Control de versiones usando Eclipse con GIT y XP-DEV

Introducción

En esta guía vamos a mostrar cómo llevar el control de versiones de vuestro proyecto desarrollado en Eclipse utilizando Git, un software de control de versiones. Para ello se requiere que Eclipse tenga instalado el plugin de Git.

Además, utilizaremos XP-DEV como repositorio remoto de vuestro proyecto. Así los profesores podremos tener acceso a vuestro código fuente, ya que os hemos pedido que nos incluyáis como miembros de vuestros equipos.

Pasos a seguir

0. Instalación del plugin Git en Eclipse

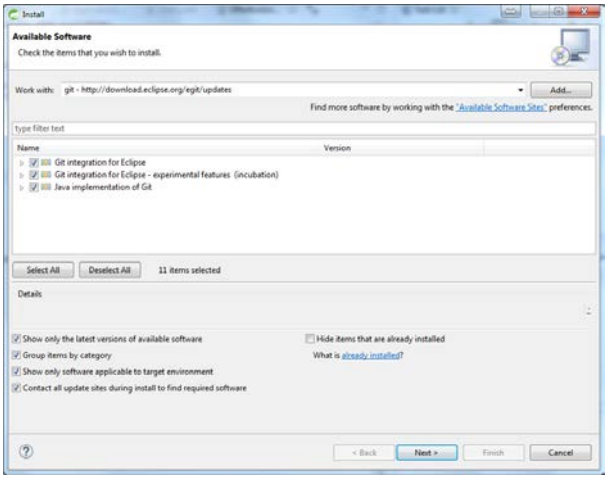
Si no está instalado el Git en Eclipse (si al hacer File => New => Other => no aparece "Git") deberéis instalarlo:

1) Usando Eclipse Marketplace

Help => Eclipse Marketplace => Find "Egit" => Go => "Egit – Git Integration for Eclipse" => Install

2) Buscando e instalando los plugins desde repositorios

Help => Install New Software => Add => http://download.eclipse.org/egit/updates

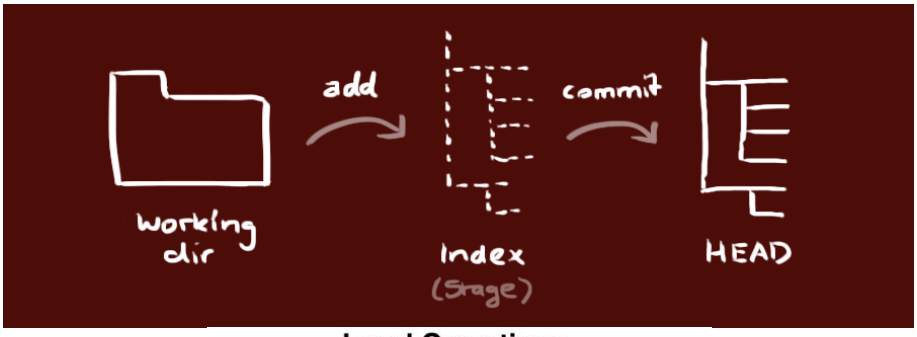


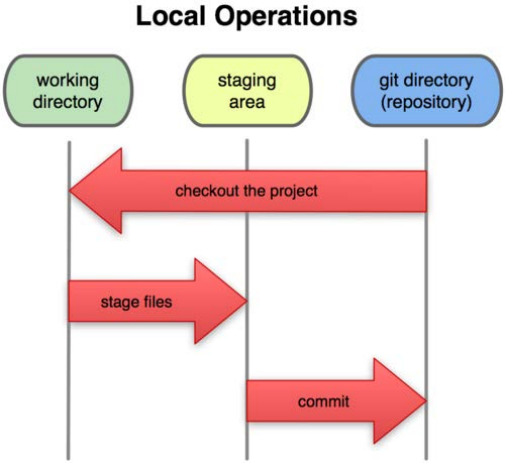
Nota: si da problemas de incompatibilidades encontrad la URL donde se encuentre la versión apropiada para la instalación de Eclipse.

1. Descripción de la estructura de Git

Cuando usamos el software de control de versiones Git, los ficheros del proyecto pueden encontrarse en tres estados: 1) Confirmado (committed) que significa que los datos están almacenados de manera segura en la base de datos local de Git; 2) Modificado (modified) cuando contiene algún cambio pero que todavía no se ha confirmado y 3) Preparado (staged) que significa que se ha marcado un fichero modificado para que se confirme en el siguiente commit.
Esto nos lleva a las tres secciones principales de un proyecto de Git: el directorio Git (Git directory o HEAD), el directorio de trabajo (working directory), y el área de preparación (staging area o INDEX).

Las siguientes figuras sacadas de http://rogerdudler.github.io/git-guide/index.es.html y de http://librosweb.es/libro/pro_git/capitulo_1/fundamentos_de_git.html nos muestran cómo cambiar el estado de los ficheros de un proyecto. A destacar en este momento que con la operación "commit" se confirman los cambios, y con la operación "checkout" se puede reestablecer el directorio de trabajo desde un directorio git.





Además, los repositorios Git pueden ser locales (que sirven para gestionar los cambios de un único usuario) y remotos (que permiten compartir versiones confirmadas entre diferentes miembros del grupo de desarrollo)

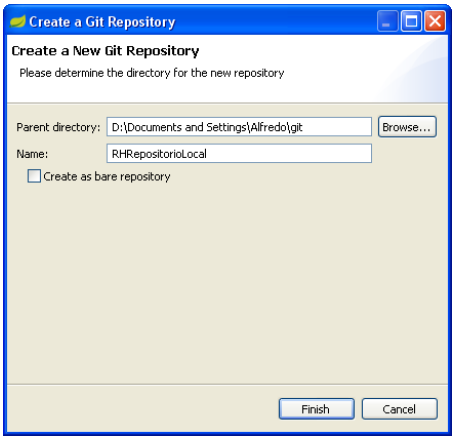
La idea básica de trabajo con Git es la siguiente: cada usuario trabaja en el proyecto en un directorio de trabajo al que le va añadiendo cambios, y cuando considera que tiene una versión estable entonces hace un commit al repositorio local. Después puede hacer un "push" del directorio Git local a un directorio Git remoto en un servidor para que otros desarrolladores puedan acceder a esos cambios.

En cualquier caso, hay que tener cuidado con los “merge” entre las distintas versiones porque no todo se puede resolver de manera automática. Habrá que resolver de manera manual cuando dos desarrolladores hayan hecho cambios exactamente en el mismo recurso.

2. Creación de un repositorio local Git e incluir un proyecto el mismo

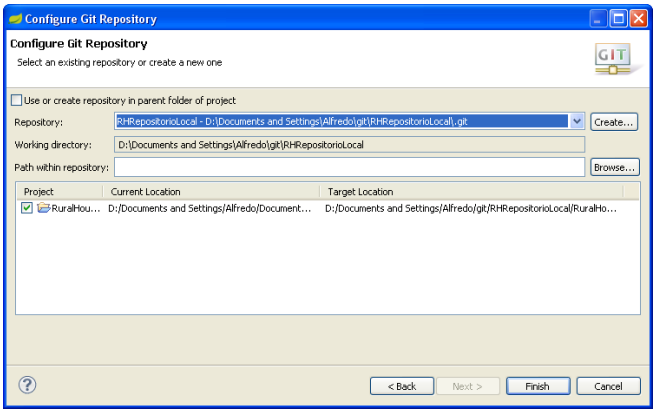
Para crear un repositorio local:

Window => Open Perspective => Other => Git => Create a new Local Git Repository => No activar “Create as bare repository” porque en principio se trata de un “working repository” (se trata de un repositorio de trabajo y no de un repositorio centralizado donde se quiere compartir los cambios)



Para incluir el proyecto en el repositorio local:

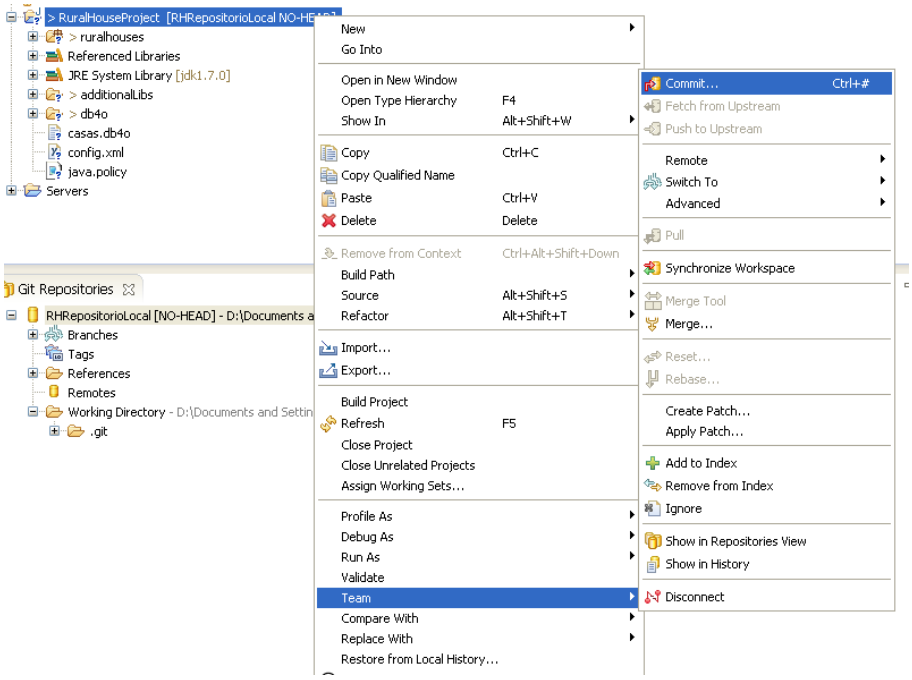
Hacer click derecho en el proyecto (por ejemplo Bets) en el explorador de paquetes => Team => Share Project => Team => Git => Next => Escoger el repositorio local Git en la lista desplegable etiquetada con “Repository:” => Finish



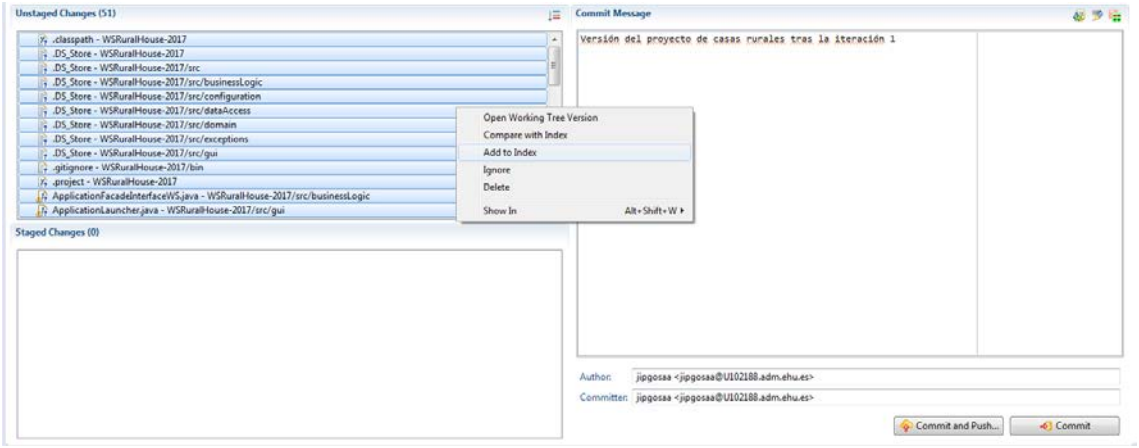
Una vez creado se verá que se ha modificado el proyecto en el explorador, y se puede ver que se encuentra en el repositorio local, pero que todavía no se ha hecho un “commit” del mismo. Lo que haremos ahora será hacer el “Commit” de dicho repositorio local, para decir que esta es nuestra primera versión del proyecto que vamos a controlar por medio del sistema de control de versiones.

Para hacer el commit en el repositorio local:

Click derecho en el proyecto => Team => Commit

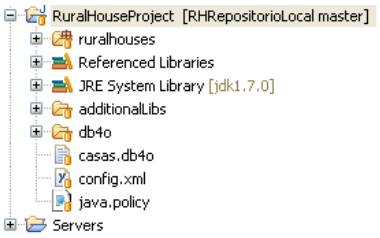


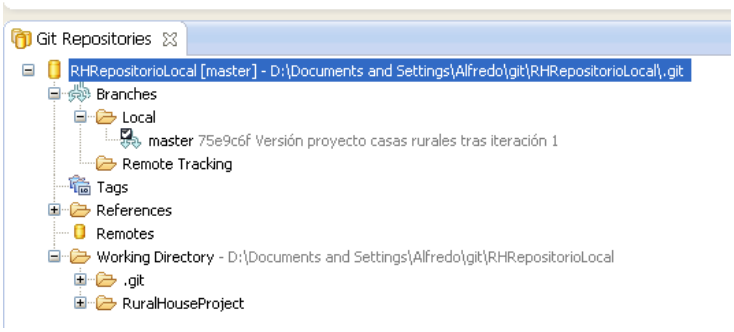
Poner el mensaje de confirmación (commit) que diga qué versión es (por ejemplo "Versión del tras la iteración 1") => Preparar (añadir al stage o al index) todos los ficheros, ya que queremos confirmar todos esta primera vez en el directorio local Git. Para ello seleccionar todos los ficheros en el panel de "unstaged changes" => Add to Index



=> Commit (Para confirmarlos en nuestro repositorio local Git. Para subirlos a un repositorio remoto Git sería "Commit and Push", pero todavía no estamos en ese punto)

Después de ello, ya tendremos creada la copia máster local, en nuestro directorio de trabajo (el Working Directory) y una sola rama (la rama “master” en branches, con información de qué versión es).





3. Realización de algún cambio para confirmar o deshacer

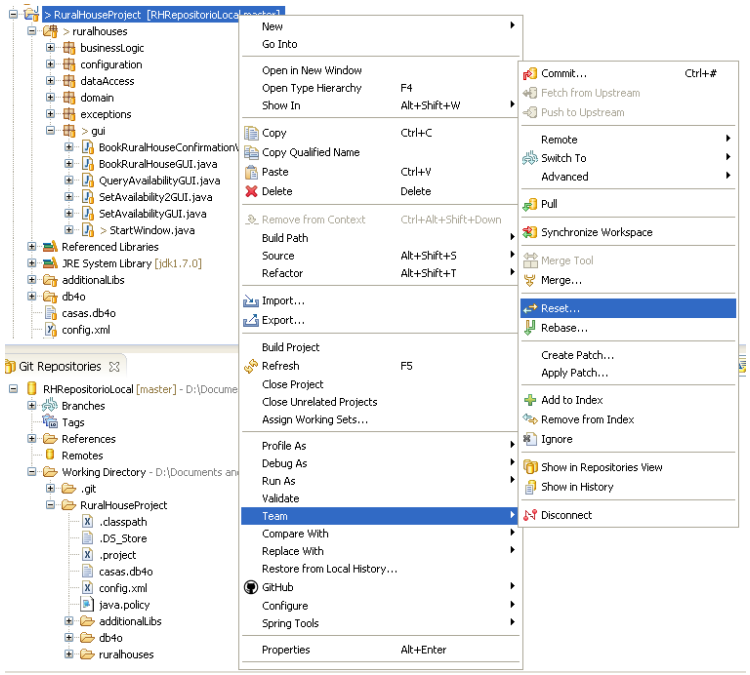
Después de unos cuantos cambios en el proyecto podríamos encontrarnos ante dos opciones: que tenemos una versión inestable y queremos volver a una versión anterior, o bien, que ya tenemos una nueva versión que queremos “confirmar” y dar un nombre a la versión actual.

En este punto vamos a probar ambas cosas, pero no después de hacer muchos cambios sino después de hacer uno simple.

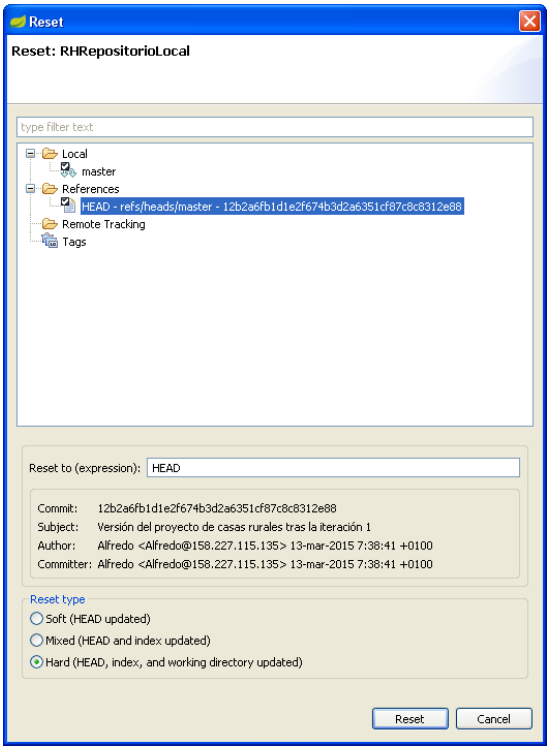
Realizar modificaciones y deshacer:

La modificación simple a realizar será: cambiar el texto la etiqueta `lblNewLabel` del JFrame principal MainGUI.java por “Seleccionar opción:” y guardar los cambios en el proyecto:File => Save all

Si después de realizar ese cambio, quisiéramos volver a la versión estable anterior podríamos hacerlo así: Click derecho en proyecto => Team => Reset



Seleccionar la versión deseada (hay que fijarse que la correcta, mirando en el mensaje con el que se hizo el commit; en este caso es la HEAD que tiene el mensaje “Versión del proyecto tras la iteración 1”) y hacer un “HARD” reset, esto es, que actualice el HEAD, el index y el working directory o directorio de trabajo.

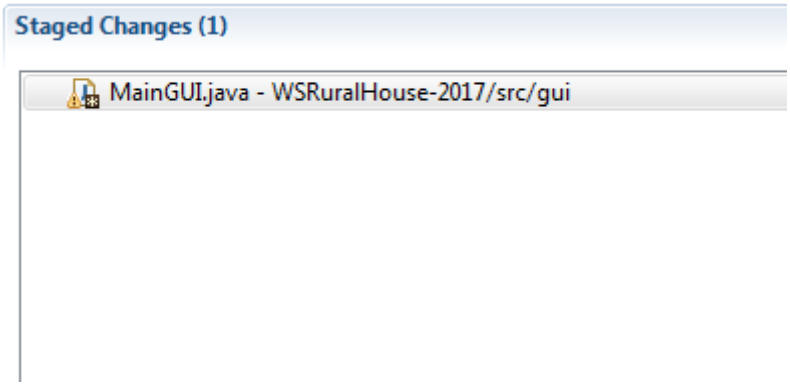


Podréis comprobar que otra vez tenemos en el MainGUI.java la etiqueta `lblNewLabel` con el texto ResourceBundle.*getBundle*("Etiquetas").getString("SelectOption").

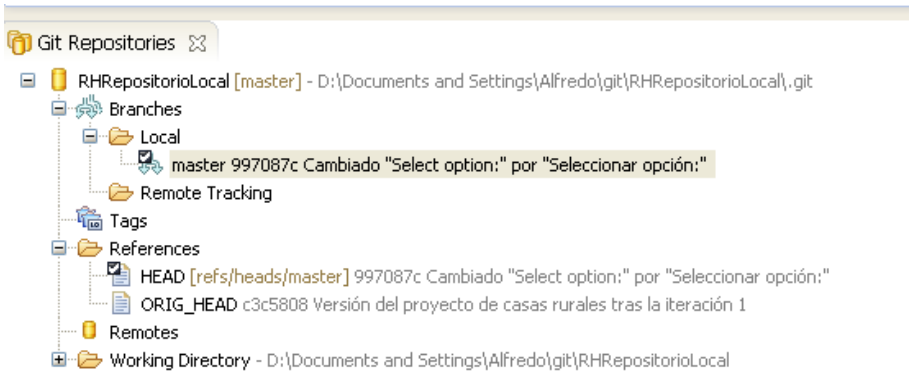
Realizar cambios y confirmar una nueva versión:

La modificación simple a realizar será de nuevo: cambiar la etiqueta `lblNewLabel` del del JFrame principal MainGUI.java por “Seleccionar opción:” y guardar los cambios en el proyecto:File => Save all

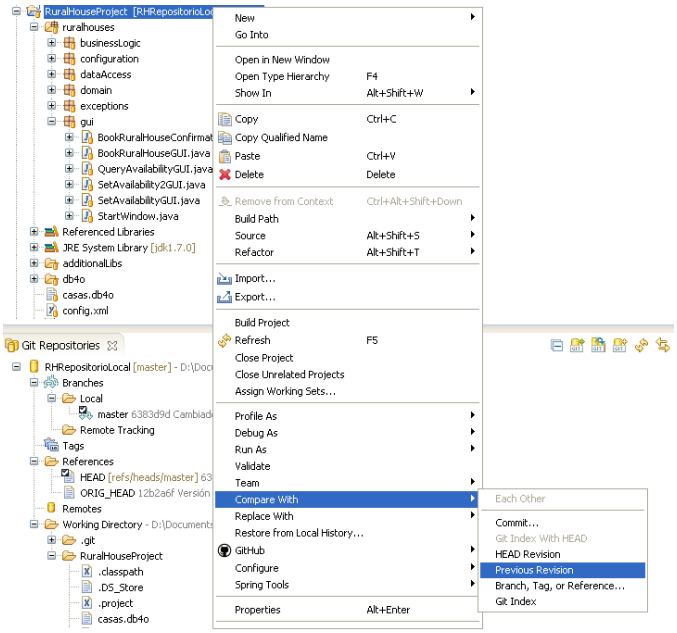
Después de hacer ese cambio no tendríamos por qué confirmar los cambios puesto que estos permanecen en nuestro “working directory” local, pero vamos a suponer que esto supone una nueva versión “estable” que queremos confirmar y poner por tanto en nuestro repositorio Git local. Para ello: Click derecho en proyecto => Team => Commit => Poner un mensaje de commit (Por ejemplo: ‘Cambiado “Select option:” por “Seleccionar opción:”’) y veremos que ya nos añade como preparadas (staged) las clases que se han modificado, para ser confirmadas en el commit que se va a hacer. En este caso solamente MainGUI.java



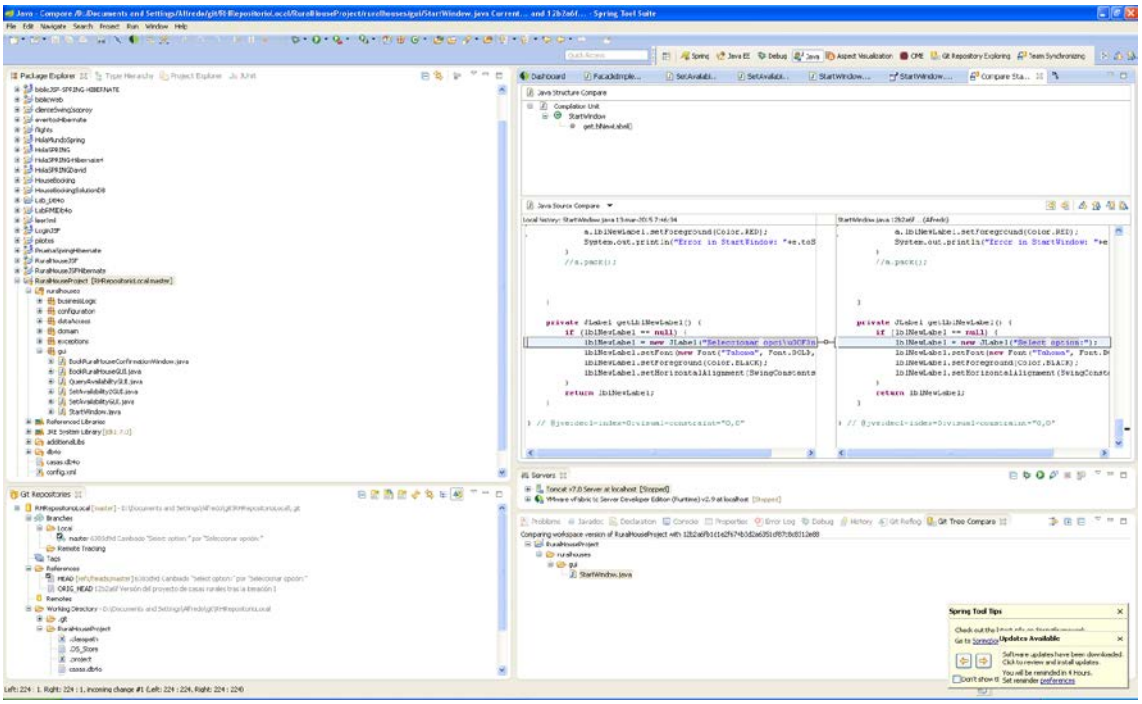
En la información de los repositorios Git podemos ver que la rama master ha cambiado, pero que en realidad tenemos la versión anterior. Y si comprobarais la clase MainGUI.java del working directory veríais que es la que contiene la etiqueta en castellano.



Una característica interesante es que se puede comparar una versión con otras anteriores si se hace click derecho en el proyecto => Compare with => Selecciona la rama, o versión commit, head, etc. En este caso sería la revisión anterior:



En la vista Git Tree Compare (si no aparece se puede mostrar con: Window => Show View => Other => Git => Git Tree Compare) se puede seleccionar la clase MainGUI.java y después de mostrarán las diferencias entre una versión y la otra y se podrá ir navegando por las mismas con estos iconos: 



4. Utilización de repositorios remotos

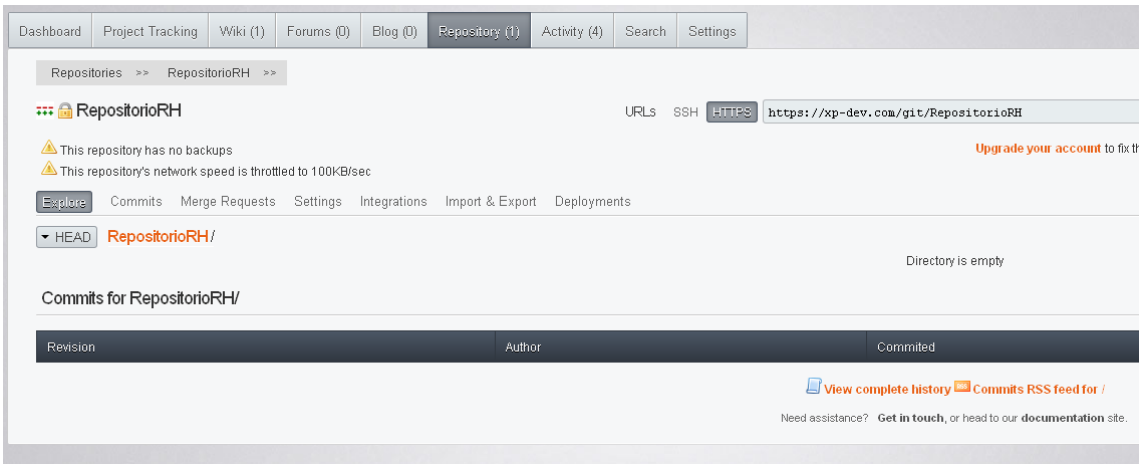
Hasta ahora hemos realizado el control de versiones en una copia Git local, que puede ser interesante aunque solamente haya un desarrollador. Sin embargo, para que se pueda compartir el proyecto entre varios desarrolladores, es necesario que se suba el repositorio Git a un repositorio Git compartido.

Uno de los repositorios más utilizados es GitHub https://github.com/ pero en nuestro caso vamos a utilizar XP-DEV para llevar de una manera integrada la gestión del proyecto (iteraciones, historias de usuario y bugs con sus tareas asociadas de nuestro proyecto SCRUM) con el repositorio software.

5. Creación de un repositorio remoto en XP-DEV

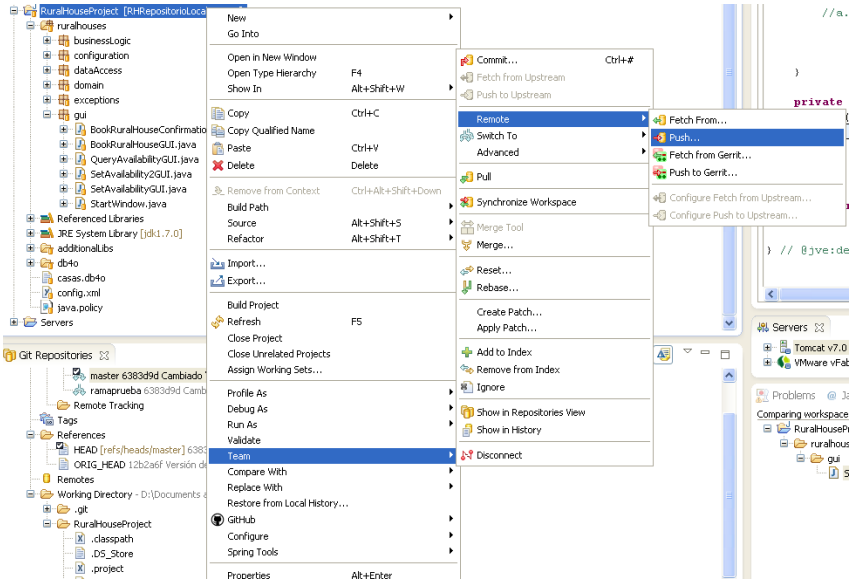
Entrar con la cuenta del SCRUM Máster de vuestro grupo en XP-DEV => Seleccionar el proyecto => Repository => Create a new repository => Dar un nombre al repositorio y seleccionar que sea de tipo “Git” => Save. El nombre tendrá que ser único.

En la siguiente ventana podríamos ver la URL donde estará nuestro repositorio remoto compartido por todos los miembros del grupo: en este caso: https://xp-dev.com/git/RepositorioRH

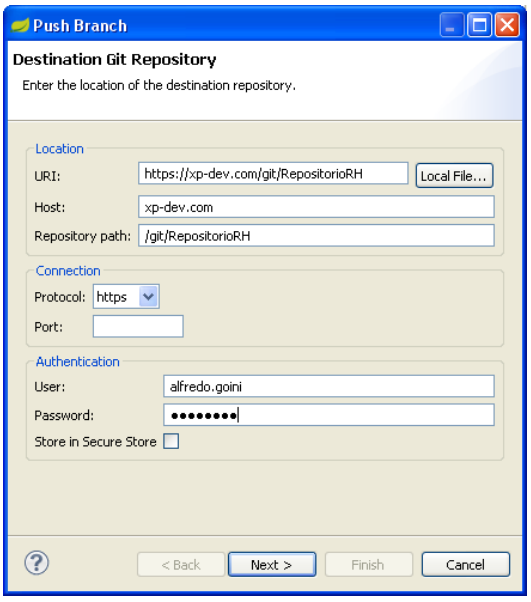


6. Subir rama de repositorio local Git a repositorio remoto en XP-DEV

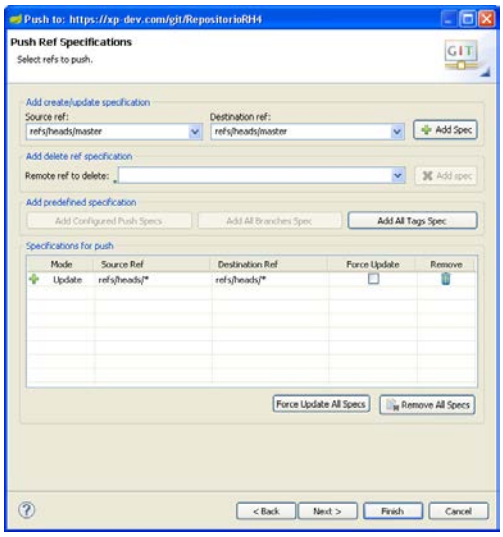
Seleccionar el proyecto (o bien la rama que queramos subir) haciendo Click Derecho => Team => Remote => Push



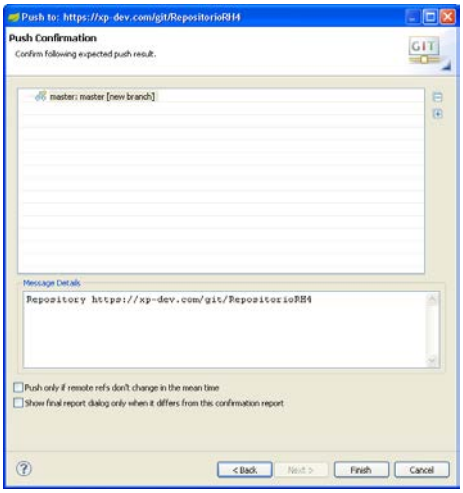
Y después indicar la URI del repositorio Git remoto creado en XP-DEV (https://xp-dev.com/git/RepositorioRH en este caso) y la cuenta y password XP-DEV.



Escoger después la rama fuente y la rama destino => Pulsar botón “Add All Branches Spec” => Finish

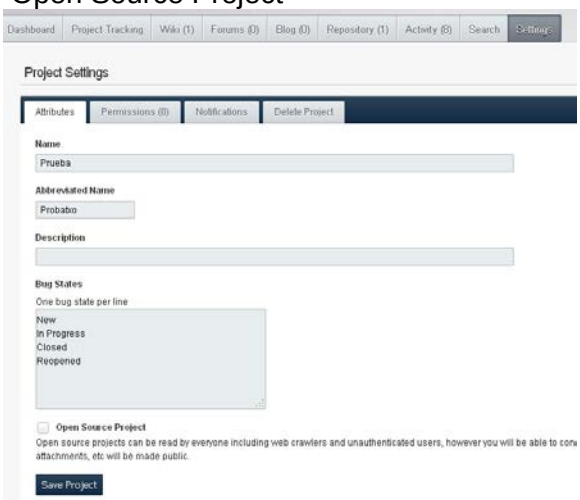


Y veremos que se confirma la subida de la rama:



En el repositorio alojado en XP-DEV deberíamos también ver que se encuentra el código del proyecto.

NOTA: si al subir un proyecto al repositorio XP-DEV da un error de autorización: (401 Unauthorized), puede ser que se definió el proyecto como público y no privado (esto es, se seleccionó la opción “Public Project”). Para cambiarlo: Project Settings => No seleccionar la opción “Open Source Project”



7. Trabajo en equipo con el repositorio del proyecto

Una vez que el SCRUM Máster del grupo haya subido el proyecto al repositorio del XP-DEV, el resto de miembros podrá trabajar con el mismo, esto es, realizar nuevos commits, obtener nuevas versiones del proyecto realizadas por otros miembros del equipo (e incluso aspectos más avanzados como trabajar con ramas para más adelante hacer merges, que mencionaremos al final del laboratorio). Para ello es necesario que:

- Todos los miembros tengan permisos de administrador (tal y como se explicó en el laboratorio de XP-DEV).

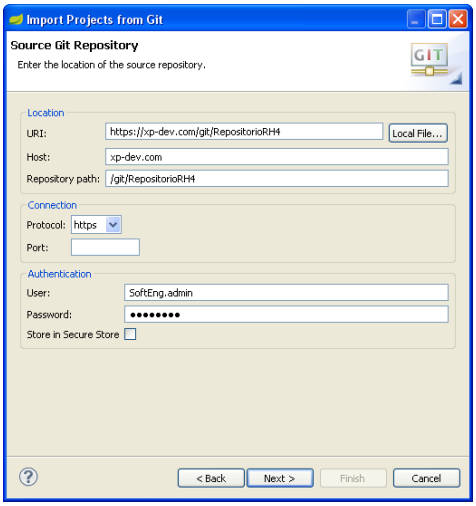
- Que XP-DEV reconozca que todos los usuarios están trabajando con el MISMO proyecto subido inicialmente por el SCRUM Máster. Para ello, os recomendamos que el resto de usuarios importe previamente el proyecto subido al repositorio remoto de XP-DEV en su directorio GIT local (tal y como se explica en la siguiente sección).

  Nota: si no habéis hecho esto, entonces, la nueva versión deberá subirla de nuevo desde su repositorio GIT local quien hizo el primer commit, y después, tras importar todos esa nueva versión, ya se podrá trabajar de manera adecuada.

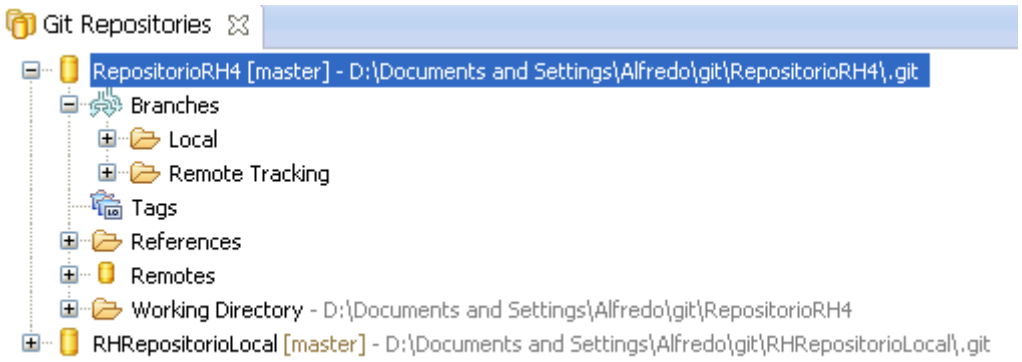
8. Importar/Clonar proyectos de repositorios remotos

Para importar un proyecto de un repositorio remoto en un repositorio GIT local y en el working directory (workspace de Eclipse) hay que hacer:

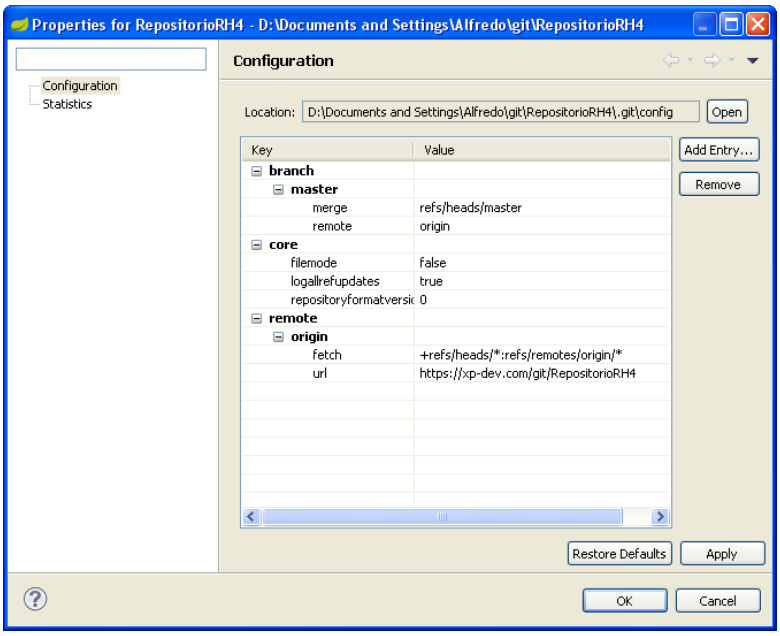
File => Import => Git => Projects from Git => Next => Clone URI => Next => Dar los datos del URL de vuestro repositorio, cuenta y password XP-DEV => Next =>



De esta manera se verá en la ventana de repositorios Git.

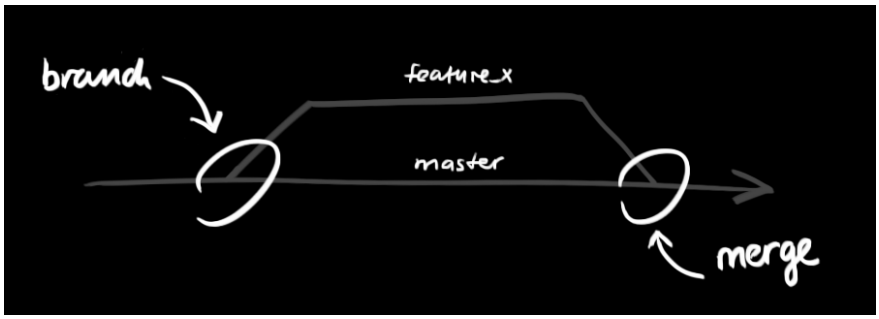


Y se podrán las propiedades de los repositorios haciendo click derecho en uno de ellos => Properties

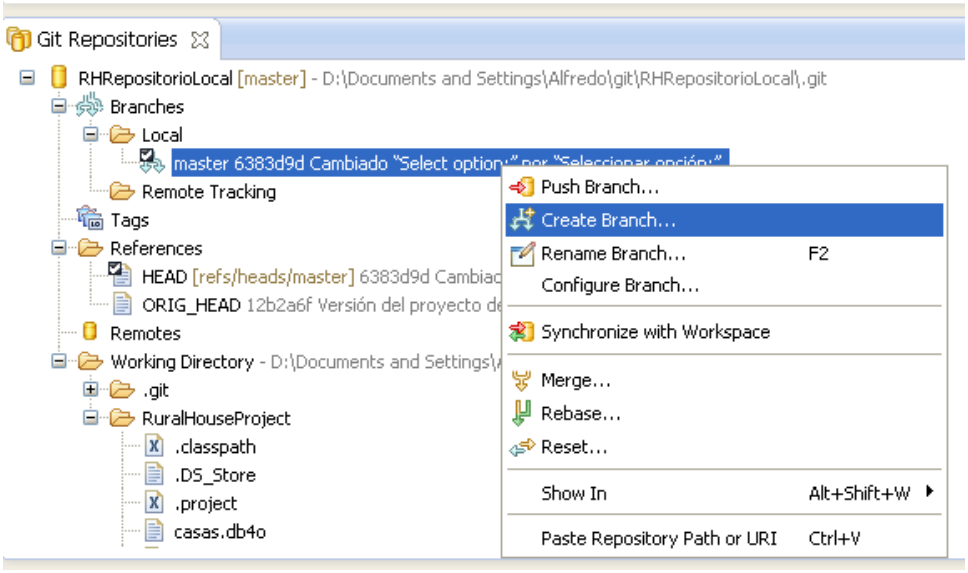


9. Creación de ramas (branches) en el proyecto

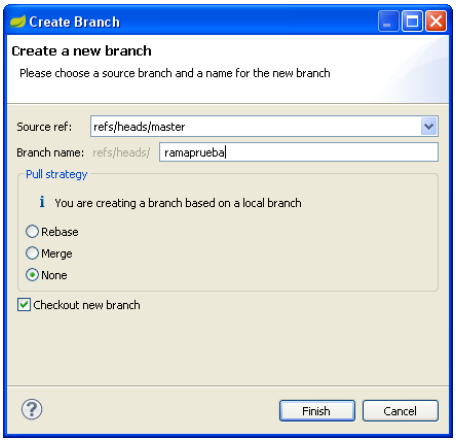
Hasta ahora hemos usado una única rama, que era la rama por defecto llamada master. Podemos crear nuevas ramas si queremos añadir o probar nuevas funcionalidades aisladas, de tal manera que más adelante, podamos darlas por buenas con commit, o si no, volver a versiones “estables” como podría ser la rama master, o también fusionar (merge) la nueva rama con la master anterior. Esto es claramente bueno, porque a veces nos liamos a probar cosas nuevas, que al final no funcionan y es una tarea desagradable y propensa a fallos el volver a la versión anterior. En general esta técnica sirve para que se puedan mantener ramas diferentes del software: una que sea la que está en producción, otra en desarrollo, otra con el software en fase de pruebas, etc.



Por ejemplo, podríamos querer crear una rama nueva a partir de la rama master anterior, haciendo click derecho en la rama (en la ventana de “Git Repositories”) => Create Branch



Le indicamos que queremos crearla a partir de la master (en Source ref:) y le damos un nombre a la rama: ramaprueba. => Finish. Si se activa la opción “checkout new branch” hará que el “working directory” contenga esta nueva rama.

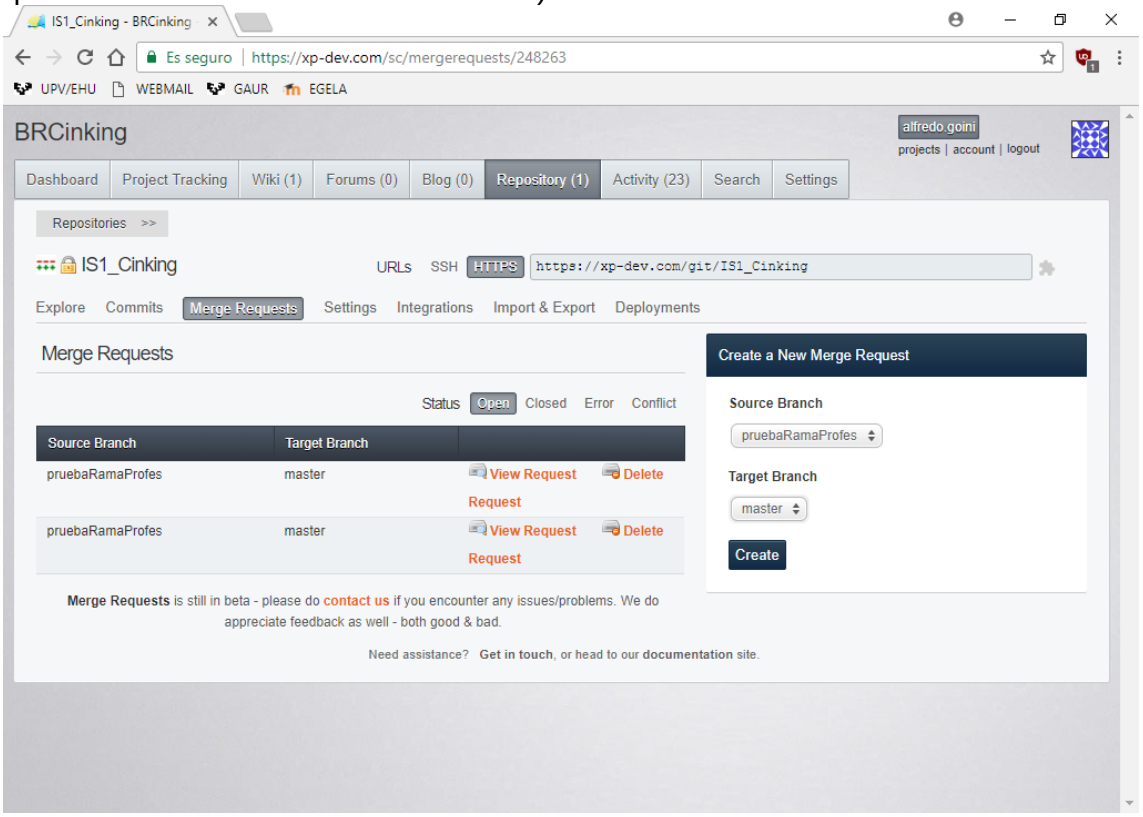


Las posibilidades son muchas a partir de este momento, pero hay que tener cuidado y seguramente tendréis que consultar la documentación si os encontráis con problemas: http://wiki.eclipse.org/Es:EGit/Es:User_Guide

10. Realización del MERGE de ramas en XP-DEV

XP-DEV permite realizar la fusión (MERGE) de distintas ramas subidas al directorio remoto. Para ello, una vez subida la rama, tal y como se explica en el apartado “Subir rama de repositorio local Git a repositorio remoto en XP-DEV” se puede pedir que XP-DEV haga un merge de las ramas de esta manera:

Situados en Repository en XP-DEV => Merge Requet => Seleccionar “Source Branch” y “Target Branch” (en el caso mostrado en la siguiente figura fusionaríamos la rama “pruebaRamaProfes” con la rama máster) => Create => Añadir Comentario => Save



Nota: XP-DEV dice que esta funcionalidad está en fase BETA, por lo que si no funciona, podría convenir que hicierais localmente los merge en un directorio GIT local y luego subirlo como rama máster.