Tutorial SVN en Wollok

por Juan Contardo,
Fernando Dodino
Versión 1.0
Junio 2016

Índice

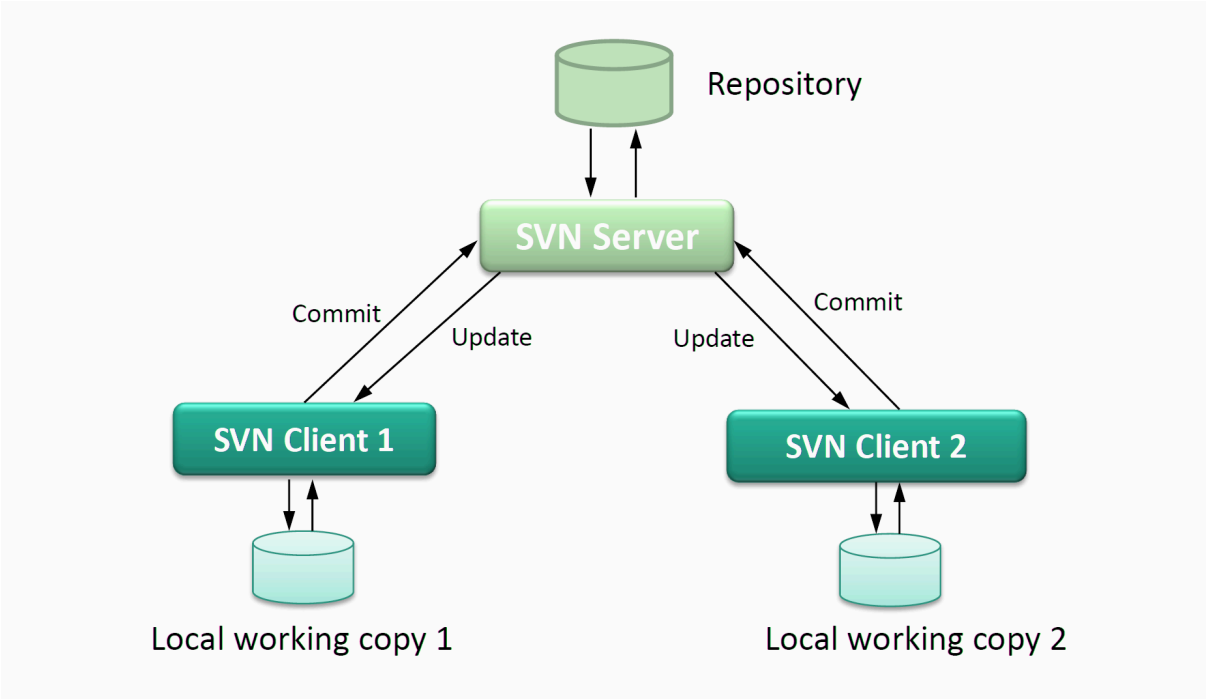
1 ¿Qué es SVN?

SVN, o mejor dicho Subversion, es un sistema de control de versiones. Permite la integración y el trabajo de un grupo de personas distribuido. Nos va permitir tener diferentes versiones

- una “estable” en el repositorio SVN
- y cada uno tendrá una copia local con cambios pendientes para subir

Cuando considero que mis cambios realizados sobre la versión que estoy trabajando ya están funcionando correctamente[1], entonces agrego los mismos a la versión del servidor, de forma que tengo siempre al menos una versión andando y compartida.

Esto nos facilita la forma de trabajo y nos permite que múltiples personas interactúen sobre los mismos archivos, por supuesto con cierto cuidado.



En el servidor pueden guardarse versiones

- del proyecto[2]
- o bien de los archivos que lo componen. Esto permite comparar el historial de modificaciones que tuvo un archivo a lo largo del ciclo de vida del proyecto (TP)

El usuario trabaja sobre su working copy (su copia local del proyecto), generando cambios sobre archivos, creando nuevos o borrando los mismos. Para propagar esos cambios a todo

[1] Y lo sé si soy responsable de definir y automatizar mis casos de prueba
[2] Como un conjunto de archivos que constituyen una versión estable en algún punto del tiempo

el equipo se hace un commit, que es impactar los cambios de los archivos en el servidor generando una nueva versión (*revision*).

Cuándo hacer commit (o *commitear* en spanglish) va a depender de cómo trabaja cada grupo. Se considera una buena práctica no commitear cosas que rompan todo el proyecto, pero tampoco es correcto el otro extremo de commitear cada 6 meses ya que es más que probable que se encuentren modificados los mismos archivos en los que estamos trabajando por otros usuarios.

En caso de que dos usuarios distintos modifiquen el mismo archivo, el svn tratará de hacer un merge por sí solo (integrar o *mergear* los archivos). Pero si no puede porque tocaron las mismas líneas o parecidas va a generar un conflicto que se debe resolver a mano.

Vamos a ir viendo cada uno de estos conceptos paso a paso a lo largo del tutorial :)

Existen múltiples clientes para manejarse con SVN. En el caso de Wollok ya tiene integrado un cliente llamado subversive al IDE, sobre el cual se encuentra basado este tutorial.

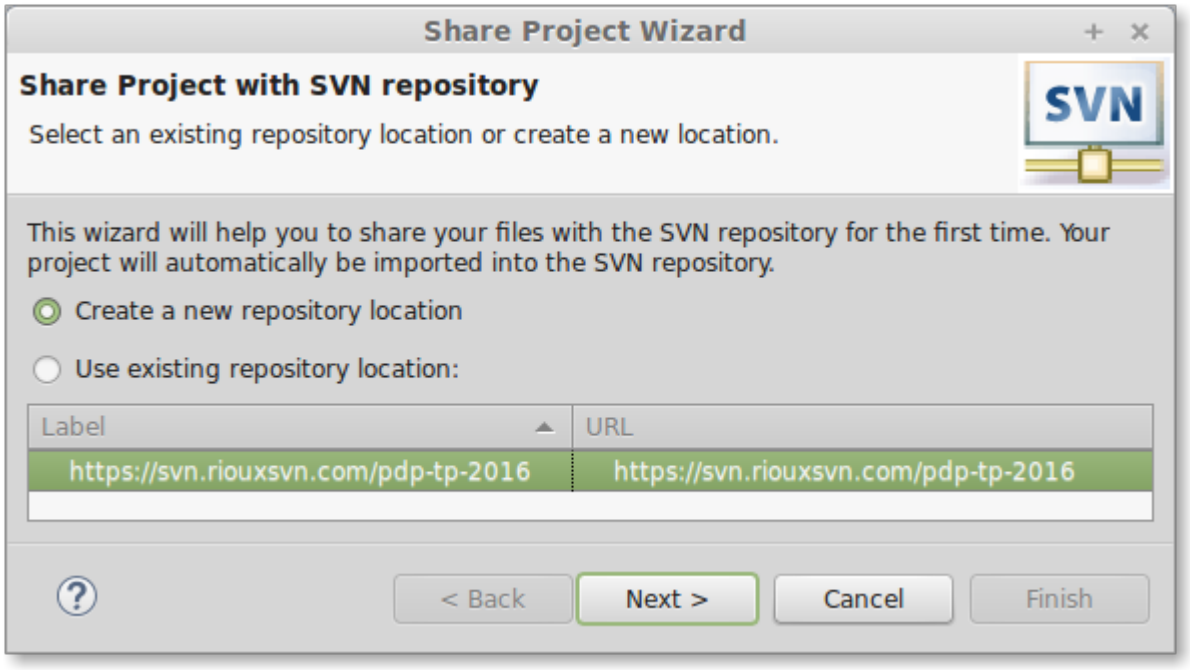
2 Compartir nuestro proyecto

Basta con ponerse de acuerdo, y uno de los integrantes del proyecto crea el proyecto en su entorno Wollok y lo sube al repositorio por única vez[3].

Debe crear un proyecto Wollok en el eclipse donde defino las primeras líneas del proyecto. Ya estoy por subir el proyecto al repositorio. ¡Qué rápido! ¿no? Para esto, click derecho sobre el proyecto, **Team -> Share Project**, y ahí tienen dos opciones:

- Si es la primera vez que intentan compartir algo, pasen a la siguiente viñeta. Ok, si estás leyendo acá es porque ya venís trabajando con varios repositorios, por lo tanto la primera pantalla que nos va a aparecer es una selección de repositorio como ésta:

[3] Luego, todos los demás lo deben bajar en sus máquinas.

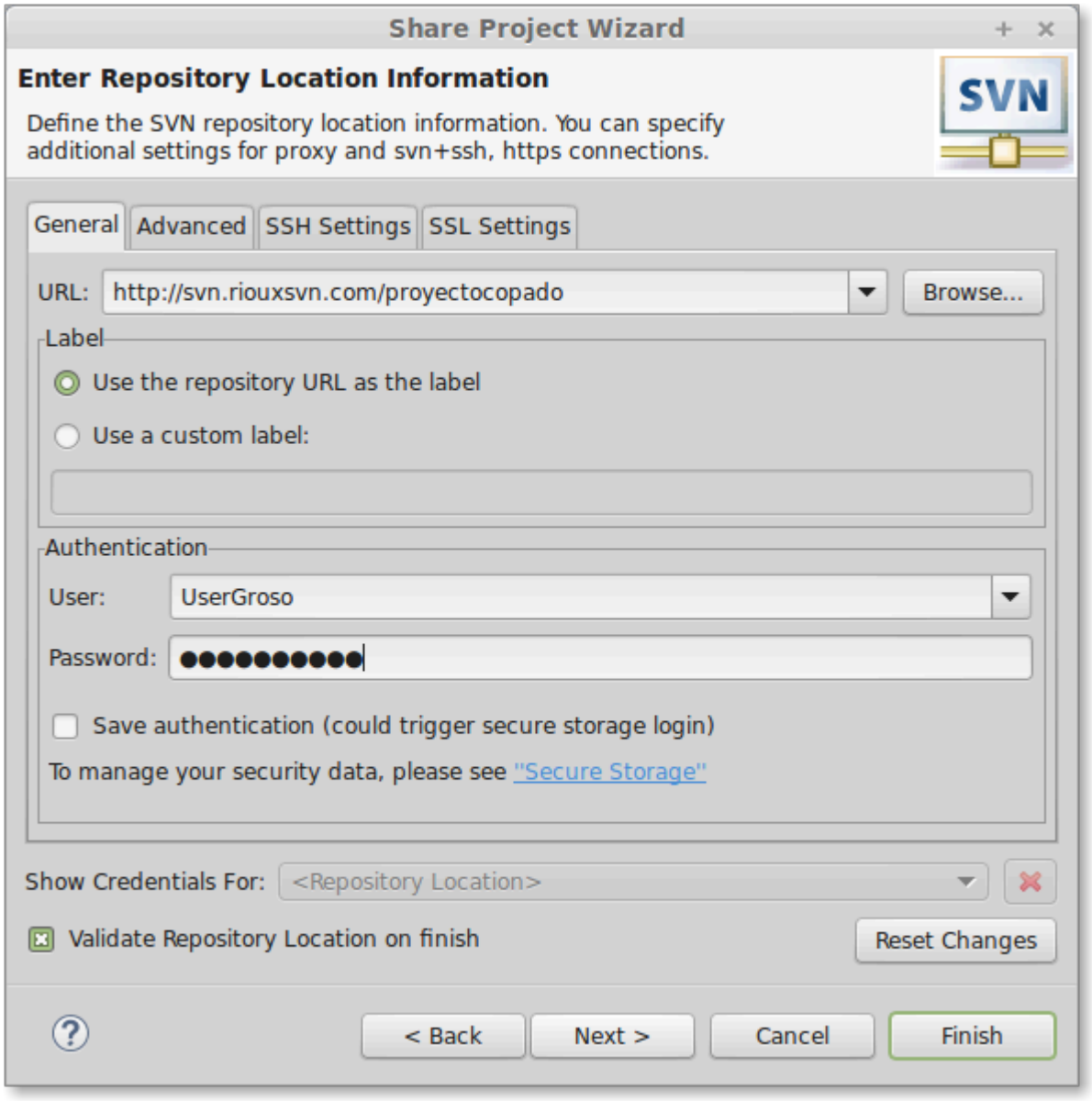


Existen diferentes puntos de vista sobre cómo almacenar los proyectos en un repositorio:

- **Un proyecto por repositorio:** tiene la ventaja de poder aprovechar la estructura que tienen los sitios web con servicios de repositorio SVN al que le suman un issue tracker para hacer el seguimiento y gestión de cambios o bugs o el manejo de una wiki, etc. Tiene la facilidad de hacer sincronizaciones menos pesadas, facilita la gestión de permisos ya que es individual por proyecto, entre otras ventajas.
- **Varios proyectos en un solo repositorio:** si bien tenemos que ser ordenados en la estructura interna de directorios que le estamos dando al repositorio para tener ordenados los proyectos, facilita el acceso a los mismos y tenemos un solo punto de administración.

Por simplicidad se recomienda tener un repositorio diferente por cada proyecto que vayamos a hacer.

En nuestro caso crearemos una nueva ubicación del repositorio seleccionando *Create a new repository location*. Nos va a aparecer una ventana donde podemos completar los datos de nuestro repositorio remoto. Van a poder ver una pantalla como la siguiente:

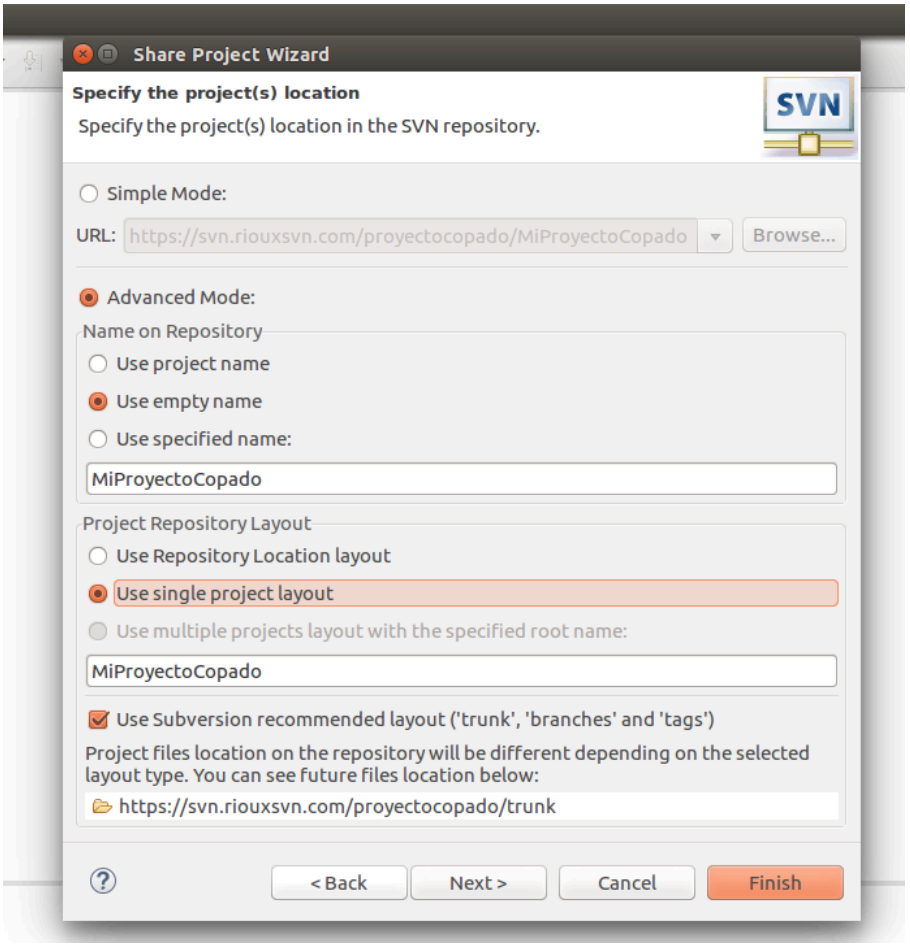


Entonces completamos la URL de nuestro servidor, el usuario y la contraseña y tildamos la opción de *Save authentication*. Habitualmente SVN crea inicialmente una estructura de tres carpetas de trabajo:

1. Trunk: que va a ser la raíz de nuestro proyecto,
2. Branches: es donde se crean ramas de trabajo - cosa que queda fuera del scope de este tutorial - y
3. Tags: que veremos más adelante.

Luego en la ventana siguiente vamos a poder editar el modo en que utilizamos el repositorio. Vamos a seleccionar **Advanced Mode** y marcamos las siguientes opciones:

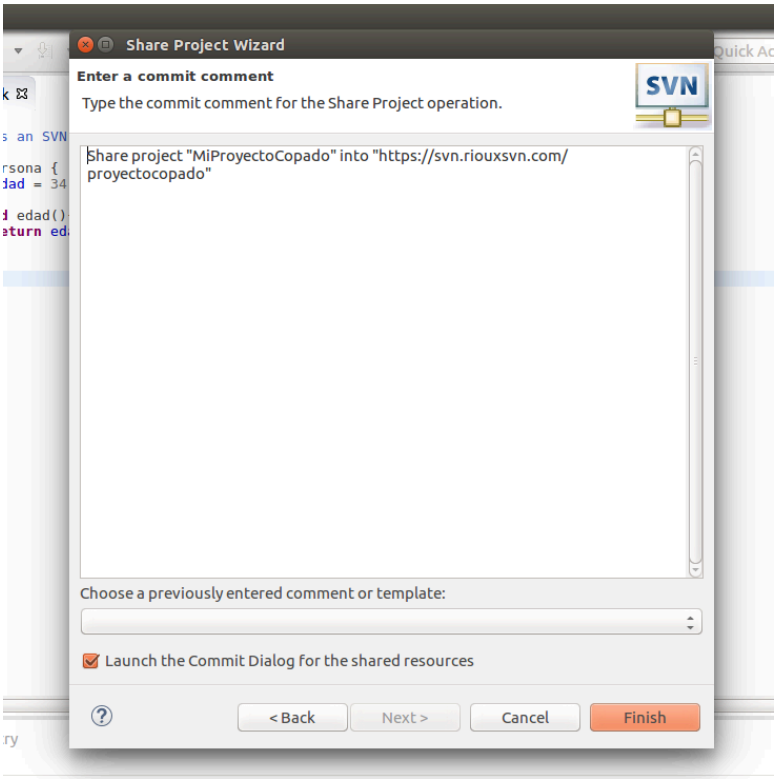
- *use empty name* para **Name on Repository.**
- *use single project layout* para **project repository layout.**
- dejamos tildado *use Subversion recommended layout*.



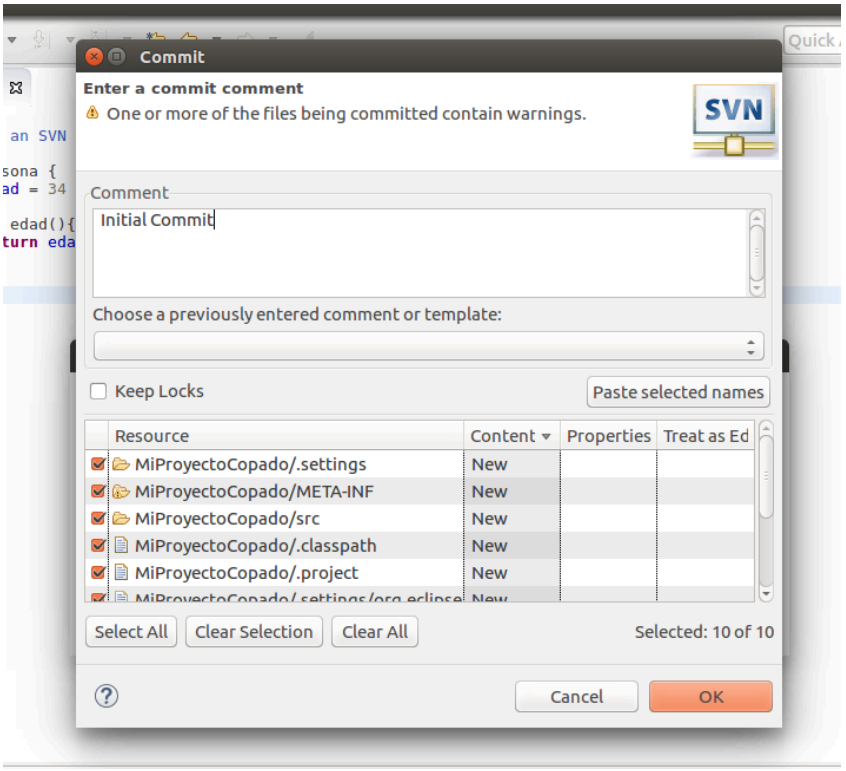
Esto va a permitir acomodar nuestro proyecto dentro de la estructura de carpetas mencionadas previamente que se encuentran creadas en el repositorio remoto.

El paso siguiente nos va a pedir que escribamos un comentario para poder compartir el proyecto. Cada vez que *commiteamos* en SVN, nos permite redactar un breve comentario describiendo cuáles son los cambios que vamos a subir en el commit. Es una muy buena práctica realizar una descripción precisa de nuestros cambios, dado que si en algún momento tenemos que volver a una versión atrás podemos saber con certeza dónde está el cambio al cual queremos volver[4]. La pantalla que veremos será como la siguiente:

[4] En equipos de desarrollo grandes, poner un mensaje de commit como “lala”, “hola mundo”, “cambio groso” se penaliza con una docena de facturas.



Al presionar *Finish*, nuestro proyecto está solamente *compartido*. Para subir los archivos de nuestro proyecto ya compartido al repositorio, nos aparece otra ventana para completar con un comentario opcionalmente distinto:

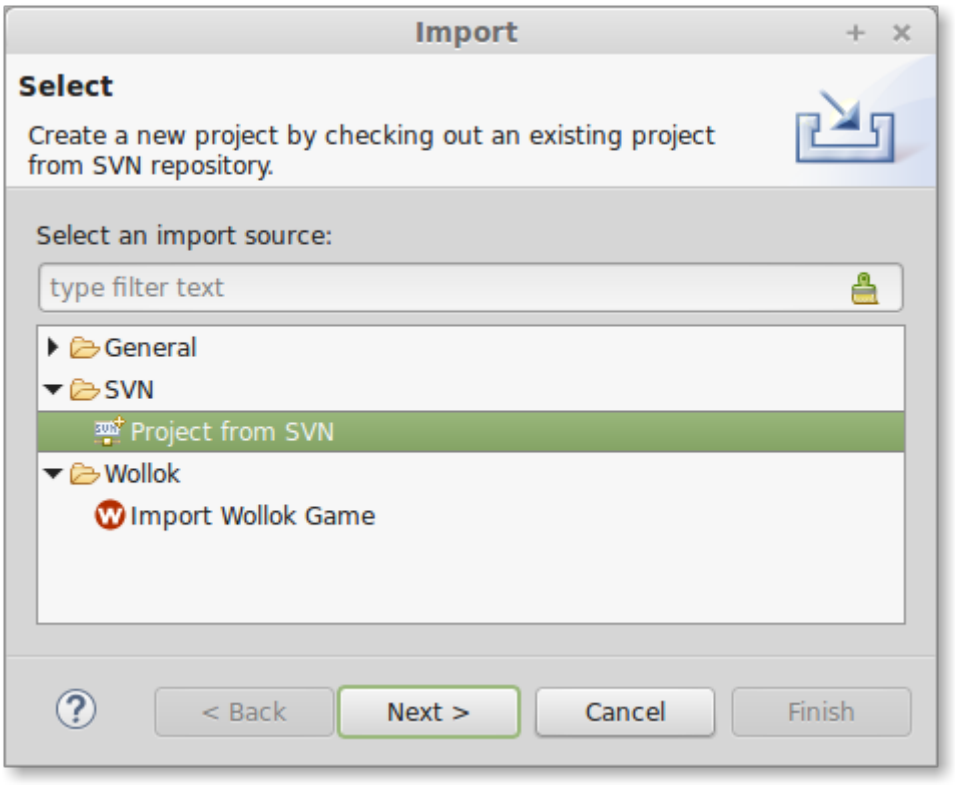


En esta ventana van a tener en la parte superior un campo de texto donde completamos el comentario y en la parte inferior una lista que posee todos los archivos que han sufrido modificaciones. Cuando trabajamos en Wollok debemos subir **TODOS LOS ARCHIVOS**, ya que estos son los que contienen la información necesaria para que eclipse entienda el proyecto como lenguaje Wollok y adapte la estructura a la requerida por el lenguaje.

Le damos ok y..... ¡voilá! ya tenemos nuestro ejemplo subido al repositorio remoto.

3 Bajando un proyecto del repositorio

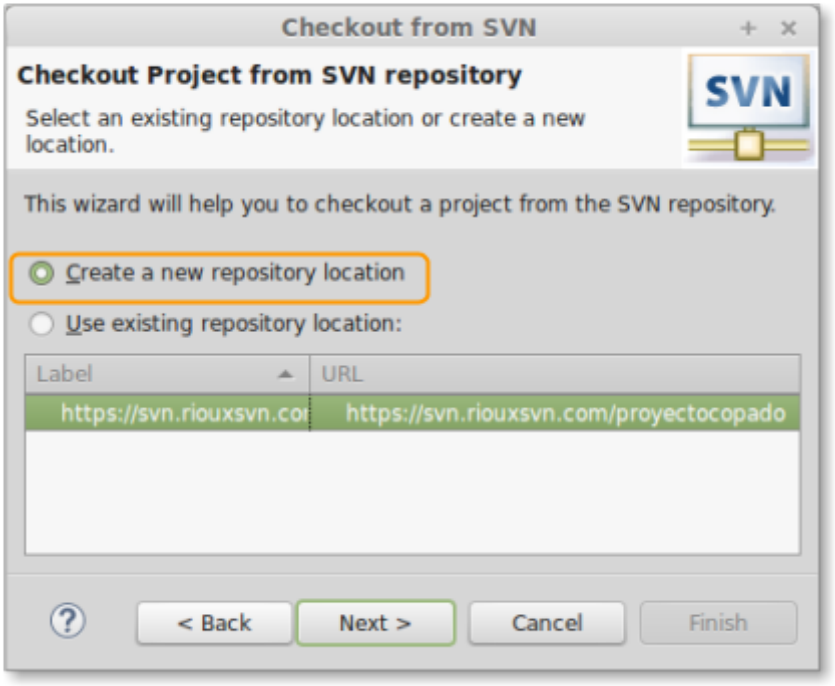
Bien, ahora nos paramos del lado del "resto del grupo" que quiere sincronizarse con el trabajo que subió nuestro primer integrante. Entonces en el Project explorer y hacemos **click derecho -> import**.



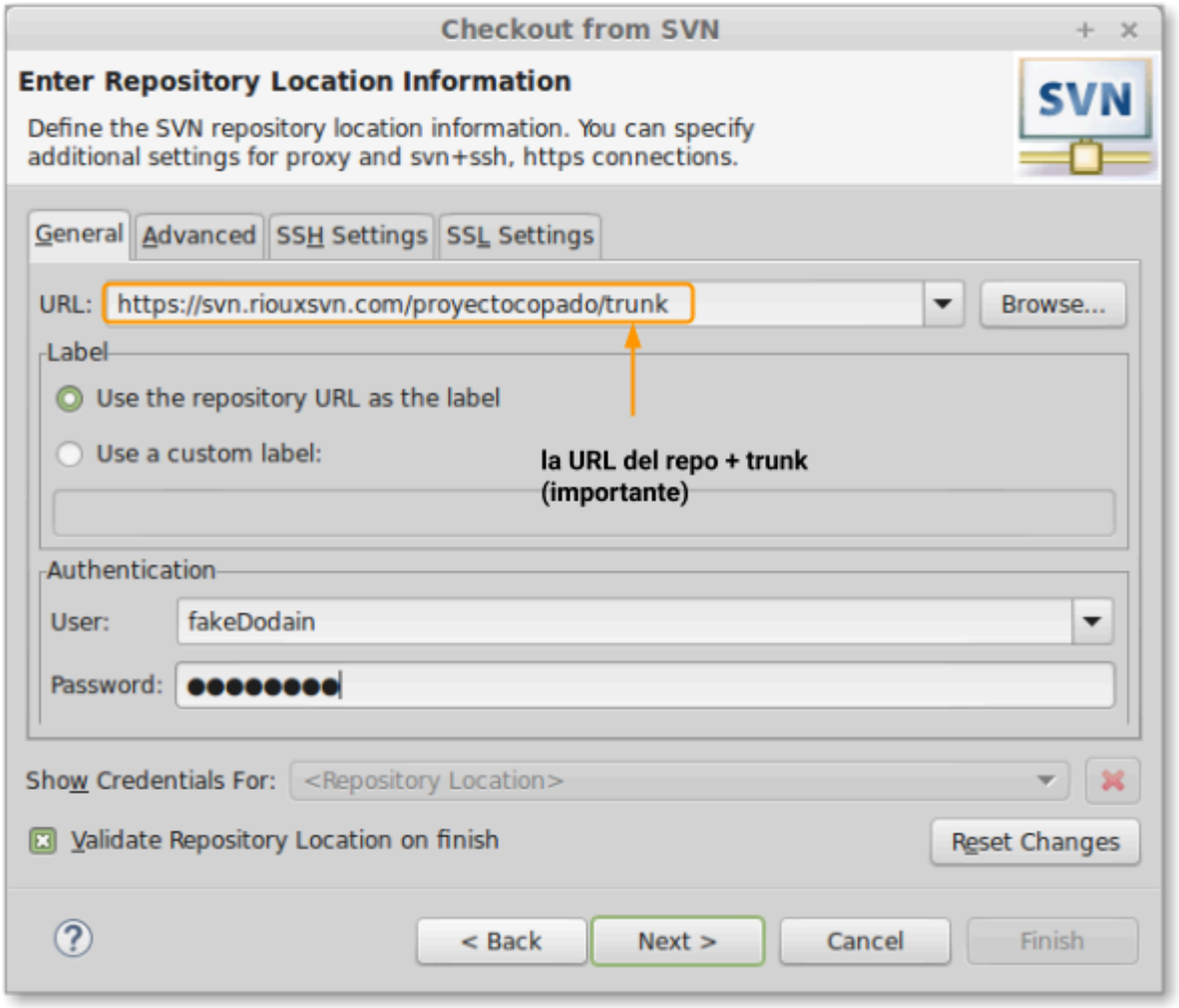
Seleccionamos la opción de project from SVN y aquí volvemos a tener las mismas dos opciones como cuando compartimos un proyecto:

- Si ya tenemos repositorios cargados, podemos seleccionar uno existente o uno nuevo.

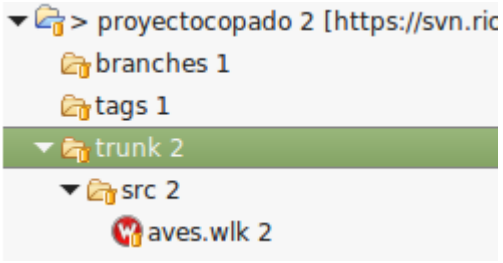
- Cuando se trata de nuestra primera bajada del proyecto, creamos directamente un nuevo repositorio.



Ahora tenemos que seleccionar la URL de nuestro repositorio

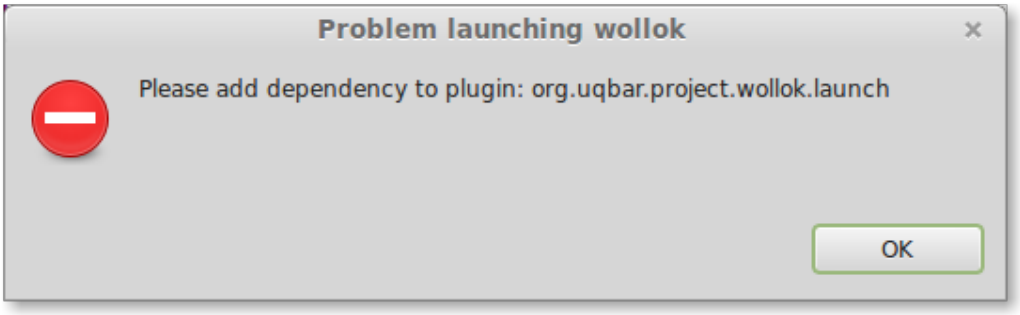


Un detalle importante: supongamos que la URL del proyecto es:
https://svn.riouxsvn.com/proyectocopado



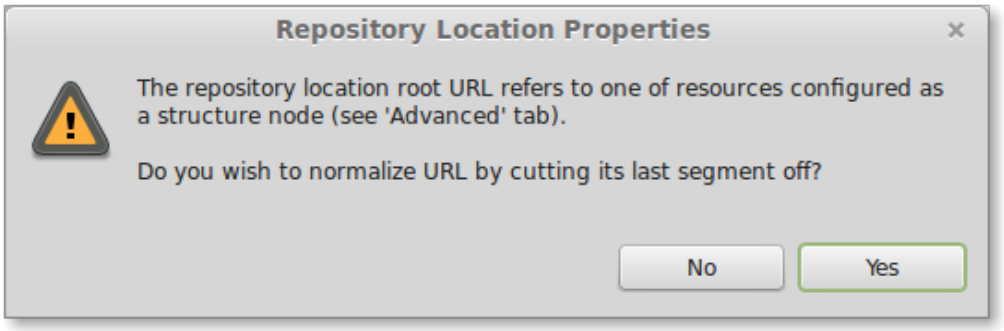
Entonces el checkout debe hacerse sobre la rama principal o *trunk* del proyecto:
https://svn.riouxsvn.com/proyectocopado/trunk de lo contrario nos aparecerá un proyecto con muchas subcarpetas, como se ve en la imagen de la derecha.

Además, cuando intentemos ejecutar un programa, un test o la consola REPL, nos aparecerá un mensaje de error como el que sigue:

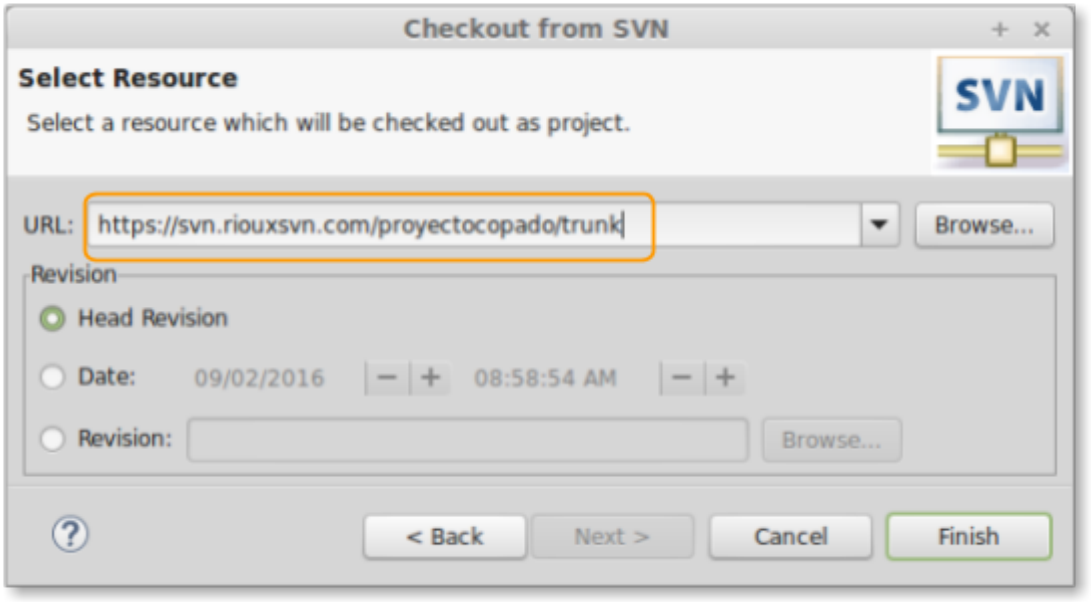


En ese caso, botón derecho sobre el proyecto, Delete y chequeamos que borre el proyecto en nuestro file system, volvemos al paso anterior y nos debemos acordar de escribir la URL del repositorio + "/trunk".

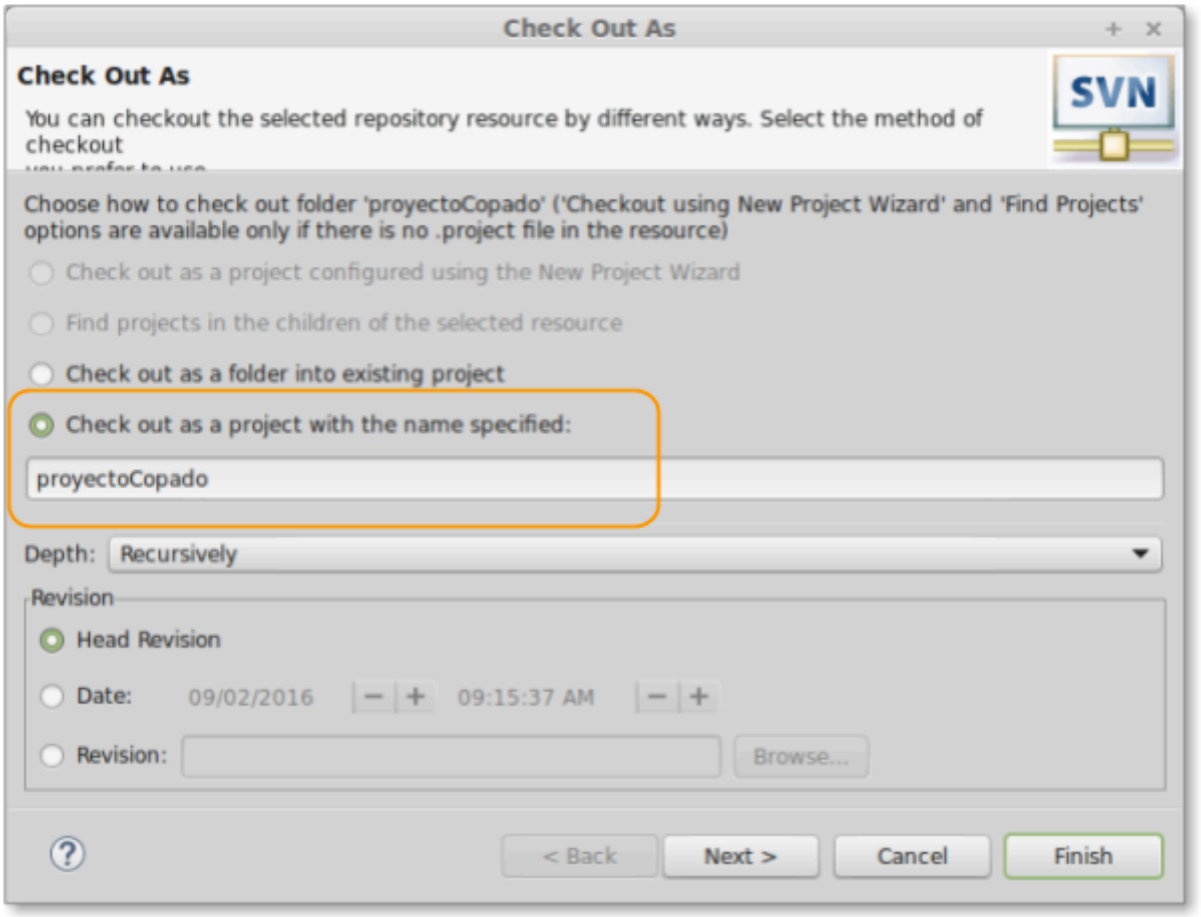
Cuando avanzamos a la siguiente pantalla, nos aparece un cartel de advertencia:



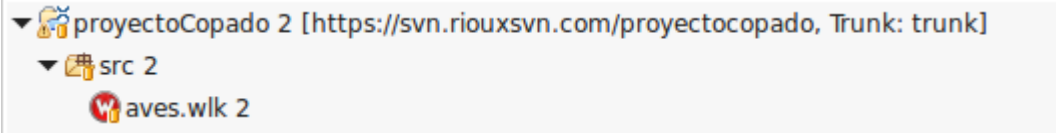
Le decimos que sí.
En la siguiente ventana, una vez más cambiamos la URL a trunk:



y dejamos chequeada la "Head Revision" que es la última subida al repositorio. Al darle Finish todavía necesitamos confirmar el nombre del proyecto



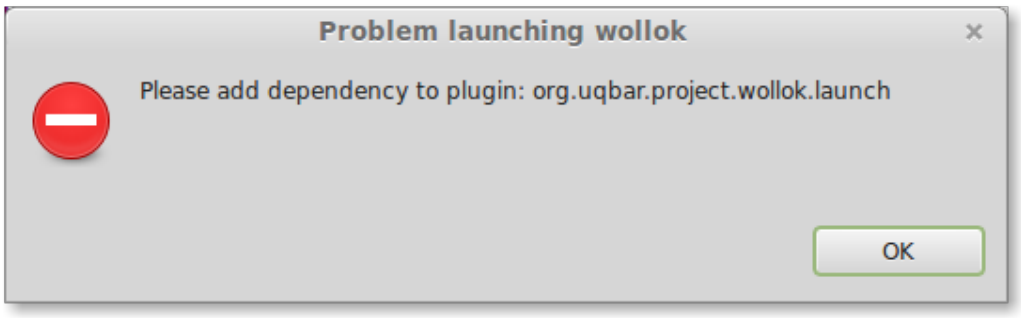
Y ahora sí presionamos el botón Finish.
Nuestro proyecto debería tener esta estructura en el Package Explorer



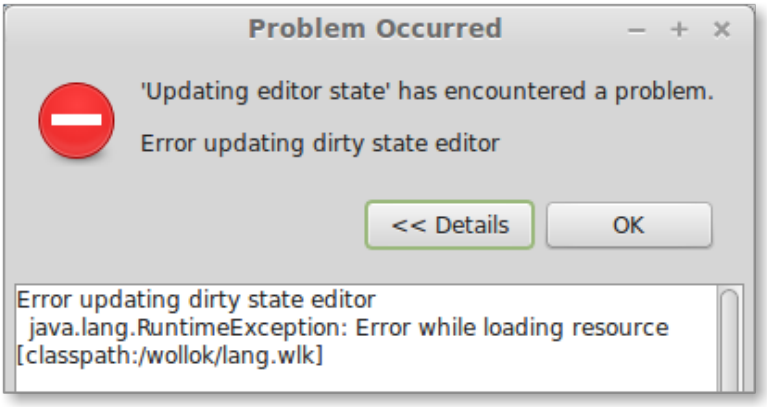
(por supuesto, puede haber otros archivos dependiendo del trabajo que hayan realizado)

4 Importé el proyecto y ¡no me anda nada!

Si el error es



o bien

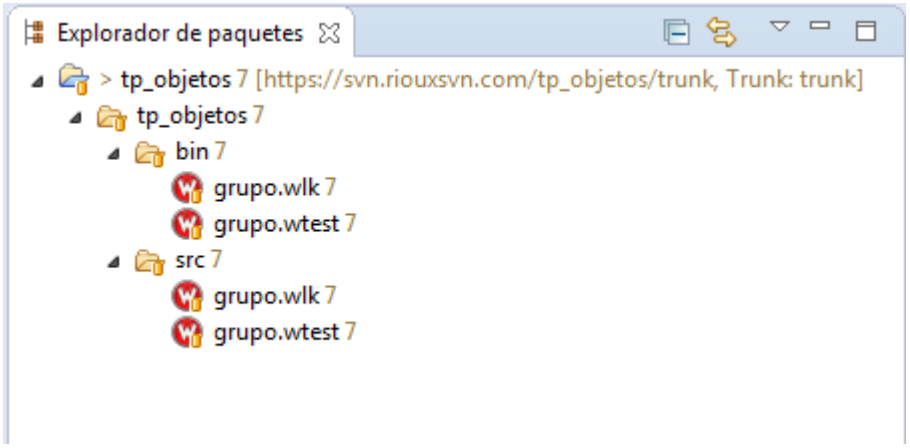


Revisá estas configuraciones

4.1 El proyecto no tiene la estructura Wollok

¿Qué significa “no tiene la estructura Wollok”? Que no hay un source folder src después del raíz del proyecto. Cosas que hay que mirar:

- ¿fui yo? revisá que hayas checkouteado la URL de tu repositorio agregando al final el directorio “/trunk”, como se explica en la página 11
- ¿o fue el que subió primero el proyecto? Si al subir el proyecto no seguiste la recomendación del apunte en la página 7, es posible que cuando otro compañero descargue el proyecto, vea en el directorio trunk un directorio adicional que sea el nombre del proyecto:



En este caso, tp_objetos es el nombre del proyecto original pero además también es una carpeta que está dentro del trunk. Entonces al descargar el proyecto, no lo reconoce como proyecto Wollok, en particular la carpeta src y donde están definidas las dependencias. Entonces no va a funcionar correr tests, ni programas. La solución más fácil es que tu compañero vuelva a subir correctamente el proyecto (de ser necesario borren el repositorio original y creen otro).

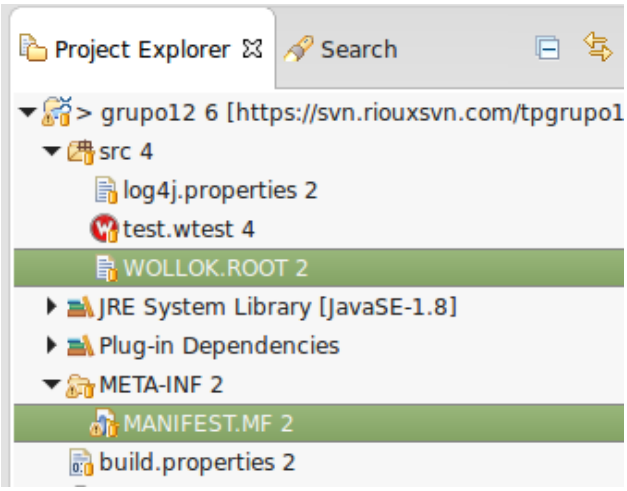
4.2 No se subieron todos los archivos al proyecto Wollok

Si no fue ninguna de las anteriores opciones entonces los mensajes de error arriba descriptos aparecen si previamente no se subieron todos los archivos en el commit inicial del proyecto:

1. **en el primer caso (Please add dependency...) falta un archivo importante que es el MANIFEST.MF que está en el directorio META-INF, con el que se resuelven las dependencias del ejecutable de Wollok.**
2. **En el segundo caso (Error updating dirty state editor...), falta un archivo WOLLOK.ROOT en la carpeta src que establece el path donde se encuentran las clases de las librerías de Wollok**

¿Cómo resolverlo?

1. Pedile a tu compañero que hizo el commit inicial que suba los archivos que están faltando.
2. Los podés copiar de cualquier proyecto vacío Wollok que generes desde el IDE, entrando por el Project Explorer (Window | Show View | Project Explorer) o bien revisando la carpeta desde el sistema operativo. Y claro, después hay que subirlos sí o sí al proyecto, para que no les pase lo mismo al resto de tus compañeros.

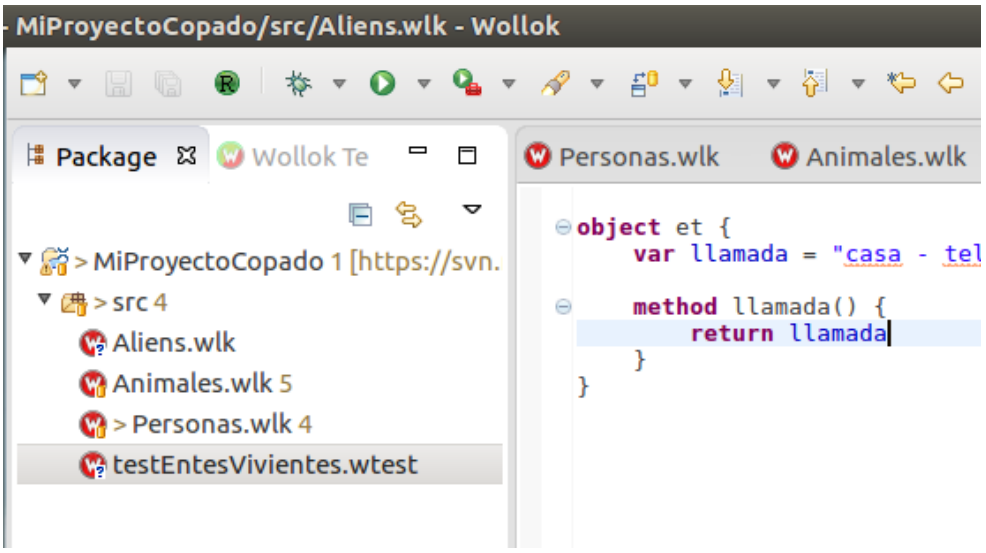


5 Subiendo nuevos cambios

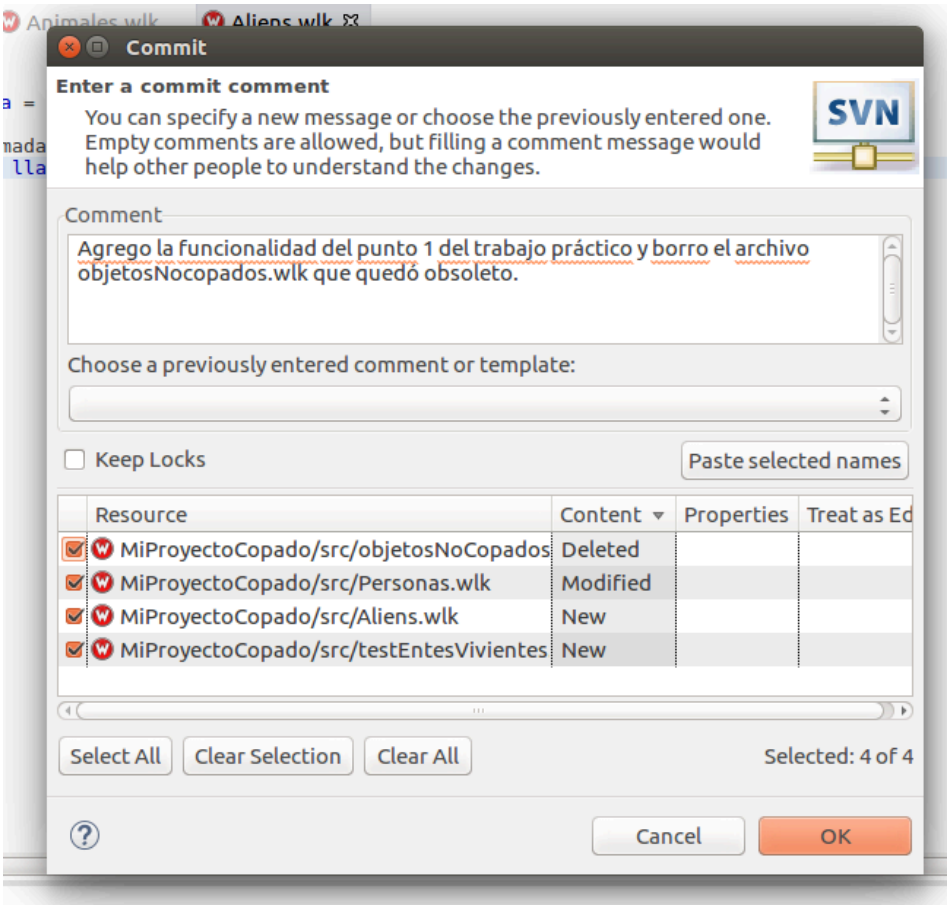
Ok, nuestro equipo está listo para codificar el proyecto que vamos a desarrollar. Entonces ahora el integrante más valiente del grupo toma la posta y va a desarrollar la primer funcionalidad que es requerida. Entonces …. ¿qué puede pasar acá con los archivos del proyecto? Tenemos estas tres opciones:

- Podemos modificar un archivo existente - *Marcado al lado del nombre con un símbolo ">"*

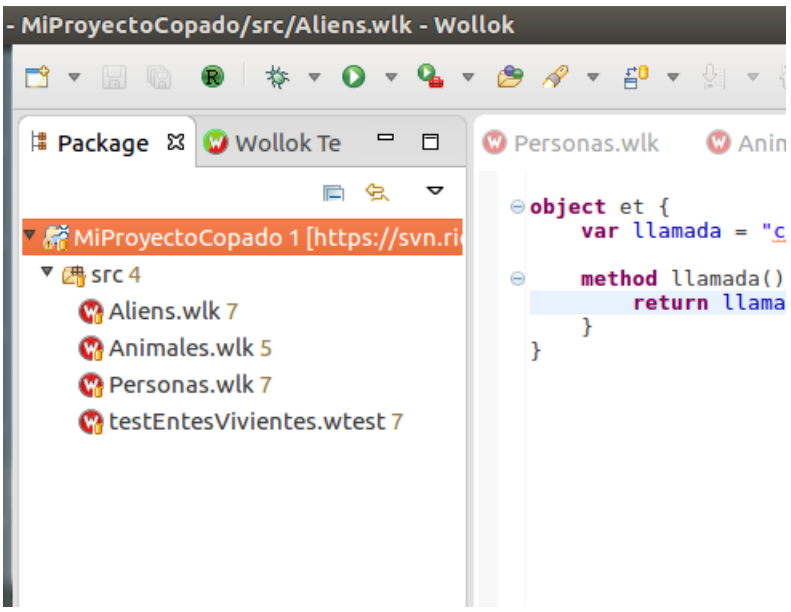
- Agregar un nuevo - *Marcado con un signo de interrogación "?" en el ícono del archivo*
- O bien eliminarlo - *Que desaparece del project browser*



Una vez que tenemos ya decididos todos nuestros cambios, corrimos los test y todo parece estar funcionando correctamente, vamos a pararnos arriba del proyecto y hacemos **click derecho -> team -> commit**. Ahora vamos a tener la misma ventana que nos apareció al hacer el commit inicial. En la misma tenemos la posibilidad de dejar un comentario para este commit y seleccionar los archivos que vamos a commitear



Podemos ver que en la parte inferior tenemos la posibilidad de ir tildando con un checkbox cada archivo que deseo subir, mientras que en la columna *content* nos indica que es lo que va a pasar en el repositorio remoto: El archivo es nuevo, se modifica o se borra. Una vez que presionamos el botón ok, ya se suben nuestros cambios y nos queda el proyecto con el mismo símbolo del cilindro amarillo en todos los archivos.

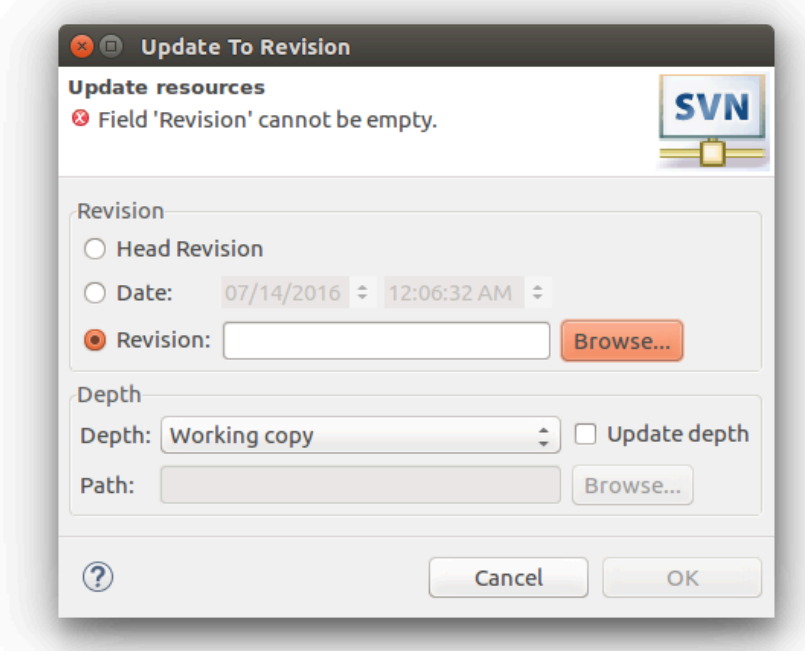


6 Actualizando el proyecto

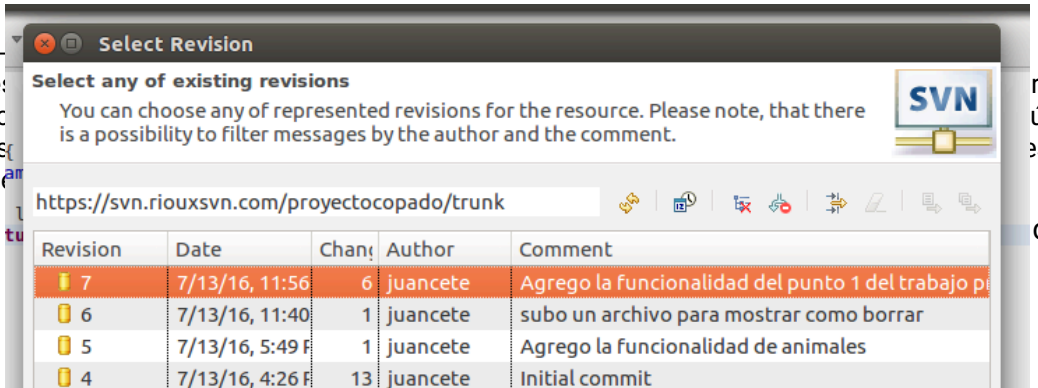
Ahora nuestros compañeros de equipo subieron cambios. Entonces nosotros tenemos que actualizar nuestro contenido del IDE con el del que se encuentra en el repositorio remoto. Para realizar esta operación solo es necesario hacer **click derecho sobre el proyecto -> team -> update**. Automáticamente se va a sincronizar el contenido con el del repositorio.

Es una buena práctica siempre que esté por empezar a codificar una funcionalidad, hacer un update para bajarme la última versión que se encuentre en el repositorio. Esto nos beneficia en evitar a futuro tener conflictos con el código a subir si es que ya sufrió actualizaciones en el período que me tomó llevar a cabo el desarrollo.

Otra utilidad que podemos explotar es buscar una versión determinada para un archivo o bien para el proyecto entero. Esto se puede ejecutar haciendo **click derecho sobre el proyecto/archivo -> team -> update to revision**. Aquí podremos seleccionar Head Revision que es la última versión del repositorio (que sería como hacer un simple update) o bien podemos buscar por fechas o bien seleccionar una revisión en particular[5].



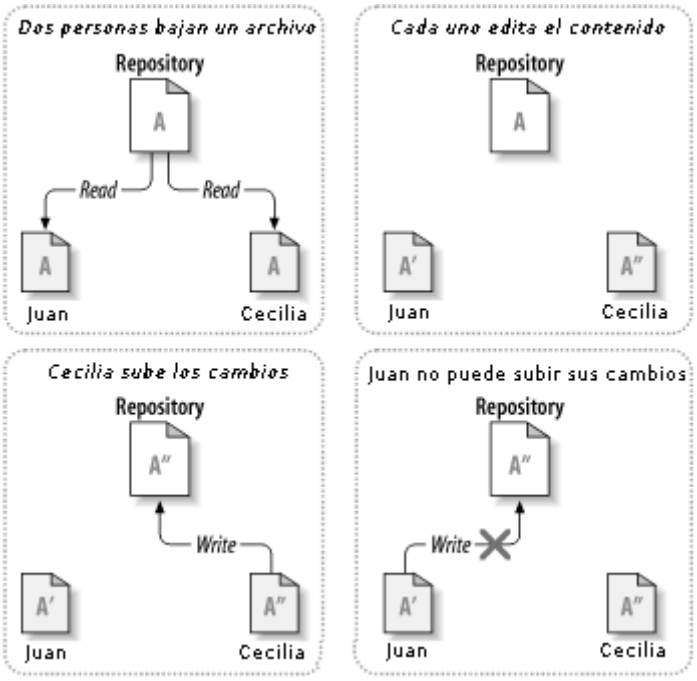
Para la última opción tenemos que seleccionar el option “Revision:” y clickear en browse. Automáticamente vamos a ver una nueva ventana que contiene todas las versiones, sus respectivos cambios y el usuario que realizó el commit en cuestión.



[5] Si tené rchivo en el Pro última versión s es que hiciste d

7 Resolución de conflictos

¿Conflictos? ¿Qué pasa si todos tocamos un mismo archivo y queremos subirlo?

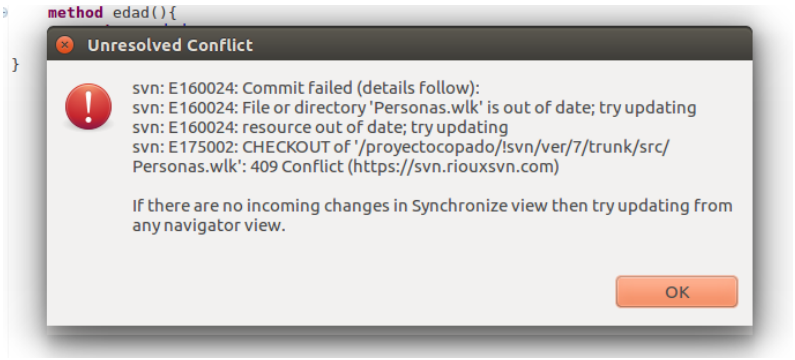


En el ejemplo que mostramos en la imagen superior, tanto Juan como Cecilia se bajan del repositorio la versión A de un archivo. Ambos tocan líneas de código generando dos versiones nuevas e independientes. Claramente Cecilia trabajó más rápido que Juan en su implementación y subió los cambios realizados al repositorio, quedando la versión A" como la vigente. Entonces Juan al momento de subir los cambios se ve alertado por el servidor SVN que le indica que su versión A' (que es más nueva que A) no puede ser subida porque hay una nueva versión en el servidor remoto (que es la evolución de la versión de la que partió Juan: A -> A"). Entonces nos pide que bajemos esos cambios del repositorio remoto, por lo tanto realizamos un *team->update*. Acá pueden suceder dos cosas:

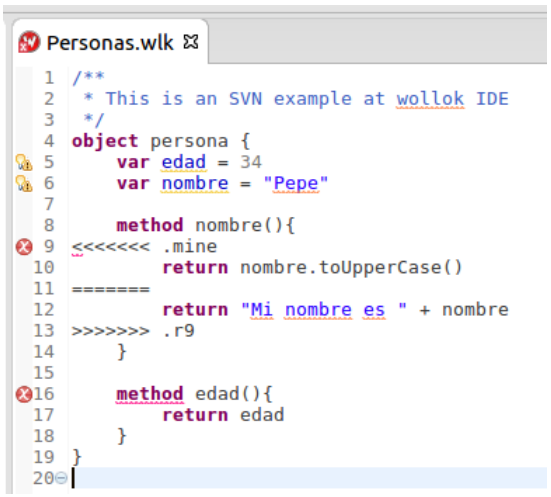
- El cliente SVN fue lo suficientemente inteligente para poder darse cuenta que las líneas de código que modificó Cecilia no se solapan con las que estuvo trabajando Juan y hace un merge automático, es decir que crea una versión **A'"** que contiene los cambios de Juan y de Cecilia. Hacemos un commit y dejamos en el servidor remoto esta última versión.
- En algún punto se modificaron las mismas líneas de código por lo cual el cliente SVN no es capaz de poder mezclar los fuentes y entonces delega en nosotros para que tomemos la decisión adecuada. Vamos a explayarnos un poco más sobre esta mecánica.

Por ejemplo en nuestro proyecto copado que tiene un archivo llamado personas.wlk, tanto Juan como Cecilia modificaron el mismo método (en nuestro caso el método es nombre()). Entonces como habíamos mostrado en el ejemplo anterior, Cecilia sube los cambios

primero y Juan intenta subir los suyos posteriormente. Entonces surge este error:



Hacemos un update primero y vamos a ver que el código nos queda de la siguiente manera:

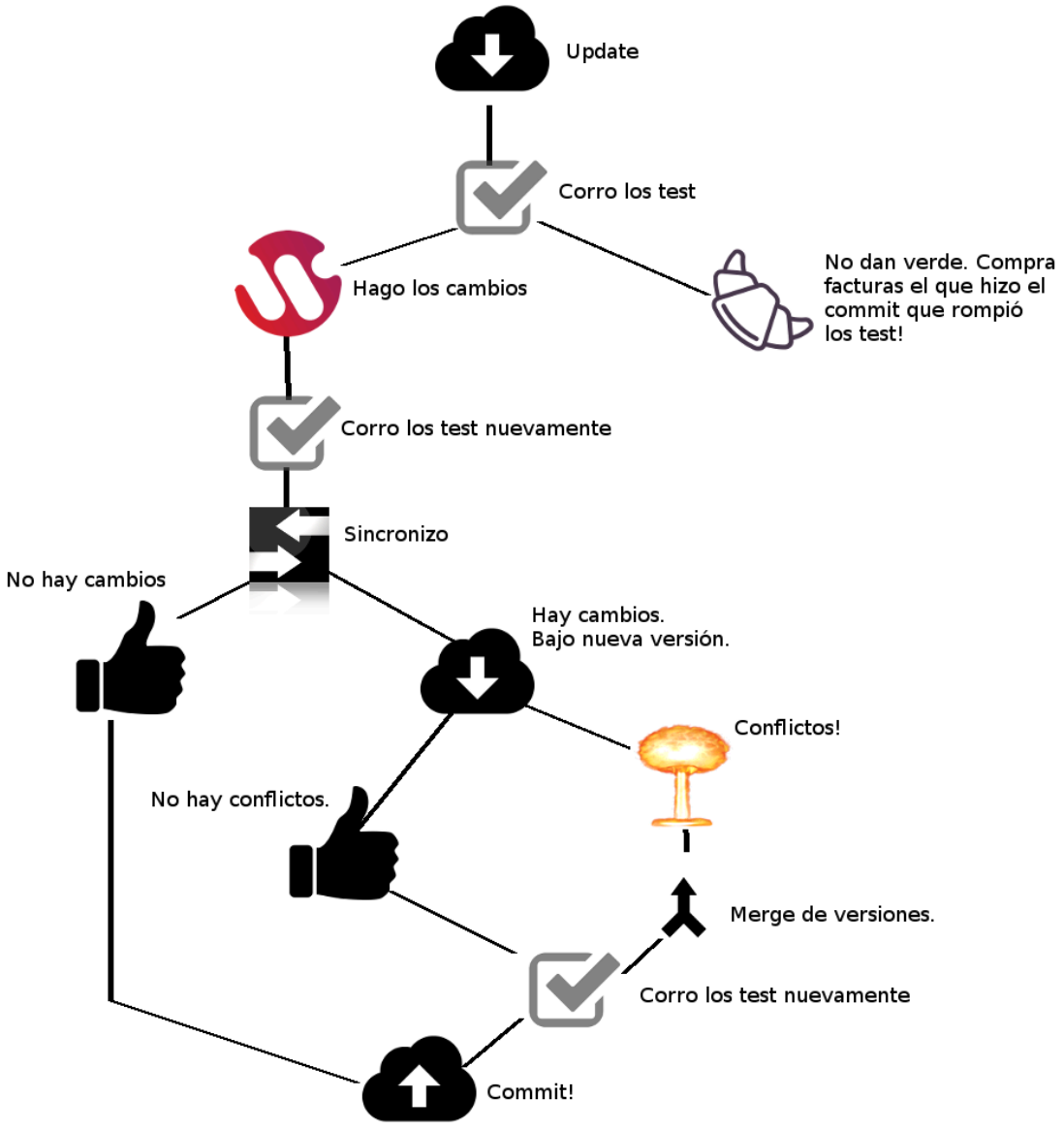


Como se puede ver ahora está indicado con signos de mayor y menor la porción de código afectada y separada por signos de igual el código que escribimos nosotros y el código que se encuentra en el repositorio (mine y r9 - revisión 9 - respectivamente). Como se puede ver el código que deja el plugin de SVN no compila!!! Esto es a propósito para que nosotros tengamos que modificar el archivo en cuestión para que sea coherente con el requerimiento pedido.

Claramente Cecilia concatenó el string “Mi nombre es “ con el atributo nombre para el getter y Juan quería devolver el atributo en mayúscula. Entonces debemos editar el archivo y dejamos que devuelva los string concatenados más el atributo en mayúscula haciendo *click derecho sobre el proyecto -> team->edit conflicts* que nos lleva a esta ventana:

Podemos editar en la ventana Working o bien si nos paramos en la flecha que une las dos ventanas nos aparecerá un ícono para mover la porción de código seleccionada a la ventana working. Una vez terminados los cambios grabamos y nos queda creada la nueva versión. Ahora nos resta indicarle al plugin que ya está resuelto el conflicto, entonces nos paramos arriba del archivo personas.wlk, hacemos *click derecho -> team -> mark as merged*. Solo nos queda hacer un commit de la nueva versión y listo :).

8 Workflow recomendado para trabajar



Recomendamos que sigas este circuito:

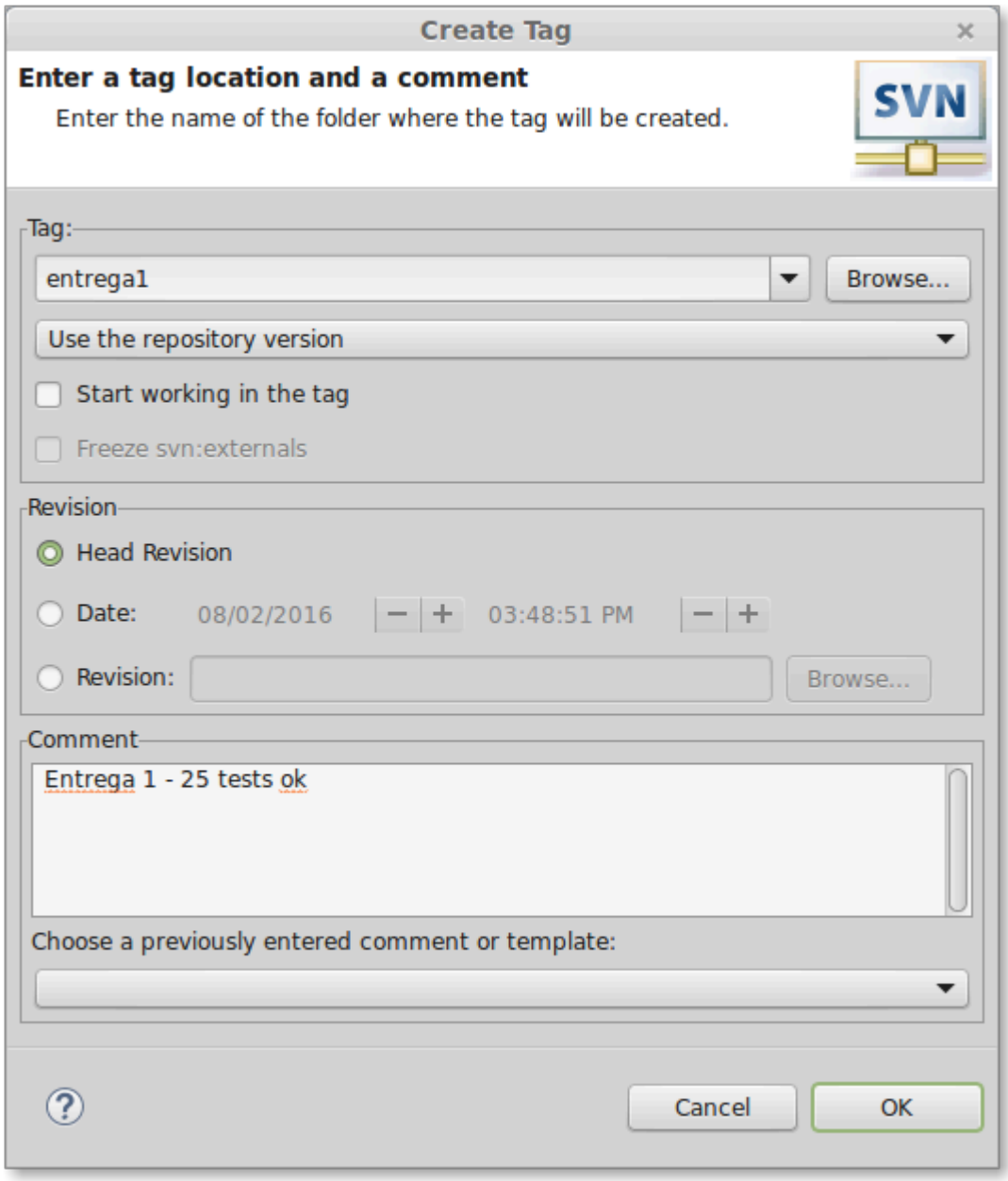
1) Bajo cambios (svn update)
2) Corro tests, si están en rojo hay que avisar al equipo que hay un commit que hizo fallar. Medialunas para el equipo a cargo del que subió el commit.
3) Hago ajustes necesarios.
4) Antes que nada... ¡¡volver a correr los tests!!
5) Ahora sí, sincronizo,
    a) si no hay updates, buenísimo: tiro el commit.
    b) si hay updates, tengo que sincronizar, correr los tests, quizás no hay conflictos, todo bien, tiro el commit si los tests dieron verde.

c) si hay conflictos, ups, hay que resolverlos a mano, correr los tests y commitear una vez que los marcaron como resueltos

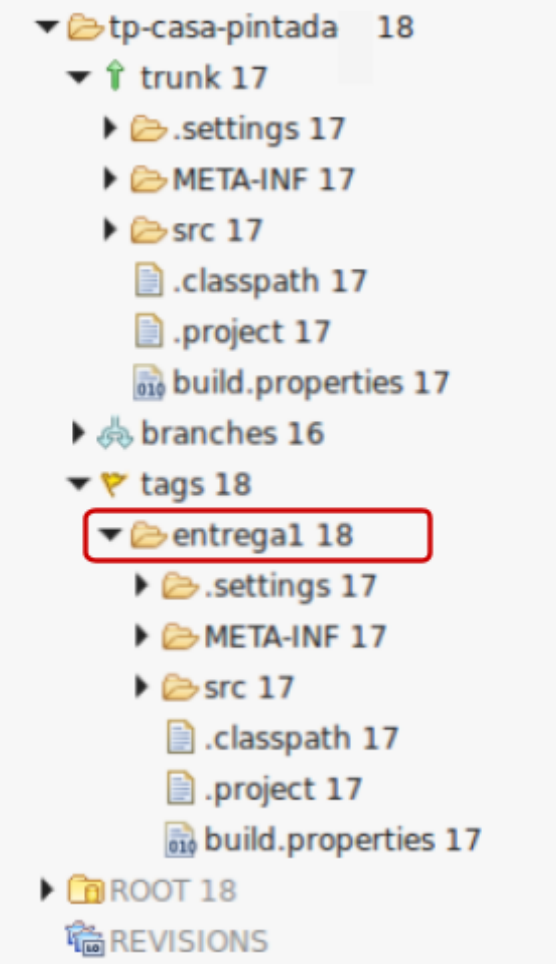
6) Y cuando cerraron la entrega, generan un tag “Entrega x” (ver explicación a continuación

9 Cómo generar un Tag

Esto se hace parado sobre el proyecto, *Team -> Tag*



Seleccionen un nombre para el tag y deberían utilizar una versión previamente guardada en el Server (“Use the repository version”), normalmente la versión actual (Head Revision). Le ponen un comentario representativo de la entrega y finalmente ok. Eso genera una estructura paralela al trunk para el proyecto:



¿Qué utilidad tiene hacer esto? Podemos restaurar el estado del proyecto sin mayores dificultades haciendo un Checkout específico sobre un Tag.

¿Qué pasa con los posteriores commits? Se harán sobre la estructura de trunk, con lo cual nuestro tag “Entrega 1” no sufre modificaciones.