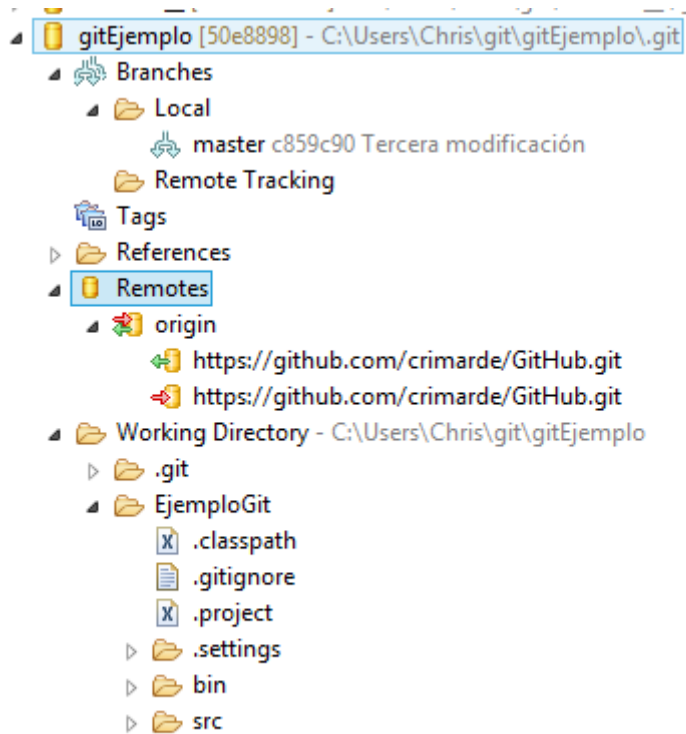
```
1 package cmdv;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Main main = new Main();
7         main.saluda();
8     }
9
10    public void saluda(){
11        // Modificación
12        System.out.println("Hola mundo!");
13    }
14
15    public void nuevoMetodo(){
16        System.out.println("Soy un nuevo método");
17    }
18 }
```

## Perspectiva Git

Podemos abrir la perspectiva de Git para tener más opciones

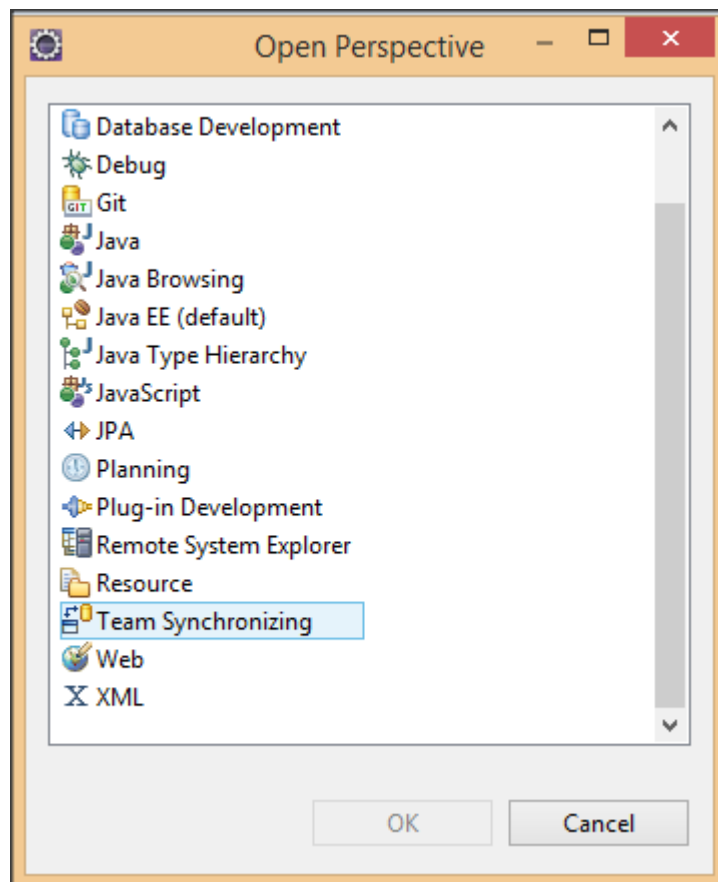


Vista del proyecto en esta perspectiva



Hacemos una nueva modificación al código y vemos que la perspectiva nos informa que hay cambios para subir.

Perspectiva Team Synchronizing

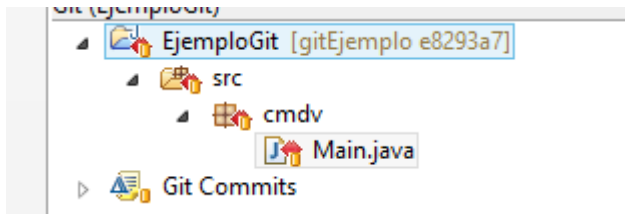


En esta perspectiva se nos informa de los cambios que hay pendientes de subir al repositorio o bajar si es que colaboramos con otros desarrolladores.

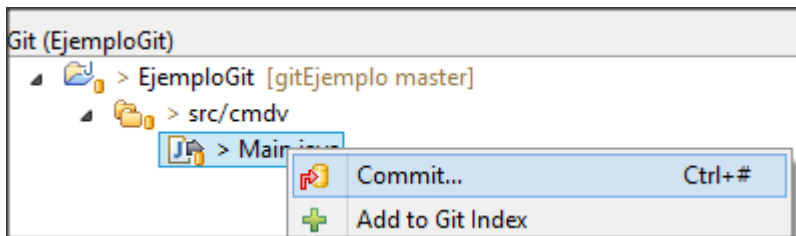
La flecha gris hacia la derecha significa que hay cambios pendientes de subir

La flecha azul hacia la izquierda significa que hay cambios pendientes de bajar.

La doble flecha roja significa que hay conflictos (hay cambios en la misma línea que ha modificado otro desarrollador).

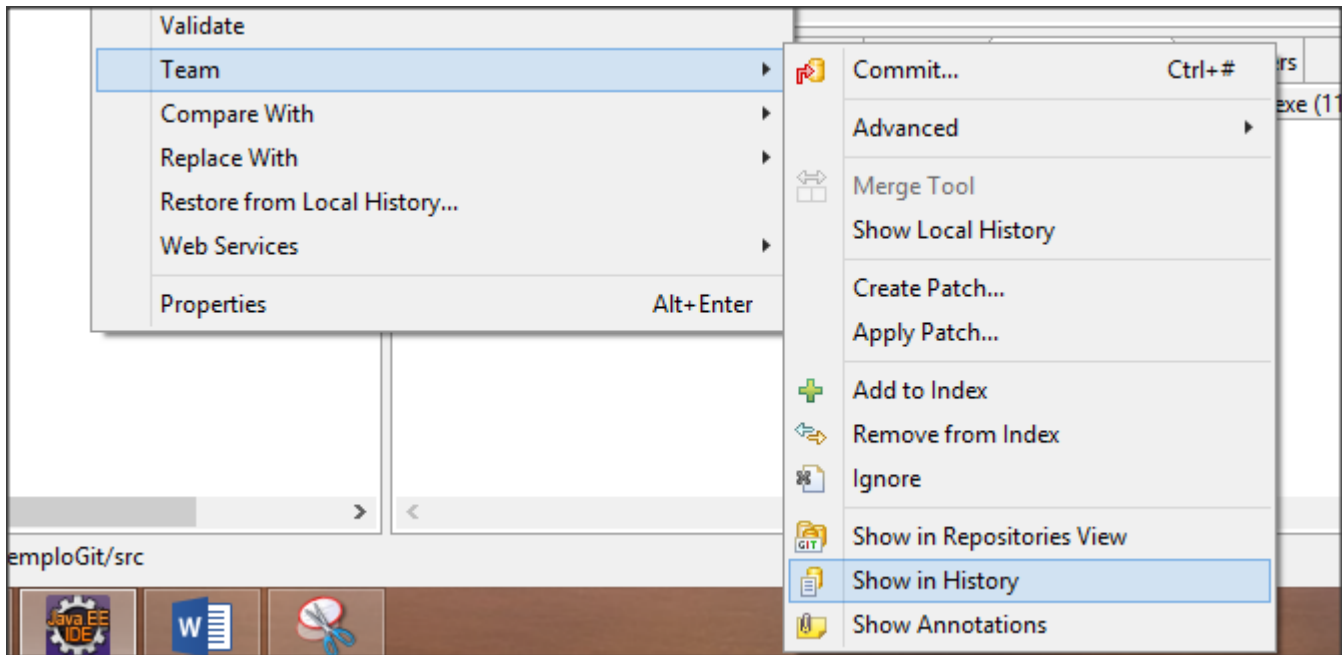


Pulsamos commit



## Volver a una versión anterior

Volvemos a ver la historia del archivo:



Podemos ver todos los commits realizados

Problems Javadoc Declaration Search Console Servers History

File: EjemploGit/src/cmdv/Main.java [gitEjemplo]

Id	Message	Author	Authored Date	Committer
16307f1	<a href="#">master</a> (HEAD) Primera modificacion al codigo de ejemplo	Chris	6 minutes ago	Chris
d97d89c	Commit inicial del ejemplo de Git	Chris	34 minutes ago	Chris

commit 16307f1a7ad5ac9634ae0dae8effeafeb2628f2b  
Author: Chris <crimarde@gmail.com> 2015-04-11 08:15:45  
Committer: Chris <crimarde@gmail.com> 2015-04-11 08:15:45  
Parent: [d97d89c20ceb3427667ad85c7d32a545f7e3a2e2](#) (Commit inicial del ejemplo de Git)  
Branches: [master](#)

Primera modificacion al codigo de ejemplo

```
----- EjemploGit/src/cmdv/Main.java -----
diff --git a/EjemploGit/src/cmdv/Main.java b/EjemploGit/src/cmdv/Main.java
index a507795..9d17c04 100644
--- a/EjemploGit/src/cmdv/Main.java
+++ b/EjemploGit/src/cmdv/Main.java
@@ -8,6 +8,11 @@
 }

 public void saluda(){
+ // Modificación
   System.out.println("Hola mundo!");
 }
+ public void nuevoMetodo(){
+   System.out.println("Soy un nuevo método");
+ }
 }
```

EjemploGit/src/cmdv/Main.java

Seleccionamos el commit al que queremos volver

gitEjemplo			
type filter text			
Commit	Commit Message	Date	Reflog Message
c859c90	Tercera modificación	2015-04-11 08:29:54	commit: Tercera modificación
16307f1	Primera modificacion al codigo de ejemplo	2015-04-11 08:15:45	commit: Primera modificacion al codigo de ejemplo
d97d89c	Commit inicial del ejemplo de Git	2015-04-11 07:48:51	commit (amend): Commit inicial del ejemplo de Git
50e8898	Commit inicial del ejemplo de Git	2015-04-11 07:47:35	commit (initial): Commit inicial del ejemplo de Git

Hacemos checkOut sobre la primera versión del archivo

d97d89c	Commit inicial del ejemplo de Git
50e8898	Commit inicial del ejemplo de Git

Checkout

Open in Commit Viewer

Copy SHA-1

Reset

Ctrl+C

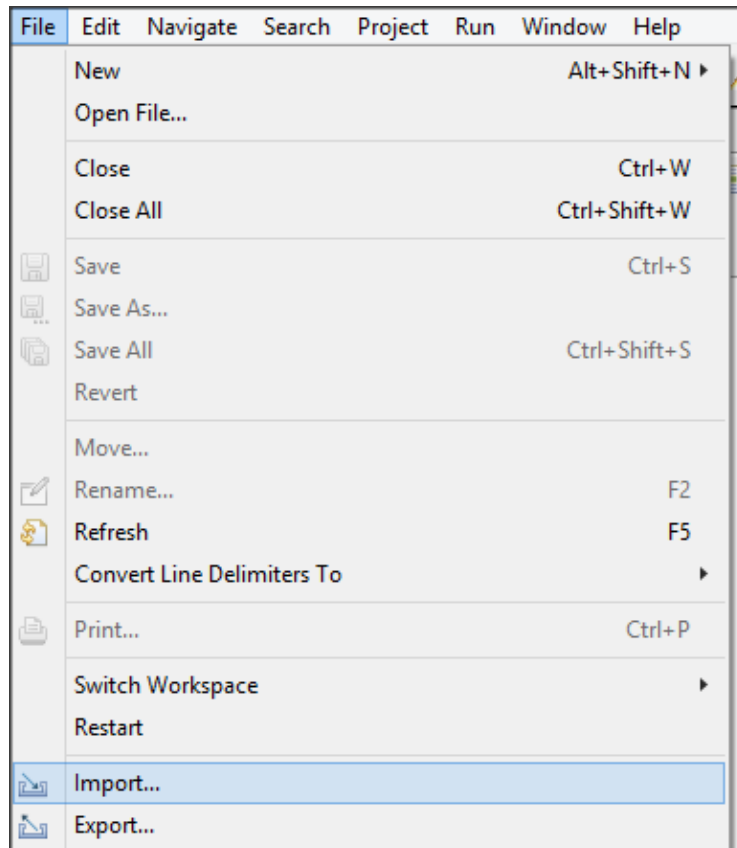
Y el código vuelve a la versión seleccionada

```
persistence.xml  error.xhtml  DatosMaestro...  Mer
1 package cmdv;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Main main = new Main();
7         main.saluda();
8     }
9
10    public void saluda(){
11        System.out.println("Hola mundo!");
12    }
13 }
14
```

## Descargar un proyecto existente (simula dos desarrolladores sobre el mismo proyecto)

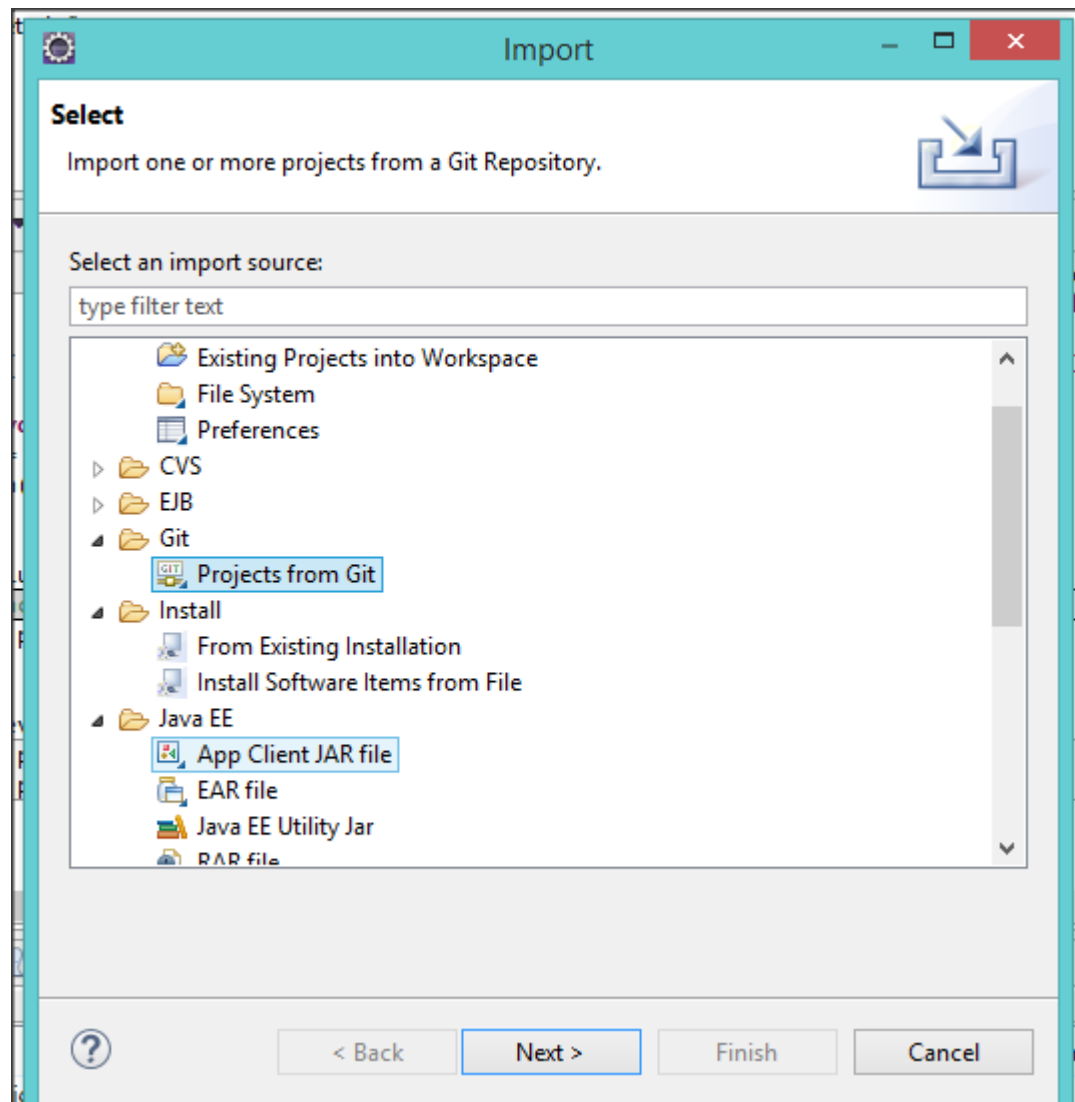
---

Vamos a descargar el código en un nuevo proyecto (Hay que cambiar de workspace porque en el actual ya existe el proyecto)

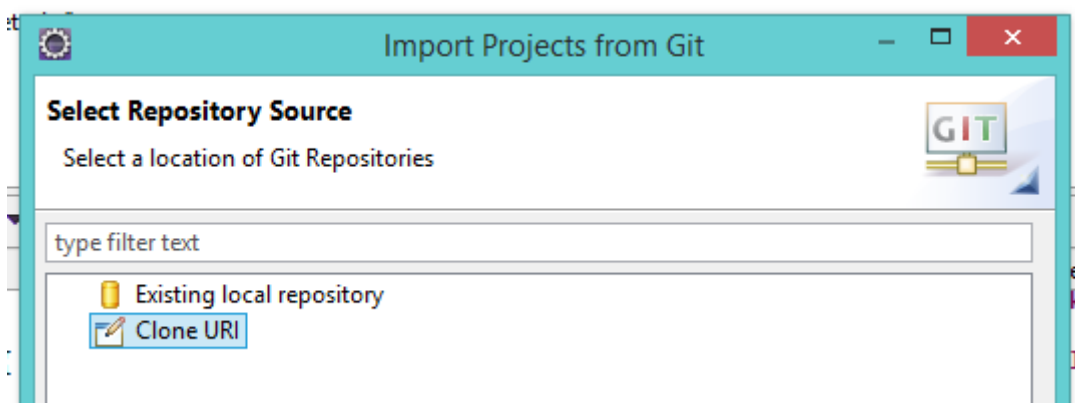


Simplemente hay que seguir los pasos que se muestran en las imágenes siguientes:

Importar el proyecto de Git



Seleccionar "Clone URI"



Introducir la URL que copiamos al crear el repositorio en la web de GitHub

The screenshot shows the 'Import Projects from Git' dialog box with the 'Source Git Repository' tab selected. The dialog has a teal header bar with the title 'Import Projects from Git' and standard window controls. Below the header, the tab title 'Source Git Repository' is displayed next to a Git logo. The main area contains three sections: 'Location', 'Connection', and 'Authentication'. In the 'Location' section, the 'URI' field is filled with 'https://github.com/crimarde/GitHub.git', and a 'Local File...' button is to its right. The 'Host' field contains 'github.com' and the 'Repository path' field contains '/crimarde/GitHub.git'. In the 'Connection' section, the 'Protocol' dropdown is set to 'https' and the 'Port' field is empty. In the 'Authentication' section, the 'User' field contains 'crimarde', the 'Password' field is masked with dots, and the 'Store in Secure Store' checkbox is checked. At the bottom, there is a help icon, a '< Back' button, a 'Next >' button, and 'Finish' and 'Cancel' buttons.

**Source Git Repository**

Enter the location of the source repository.

**Location**

URI:  

Host:

Repository path:

**Connection**

Protocol:

Port:

**Authentication**

User:

Password:

Store in Secure Store ☒

Seleccionar la rama que queremos importar, en este caso sólo tenemos la rama principal (HEAD)

The screenshot shows the 'Import Projects from Git' dialog box with the 'Branch Selection' tab selected. The dialog has the same teal header bar. The tab title 'Branch Selection' is displayed next to a Git logo. The main area contains instructions: 'Select branches to clone from remote repository. Remote tracking branches will be created to track updates for these branches in the remote repository.' Below this, it says 'Branches of https://github.com/crimarde/GitHub.git:' followed by a search filter input field containing 'type filter text'. A list of branches is shown below the filter, with the 'master' branch selected, indicated by a checked checkbox and a branch icon.

**Branch Selection**

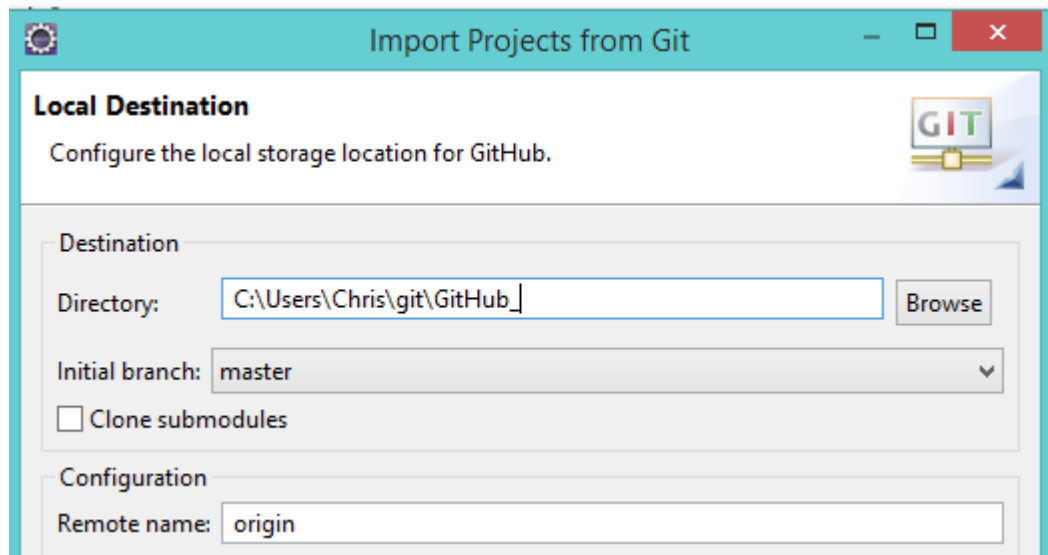
Select branches to clone from remote repository. Remote tracking branches will be created to track updates for these branches in the remote repository.

Branches of https://github.com/crimarde/GitHub.git:

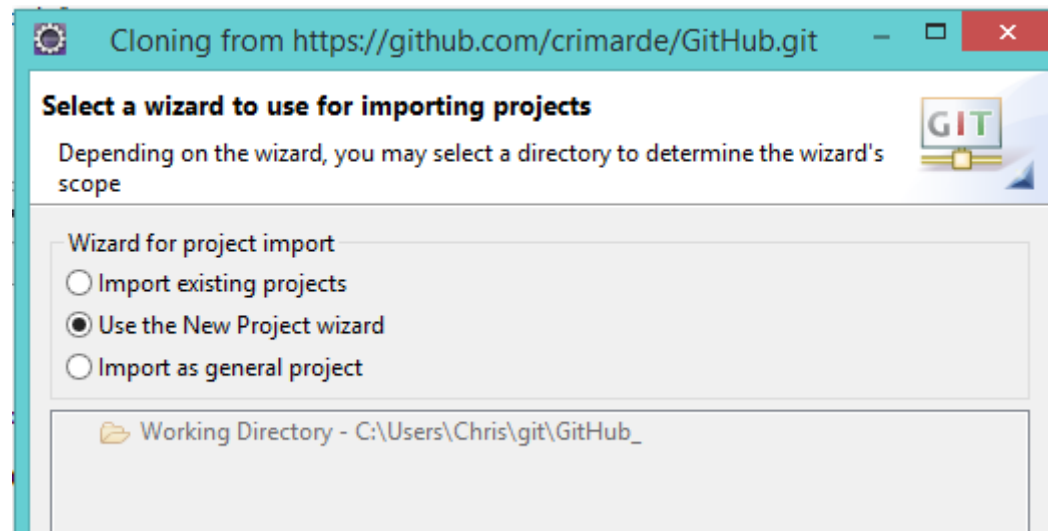
☒ master



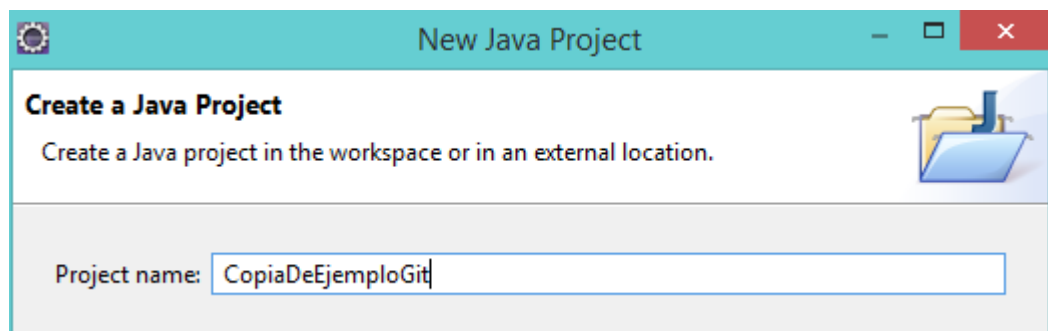
Seleccionar la carpeta del equipo en la que importar el proyecto



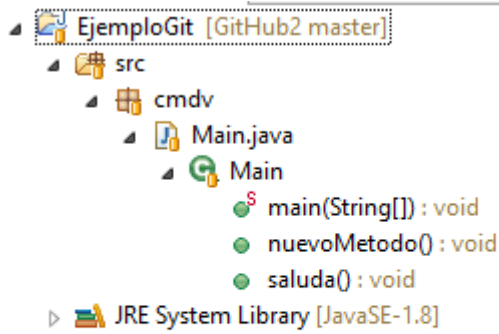
Crear un nuevo proyecto (en este caso usamos el asistente)



Crear el proyecto java



Este es el resultado:



Hagamos un cambio, sincronizemos y veamos que pasa en el proyecto original:

El código queda como sigue en la siguiente imagen:

```
main.java ↵
package cmdv;

public class Main {

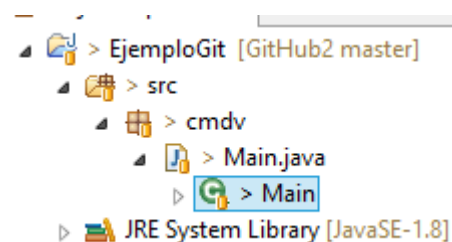
    public static void main(String[] args) {
        Main main = new Main();
        main.saluda();
    }

    public void saluda(){
        // Modificación
        System.out.println("Hola mundo!");
    }

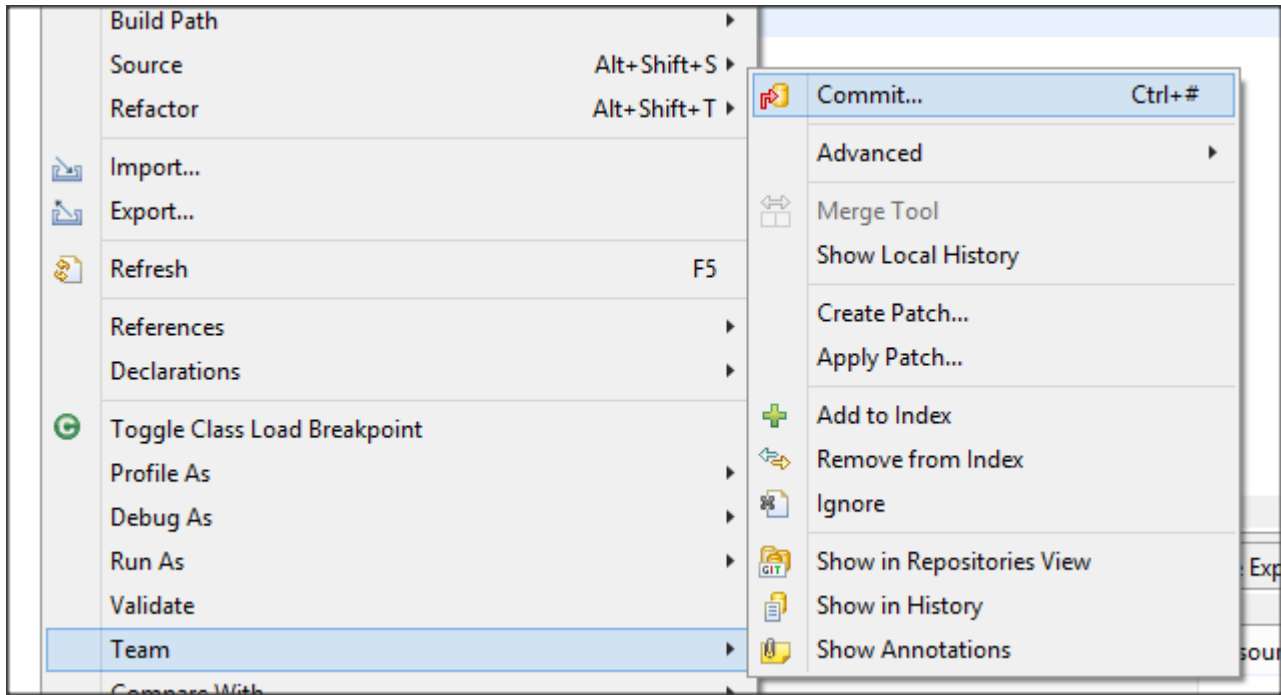
    public void nuevoMetodo(){
        System.out.println("Soy un nuevo método");
        System.out.println("Soy un nuevo método modificado en el repositorio");
    }

    public void despide(){
        System.out.println("Adios!!!");
    }
}
```

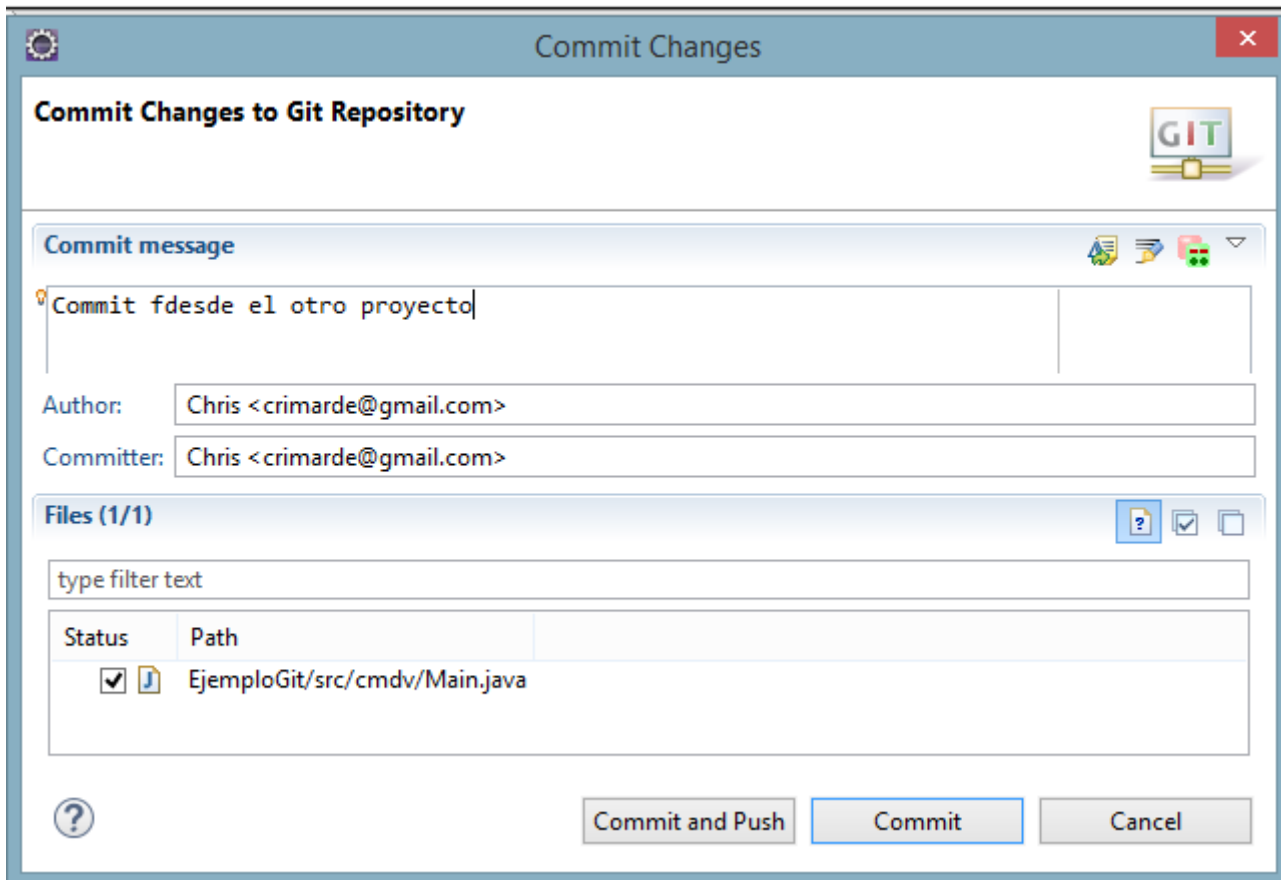
Vemos que hay cambios pendientes por subir con el símbolo >



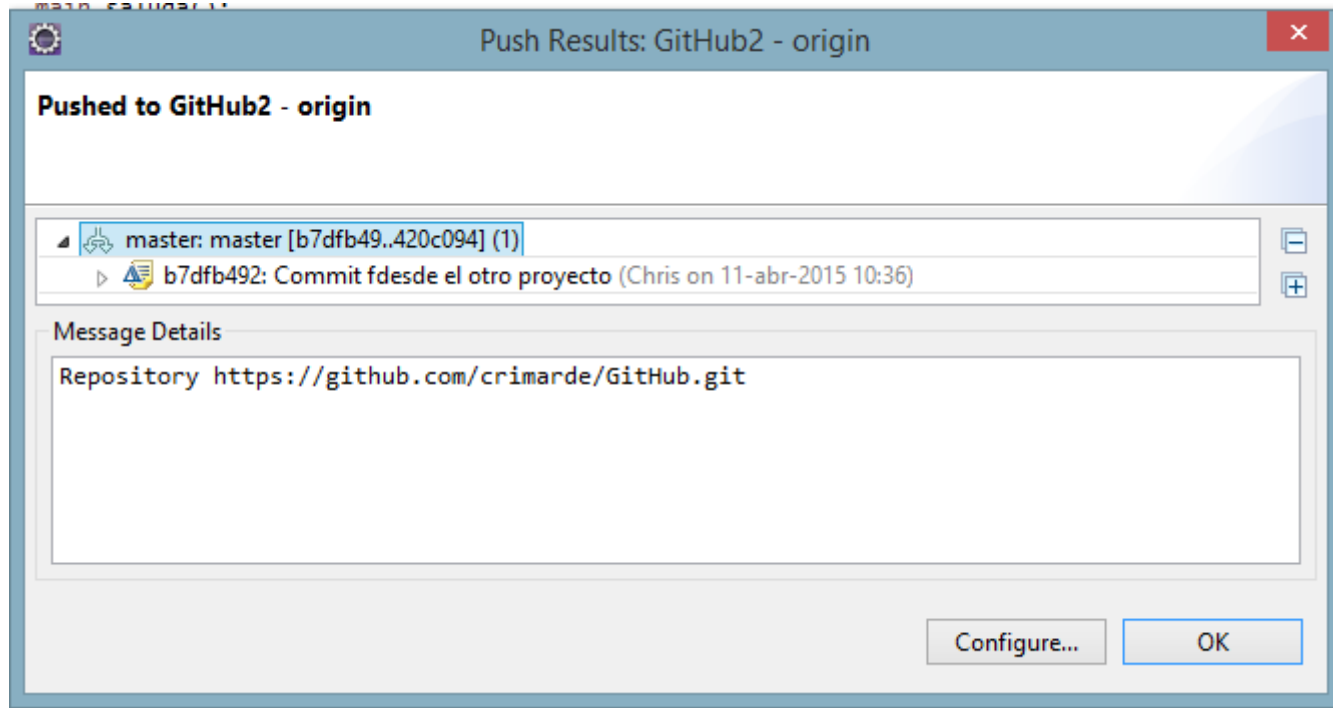
Hacemos commit



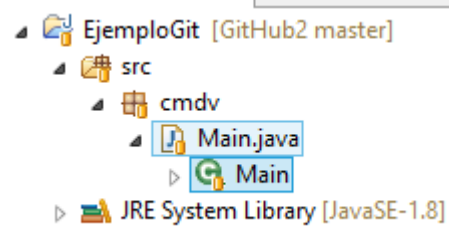
Introducimos un comentario que permita identificar los cambios realizados en el commit:



Mensaje de ok



Desaparecen las marcas de cambios pendientes



Si vamos al repositorio en la página de GitHub, podremos ver que se han sincronizado los cambios que hemos realizado. Este es el código en el repositorio

branch: master **GitHub / EjemploGit / src / cmdv / Main.java**

 **crimarde** a minute ago Commit fdesde el otro proyecto

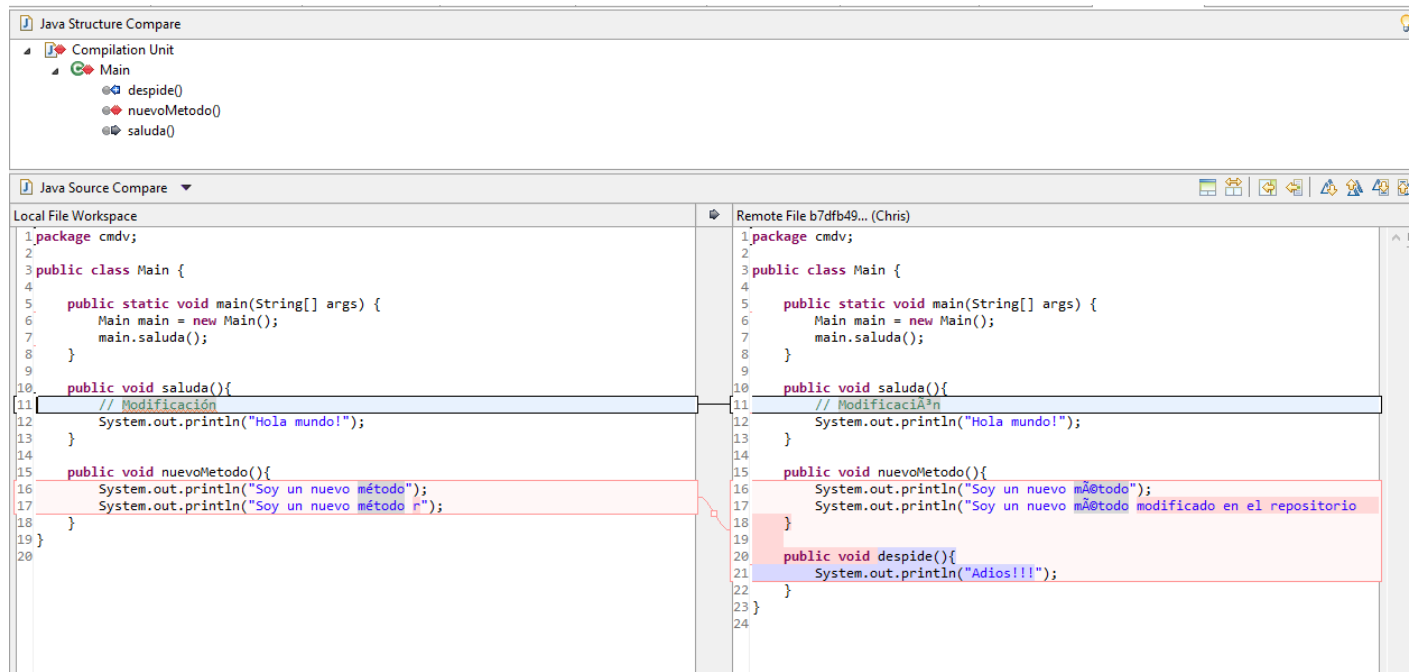
1 contributor

24 lines (18 sloc) | 0.454 kb Raw Blame

```
1 package cmdv;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         Main main = new Main();
7         main.saluda();
8     }
9
10    public void saluda(){
11        // Modificación
12        System.out.println("Hola mundo!");
13    }
14
15    public void nuevoMetodo(){
16        System.out.println("Soy un nuevo método");
17        System.out.println("Soy un nuevo método modificado en el repositorio ");
18    }
19
20    public void despide(){
21        System.out.println("Adios!!!");
22    }
23 }
```

Ahora vamos ver cómo afectan los cambios que hemos hecho en el otro workspace, el que simulaba otro ordenador. Volvemos al otro workspace

Vemos que hay cambios entrantes salientes y colisiones



## Resumen

Hemos visto:

- Cómo crear un repositorio, tanto en la web de GitHub como en local.
- Cómo sincronizar los cambios en un proyecto java con los repositorios (commit y push)
- Cómo volver a una versión anterior del código
- Cómo mantener sincronizado el código entre dos o más programadores solventando los conflictos.

## Alternativa

Después de ver el funcionamiento de GitHub a través de eclipse no querría terminar esta parte sin como mínimo comentar que GitHub permite descargar el proyecto e importarlo como un proyecto normal en eclipse, aunque evidentemente de esta manera perdemos el control de versiones.

crimarde / GitHub

Unwatch 1★ Star 0Fork 0

Description

Short description of this repository

Website

Website for this repository (optional)

Save

or 

Cancel

7 commits

1 branch

0 releases

1 contributor

branch: master

GitHub / +

Commit desde el otro proyecto

crimarde

 authored 20 minutes ago

latest commit b7dfb49202

EjemploGit

Commit desde el otro proyecto

20 minutes ago

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

Add a README

<> Code

Issues 0

Pull requests 0

Wiki

Pulse

Graphs

Settings

HTTPS clone URL

https://github.com/crimarde/GitHub

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

Clone in Desktop

Download ZIP