

Tema 2B – Control de versiones de código (Git)

Unai Aguilera Irazabal

unai.aguilera@deusto.es

Grado en Ingeniería Informática

Facultad de Ingeniería – Universidad de Deusto

¿Qué es Git?

- Sistema de control de versiones de código:
 - El código pasa por muchas fases durante su desarrollo
 - Los sistemas de control de versiones permiten guardar el estado por el que va pasando el código
 - Se pueden recuperar estados anteriores, comparar versiones, conocer las contribuciones de un desarrollador, etc.
 - Varios desarrolladores pueden trabajar simultáneamente y después integrar los cambios
 - Al ser distribuido todos los desarrolladores tienen una copia del repositorio. No necesitan una conexión a internet para trabajar (sí para obtener los cambios de otros desarrolladores o publicar los suyos).

Repositorios de Git

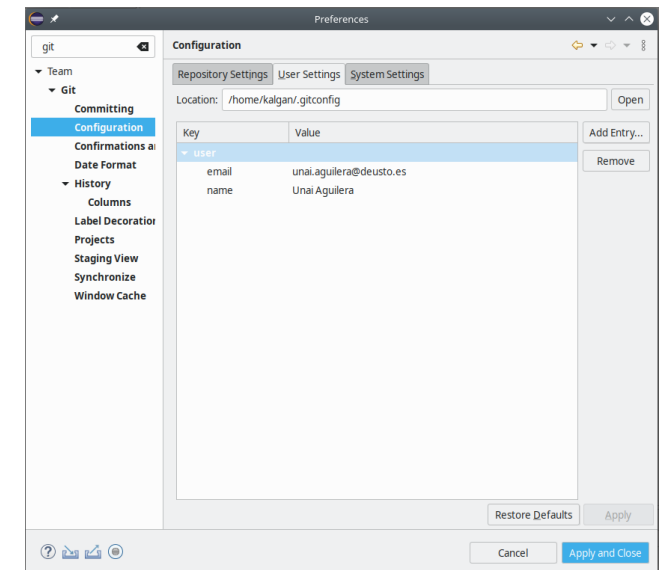
- Existen varios servicios en internet que permiten crear repositorios de Git alojados en la nube y accesibles remotamente.
- Los más usados:
 - GitHub: <https://github.com>
 - Bitbucket: <https://bitbucket.org>
 - GitLab: <https://gitlab.com>
- Existe también la posibilidad de instalar el gestor de repositorios en un servidor propio. Útil para empresas que no quieren publicar su código en máquinas de terceros.
- En la asignatura vamos a utilizar GitHub para gestionar los proyectos de la asignatura.
 - Así, que si todavía no tienes una cuenta puedes crear una.
 - Si asocias tu cuenta de @opendeusto puedes tener acceso a distintas herramientas <https://education.github.com/pack>

Pasos para el uso de Git

1. Creación del repositorio (por ejemplo, en GitHub.com donde necesitamos una cuenta de usuario)
2. Obtención de la URL del repositorio. Por ejemplo, en GitHub son de la forma `https://GitHub.com/<usuario>/<nombre_repositorio>.git`
3. Si el repositorio es privado, tenemos que tener permisos para leer o escribir del mismo.
4. Necesitamos tener un usuario y contraseña con permisos de escritura en el repositorio.
5. En el ordenador donde vamos a realizar el desarrollo tenemos que tener instalado un cliente de Git
 - Línea de comandos: <https://git-scm.com/download/>
 - Visuales: <https://git-scm.com/downloads/guis>
 - Eclipse: <https://www.eclipse.org/egit/> (ya se encuentra incluido en las últimas versiones de Eclipse). En los siguientes ejemplos vamos a utilizar esta herramienta.

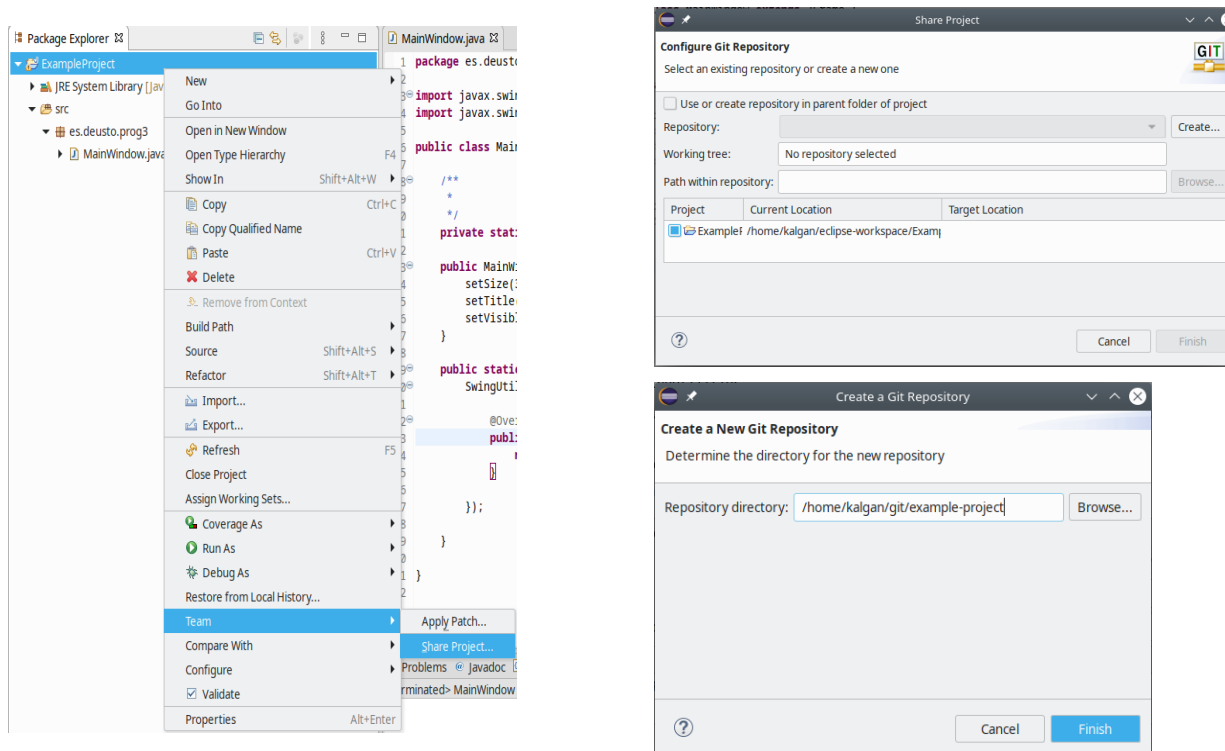
Configuración local de Git

- Antes de empezar a trabajar con Eclipse es necesario indicar en la configuración local el nombre y la dirección de correo con la que queramos que aparezcan los *commits* en el log del repositorio.
 - En Window→Preferences vamos a Team→ Git→ Configuration
 - Debemos añadir dos entradas nuevas con nuestros datos:
 - **user.name**. El nombre que aparecerá en los commits.
 - **user.email**. El email que aparecerá en los commits.
 - Para el proyecto usar nombre real + email @opendeusto.es



Inicialización del proyecto

- En este ejemplo vamos a suponer que ya existe un proyecto local con una estructura básica, pero este proyecto todavía no se ha subido por primera vez al repositorio.
 - Esta tarea la debe realizar por primera vez uno de los miembros del equipo. El resto de miembros "clonará" después el repositorio.



Pulsamos el botón "Create". Esto nos creará un directorio local con los ficheros internos de git.

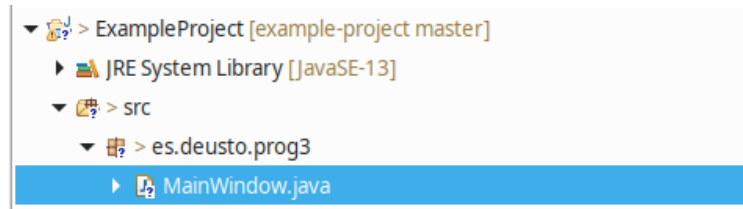
Conviene darle un nombre distinto a la opción que pone por defecto.

El directorio local de git en el ordenador, que contiene nuestros proyectos, en Windows está localizado en el directorio del usuario `C:\Users\<usuario>\git\`

Se debe dar un nombre distinto para el repositorio local de cada proyecto para evitar conflictos de nombres.

Proyecto bajo control de cambios

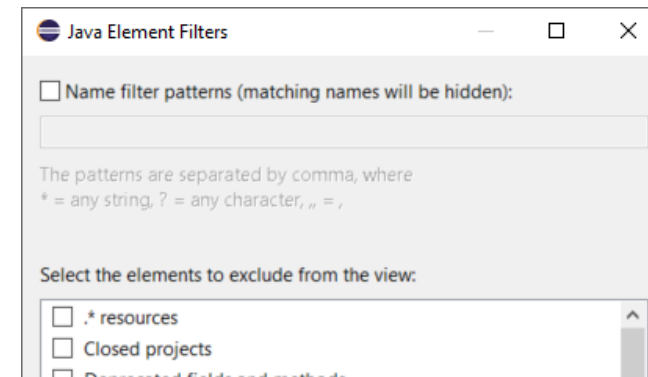
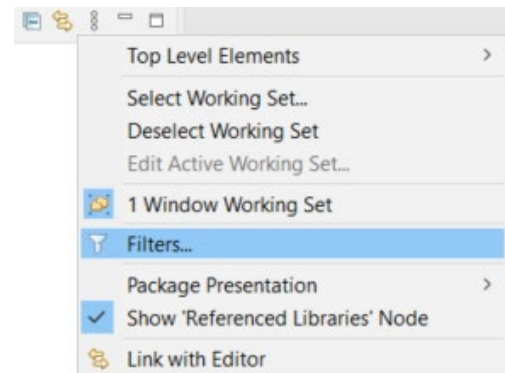
- Para saber en Eclipse si un proyecto ya se encuentra bajo control de cambios (inicializado para trabajar con git):
 - Debemos ver si aparece el nombre del repositorio local y la rama junto al nombre del proyecto.



- En este caso se puede ver que el proyecto está asociado a un repositorio local en un directorio llamada "example-project" y en la rama "master/main".
- Además, vemos que han aparecido una serie de iconos asociados a los directorios y ficheros del proyecto.
- Estos iconos indican el estado en el que se encuentran cada uno de estos ficheros: sin control, sincronizados, con cambios, conflictos, etc.

Fichero .gitignore

- En el directorio del repositorio local de un proyecto de Git, suele existir un fichero llamado .gitignore.
 - Este fichero contiene la lista de directorios y ficheros que no deben ser tenidos en cuenta por el control de cambios.
 - En Eclipse no se muestra este fichero en el proyecto, porque es un fichero oculto (empieza por un punto).
- Para listar los archivos ocultos en el árbol de ficheros
 - Desmarcar en los filtros la opción de ocultar los recursos que empiezan por punto.



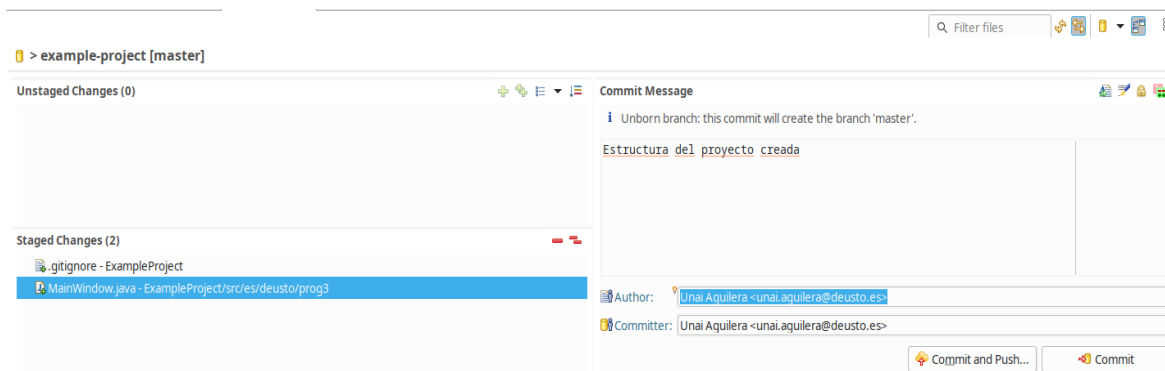
Primer commit

- Aunque hemos inicializado el proyecto para empezar a trabajar con git de forma local, todavía no hemos guardado ningún cambio.
 - Tenemos que hacer un *commit* con los cambios que queremos guardar como primera versión del proyecto.
- Sobre el proyecto, debemos seleccionar la opción Team → Commit



Preparando el commit

- Antes de realizar un commit podemos seleccionar qué ficheros se van a incluir en él, de todos los que están con cambios.
 - Esto se realiza con los botones de "Add to Index".
- Una vez que está preparado el commit debemos:
 - Escribir un mensaje descriptivo para asociar a los cambios que van a ser guardados.
 - Pulsar el botón "Commit".

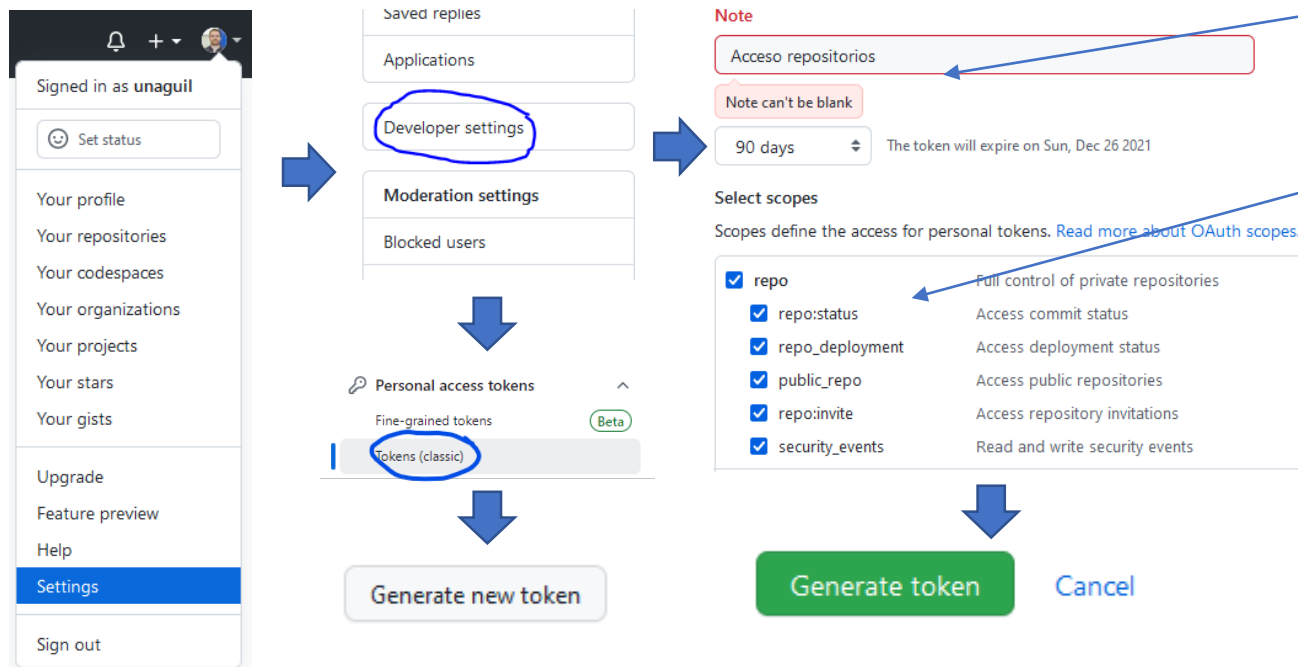


Comprobar que los campos de Author/Committer son correctos.

En caso contrario, rellenarlos a mano o configurar las opciones de Git en Eclipse.

Token para autenticación en GitHub

- Tras una actualización de GitHub en agosto de 2021, ya no está permitido el uso del usuario/contraseña para hacer commit o acceder a repos privados.
- Ahora es necesario utilizar un token personal (Personal Access Token). Para obtener este token desde GitHub



Debemos dar un nombre para identificar el token creado y cambia la caducidad a “No expiration” para evitar tener que regenerar el token.

Debemos marcar el permiso “repo” push (y pull si es privado).

Una vez generado no cierres hasta usarlo en Eclipse ya que no se pueden volver a ver, aunque siempre podemos generar más tokens (se pueden borrar si se pierden).

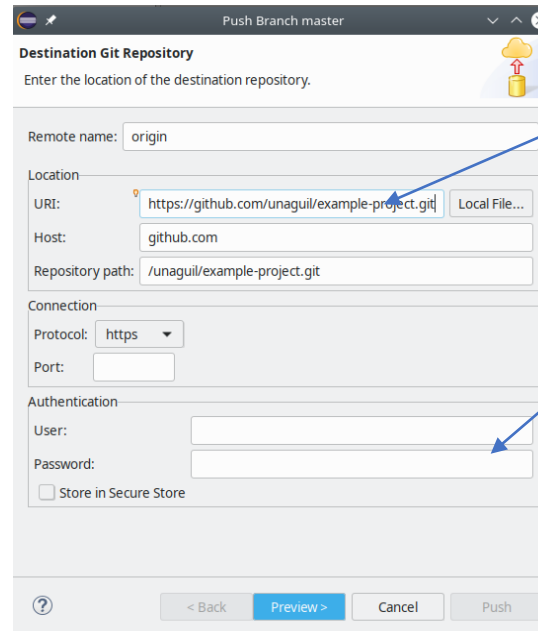
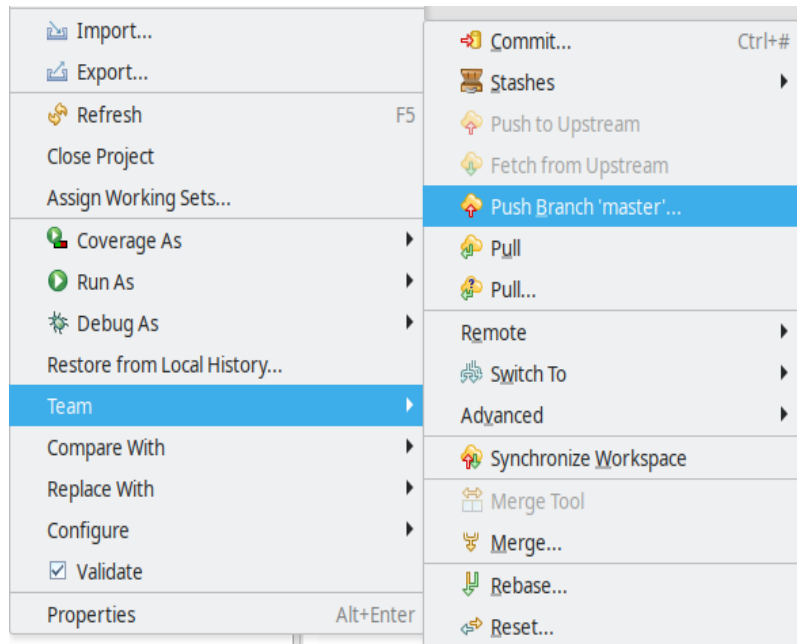
Make sure to copy your personal access token now. You won't be able to see it again!

✓ ghp_ZRHiBAairSkAxuf5qCMz1g4QHMP31X2YeNnB

Delete

Publicando por primera vez en el repositorio

- Aunque las versiones del proyecto están siendo controladas en local, la potencia de git es compartir y sincronizar estos cambios con un repositorio remoto.
- Supongamos que hemos creado un repositorio vacío en GitHub para nuestro proyecto:
<https://github.com/unaguil/example-project.git>
- Vamos a compartir el proyecto en este repositorio por primera vez usando la opción "Push Branch 'master'".



Copia la URI del repositorio de GitHub y Eclipse auto rellena los demás campos.

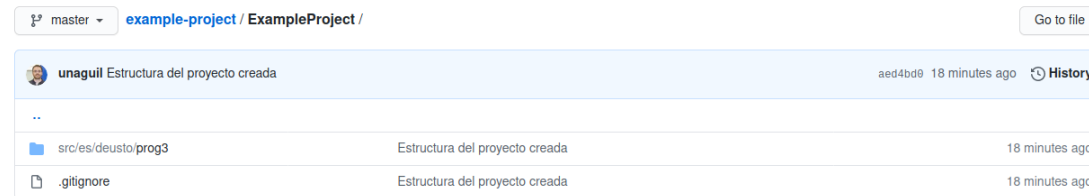
Debemos indicar también nuestro usuario y token que hemos generado en Github para poder publicar los cambios en el repositorio remoto (o descargar un repositorio privado).

Marca la opción para recordar el usuario y el token ya que podrás volver a verlo.

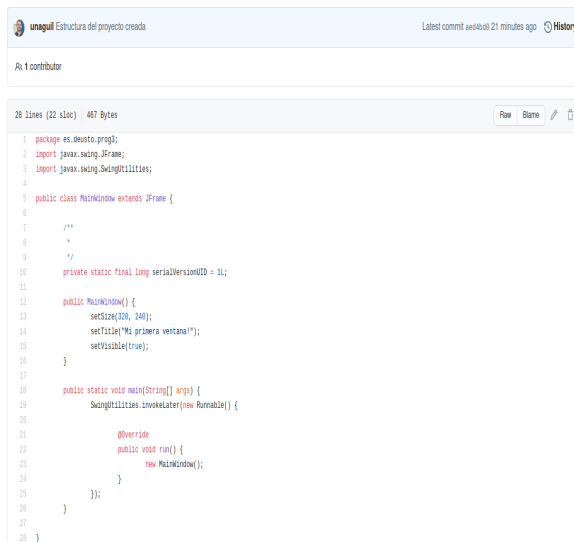
Si todo es correcto, un mensaje nos indicará que los cambios se han subido correctamente.

Inspeccionando el repositorio remoto

- Una vez subidos los cambios, el proyecto está publicado y, si tiene permisos otra persona puede inspeccionarlo, clonarlo e incluso contribuir.




- Podemos ver el historial de cambios de un fichero en GitHub navegando a través de su interfaz.



History for [example-project](#) / [ExampleProject](#) / [src](#) / [es](#) / [deusto](#) / [prog3](#) / [MainWindow.java](#)

Commits on Sep 29, 2020

Estructura del proyecto creada

 unaguil committed 22 minutes ago



aed4bd0

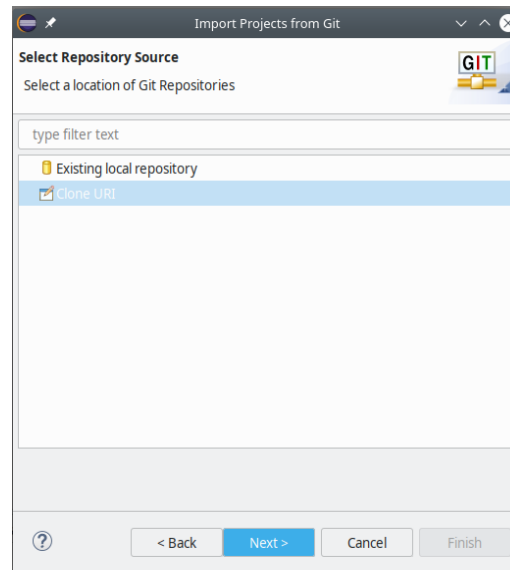
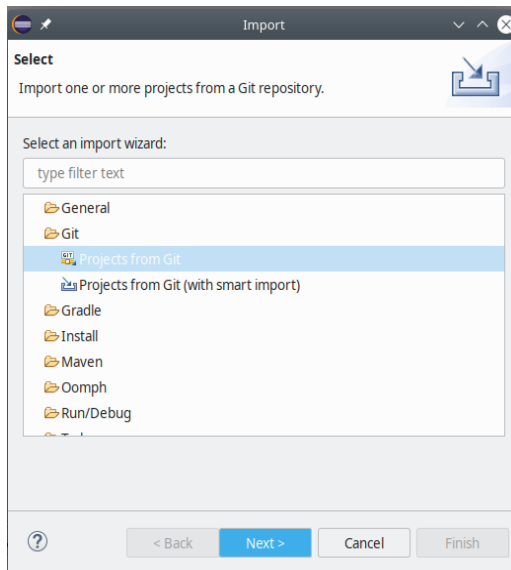


[Newer](#)

[Older](#)

Clonando un repositorio existente

- Una vez realizado el primer commit en el repositorio, otros desarrolladores del equipo pueden empezar a contribuir.
- Pero primer tienen que clonar el repositorio remoto para hacer una copia local del estado actual del proyecto.
 - File → Import → Projects from Git → Clone URI



Introducimos la URI del repositorio desde el que queremos clonar y el usuario) y contraseña si el repositorio no es público..

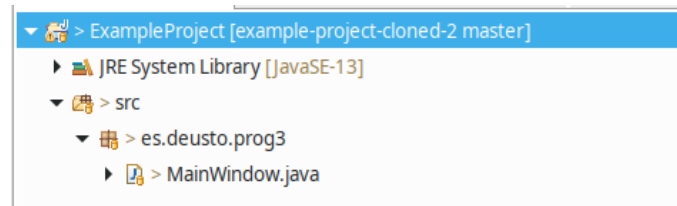
Si existe más de una rama (branch) podemos seleccionarla al clonar. Ahora solamente tenemos "master".

Debemos especificar en qué directorio se van a crear los ficheros de git → No debe ser un directorio que ya exista.

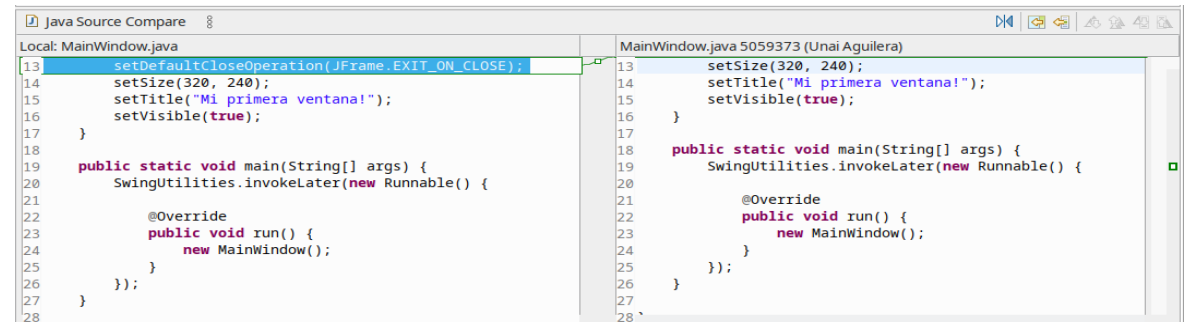
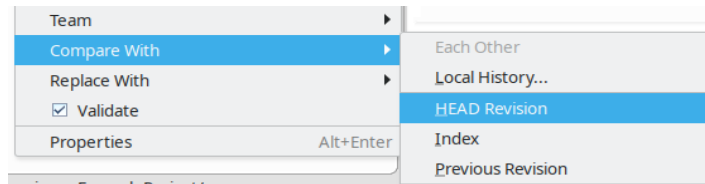
Seleccionamos la opción "Import existing Eclipse projects".

Funcionalidades del repositorio local (I)

- Cuando tenemos un repositorio local, ya sea porque lo hemos creado nosotros desde cero o hemos clonado uno remoto existente podemos:
 - Hacer cambios en uno o varios ficheros. Los cambios se marcan con el símbolo ‘>’ que indica que hay cambios más recientes que la última versión guardada.

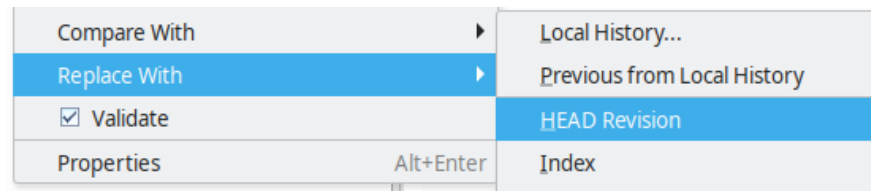


- Podemos comprobar las diferencias con la última versión guardada en un commit del fichero modificado.

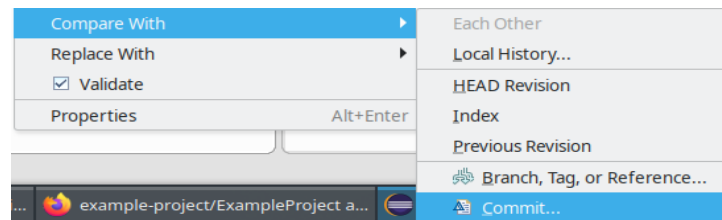


Funcionalidades del repositorio local (II)

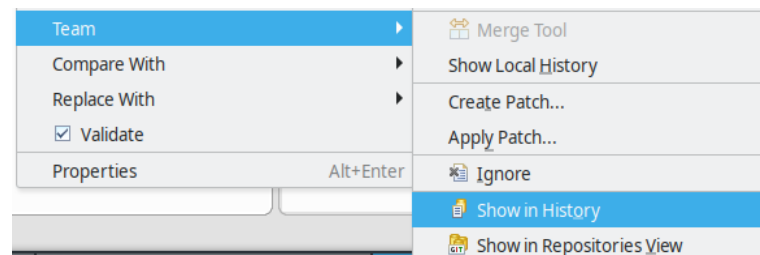
- Deshacer los cambios no guardados en un fichero y volver a la última versión guardada.



- Guardar los cambios en un nuevo commit (ya hemos visto cómo hacer esto).
- Comparar la versión actual con una versión anterior guardada.

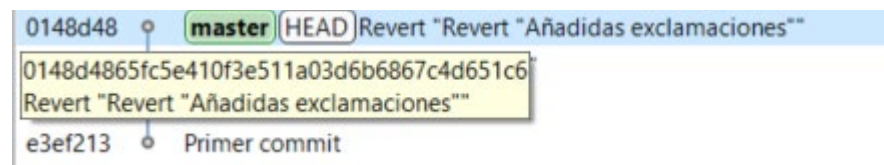
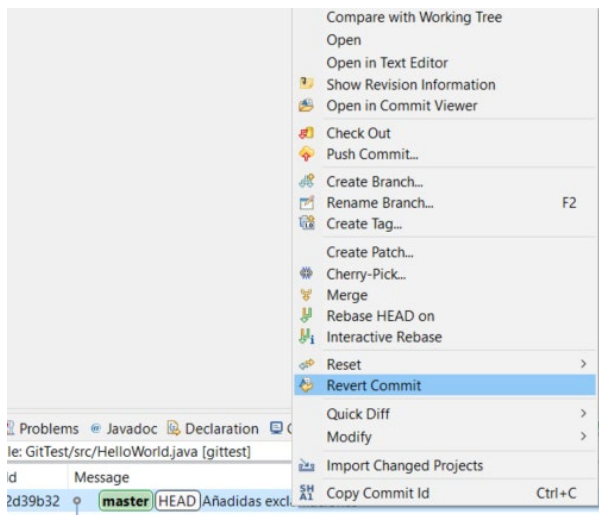


- Mostrar el historial de cambios guardados de un fichero.



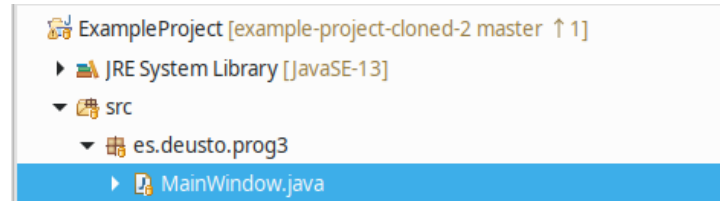
Funcionalidades del repositorio local

- Cuando estamos visualizando el historial podemos seleccionar un commit y deshacerlo (Botón derecho -> Revert commit).
 - Es importante entender que esto no borra el commit sino que añade un nuevo commit que realiza los cambios a la inversa. Esto permite recuperar lo deshecho si nos hemos equivocado.



Sincronizando el repositorio local

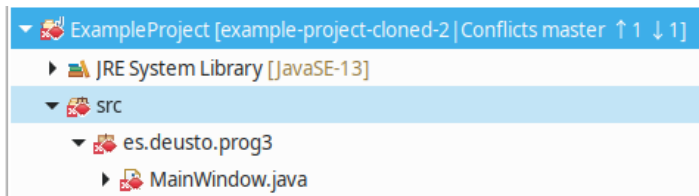
- Es importante recordar que todos los commits que hacemos se guardan solamente en local
 - Debemos publicar los nuevos commits en el servidor utilizando la opción "Push".
- El número de commits pendientes de subir aparece al lado del nombre del proyecto en Eclipse.



- Debemos usar la opción "Team → Push to origin" para publicar nuestros cambios.
- Por otro lado, los cambios subidos al repositorio por otros compañeros no se bajan automáticamente.
 - Debemos usar la opción "Team → Pull" para descargar los cambios nuevos a nuestra copia local y que se apliquen.
 - Si no hay conflictos los commits se aplican a nuestro repo local.

Conflictos en ficheros

- Git gestiona y mezcla automáticamente los cambios que se produzcan por varios desarrolladores.
 - Sin embargo, no sabe hacerlo si dos desarrolladores han modificado la misma zona de un fichero o si hay cambios que afectan al fichero completo (un usuario modifica un fichero y otro usuario lo borra).
 - En git no se pueden subir cambios si primero no se han bajado los más recientes del server.
 - Los conflictos no suceden hasta que se bajan nuevos cambios del server.
- Si ocurre un conflicto es el programador que ha bajado los cambios el que debe indicar cómo resolverlo.
 - Los conflictos se marcan el árbol del proyecto con un icono rojo.



Esto nos indica que hay un conflicto pendiente de resolver en el fichero MainWindow.java

Resolviendo conflictos

- Si hay uno o más ficheros en conflicto, ese desarrollador no puede seguir trabajando hasta que resuelva el problema.
- Supongamos que tenemos un conflicto debido a que:
 - En local hemos modificado una línea y hemos creado un commit con ese cambio.
 - Otro usuario ha borrado esa misma línea, ha hecho commit y ha subido sus cambios al servidor antes que nosotros.
- El problema es que git me obliga a bajar los cambios del server antes de poder subir los míos → Se produce un conflicto local al intentar aplicar los dos cambios.
- ¿Quién tiene razón con el cambio?
 - ¿Yo que he modificado la línea?
 - ¿El otro programador que la ha eliminado completamente?
- Git no sabe cuál es la solución correcta y nos pide ayuda.

Resolviendo conflictos

- Para resolver el conflicto debemos abrir el fichero y editarlo dejando el cambio que nos interese.

```
12     private static final long serialVersionUID = 1L;
13
14     public MainWindow() {
15 <<<<<< HEAD
16         setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
17 =====
18 >>>>>> branch 'master' of git@github.com:unaguil/example-project.git
19         setSize(320, 240);
20         setTitle("Mi primera ventana!");
21         setVisible(true);
22
23         JButton button = new JButton("Click me!");
24         JPanel panel = new JPanel();
25
26         panel.add(button);
27
```

- Git añade unas marcas en el texto para indicar dónde hay problemas, con nuestra versión "HEAD" y la otra versión descargada.
 - Debemos mirar esas líneas, borrar las marcas y dejar la versión que creamos que sea la que resuelve el conflicto.
 - Una vez resuelto el conflicto, crear un nuevo commit con la solución. Git añade un auto mensaje de solución del conflicto.

Consejos para trabajar con Git

- Haz commits frecuentemente
- Los commits deben ser unidades mínimas que añadan una funcionalidad o resuelvan un bug.
- No mezclar cambios no relacionados en un mismo commit.
- No hacer commit de código que no compila o está incompleto.
- Añadir comentarios significativos a cada commit para encontrar los cambios en el historial fácilmente.
- Publica tus commits en el repositorio cuando termines de trabajar o cuando quieras que otras personas puedan acceder a ellos.
- Antes de bajar los cambios de tus compañeros haz un commit con tus cambios.
- Si los cambios están publicados en el servidor siempre se pueden recuperar.
- Si se produce algún conflicto no te pongas nervioso, hay que aprender a resolverlos. Es parte del trabajo normal con git.

Recomendaciones para el proyecto

- En el proyecto de la asignatura se va a evaluar el uso que haces de Git para el desarrollo en equipo.
- La forma de demostrar que alguien ha contribuido al proyecto es a través de los *commits* de Git.
- Es importante que facilites la evaluación del trabajo que has realizado:
 - Comprueba que tu nombre asociado a cada *commit* incluye tu nombre y apellido y tu cuenta de @opendeusto. Configura Eclipse (o la herramienta que uses) adecuadamente.
 - Los mensajes de *commit* tienen que ser descriptivos.
 - Tus contribuciones tienen que estar asociadas a *commits*, así se puede conocer la aportación que has realizado al proyecto.
 - Si os surge algún conflicto de git, pensad detenidamente sobre el motivo y cómo se puede resolver. Si no sabéis resolverlo solicitad una tutoría para que lo solucionemos.
 - No borréis el repositorio una vez iniciado el proyecto, las fechas de los *commits* deben representar la historia de desarrollo hasta la finalización del proyecto (planificación).

Práctica 2C – Parte I

- Ejercicio en parejas. Cada desarrollador se identifica como A o B.
 - Se recomienda estar cerca para ver cómo se hace cada apartado.
- A: Crear un repositorio vacío en tu cuenta de GitHub y añadir la cuenta de su compañero con permisos de escritura.
- B: Crear un proyecto con una clase que imprima un “Hola”.
 - Realiza un *commit* y sube el proyecto a GitHub.
- A: Clonar en Eclipse el repositorio remoto.
 - Comprobar que el proyecto funciona correctamente.
- A y B: Revisar en la web de GitHub la estructura del proyecto subido.
 - Comprobar qué información existe, si es correcta, etc.
 - Recordad tener bien configurado vuestro nombre y correo para los commits.

Práctica 2C – Parte II

- A: Programa un método llamado `contar()` que imprima los números desde el 1 al 10 por pantalla.
 - Realiza un *commit* con un mensaje adecuado, pero no hagas push/pull.
- B: Mientras A está programando, añade una clase `Persona` al proyecto.
 - Debe tener un atributo `nombre` y únicamente métodos `get/set` para ese dato.
 - Realiza un *commit* con un mensaje adecuado, pero no hagas push/pull.
- Comprobad el estado del repo remoto en GitHub. No debería haber cambios.
- A: Realiza un push de tus *commits* pendientes.
- B: Intenta realizar un *push* de tus *commits* pendientes. No podrás, ya que hay cambios que no tienes. Primero haz un *pull*, comprueba los cambios y haz *push* de los cambios. No debería haber conflicto.
- A: Actualiza tu repositorio local con los últimos cambios del remoto.

Práctica 2C – Parte III

- B: Llama a la función *contar()* en el main para usarla.
- A: Modifica el mensaje de Hola para que salga Hola Mundo!!!!!!
- A y B: Subid vuestros cambios al repositorio y actualizad vuestros repositorios locales con los cambios.
 - Recordad que no puedes hacer push tus *commits* si hay cambios en el remoto que no tienes en local.
- Cuando tengas tu repositorio local actualizado prueba a inspeccionar el historial del proyecto (o de un fichero).
 - Ve a la opción Team->Show In History sobre un fichero seleccionado.
 - Podrás ver el historial de cambios sobre el fichero y otra información.

Práctica 2C – Parte III

- B: Revierte el commit realizado por A donde se añadían las exclamaciones al mensaje.
 - Comprueba que se ha creado un commit en tu repositorio local.
 - Sube este nuevo cambio al repositorio remoto.
- A: Descarga todos los cambios del repositorio remoto.
- A y B: Inspeccionar los cambios en local, comparando los cambios que se han realizado en cada commit.
 - También podéis observar el historial de cambios en GitHub.
 - Comprobad que tenéis en local todos los commits que existen en GitHub.

Práctica 2C – Parte IV

- Hasta ahora hemos trabajado en partes distintas del código. Ahora vamos a producir un conflicto, detectarlo y ver cómo se resuelve.
- A: Renombra el método *contar()* a *imprimir()*. Prueba el programa
 - Haz commit, pero no hagas *push*.
- B: Borra el método *contar()* y la llamada al mismo. Prueba el programa.
 - Haz commit, pero no hagas *push*.
- A y B: Subid vuestros cambios al repositorio.
- ¿Ha ocurrido algún conflicto?. Intentad resolverlo con los pasos vistos en clase.
 - Si tienes alguna duda, levanta la mano y pregunta cómo resolverlo.