arInstituto Argentino de Radioastronomía
Camino Gral. Belgrano km 40
Prov. BsAs. - ARGENTINA
Tel: +54-221-482'4903 Tel/Fax: +54-221-425-4909

Departamento de Sistemas

SIS-HOW-00100-PR

Procedimiento

Interacción con el Sistema de Control de Versión SVN (Subversión)

Resumen

Se describen los conceptos generales del sistema de control de versión SVN, así como también los procedimientos de interacción con el mismo a través de los programas cliente.

Autor/es: T. Fernando P. Hauscarriaga

Revisor/es: Área Sistemas

Aprobado por: No aplica

Palabras clave: IAR, SVN, Sistemas, Howto, Help

Contenido

1 Objetivo....4
2 Introducción: El sistema de Control de Versión SVN....4
2.1 Consideraciones previas....4
2.2 Conceptos Básicos....4
2.3 El repositorio....4
2.4 Modelos de versionado....5
2.4.1 El problema de compartir archivos....5
2.4.2 La solución bloqueo-modificación-desbloqueo....6
2.4.3 La solución copiar-modificar-mezclar....7
2.5 Subversion en acción....10
2.5.1 Copias de trabajo....10
2.5.2 URLs del repositorio....10
2.5.3 Revisiones....11
2.6 Ciclo básico de trabajo....19
3 SVN en el IAR....20
3.1 WebSVN....21
3.2 Repositorios del Área Observatorio....21
4 Interacción con los repositorios del IAR....21
4.1 Cliente SVN para Microsoft Windows®: ToroiseSVN (La tortugüita)....21
4.2 Checkout: importar un repositorio completo para generar una copia de trabajo local....21
4.3 La copia de trabajo local....25
4.4 Commit: Modificación y publicación de los cambios de un archivo....25
5 Conclusión....26
6 Referencias....26
7 Control de cambios....26

1 Objetivo

Se describen los conceptos generales del Sistema de Control de Versión SVN (Subversion) en conjunto con los procedimientos necesarios para la interacción con el servicio disponible en el IAR.

2 Introducción: El sistema de Control de Versión SVN

2.1 Consideraciones previas

La presente sección (2 Introducción: El sistema de Control de Versión SVN)es una copia casi fiel de algunos de los contenidos del libro de libre distribución "*Version Control with Subversion"* el cual puede encontrar en versión online y en español en [2]. En [3] puede encontrar la versión en pdf en inglés del libro y en [4] puede encontrar la versión en pdf en Español.

Recomendamos la descarga de cualquiera de las versiones en PDF para tener como referencia de escritorio. Alentamos la lectura de éste material más allá del alcance del presente documento para conocer a fondo el sistema de control de versión Subversion.

2.2 Conceptos Básicos

Subversion es un sistema de control de versiones libre y de código fuente abierto. Es decir, Subversion maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un *repositorio* central. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Ésto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. En este aspecto, mucha gente piensa en los sistemas de versiones como en una especie de "máquina del tiempo ".

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar mas rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer por que la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único—si se ha hecho un cambio incorrecto a los datos, simplemente deshaga ese cambio.

Algunos sistemas de control de versiones son también sistemas de administración de configuración de software. Estos sistemas son diseñados específicamente para la administración de árboles de código fuente, y tienen muchas características que son específicas del desarrollo de software— tales como el entendimiento nativo de lenguajes de programación, o el suministro de herramientas para la construcción de software. Sin embargo, Subversion no es uno de estos sistemas. Subversion es un sistema general que puede ser usado para administrar *cualquier* conjunto de ficheros. Para usted, esos ficheros pueden ser código fuente— para otros, cualquier cosa desde la lista de la compra de comestibles hasta combinaciones de vídeo digital y más allá.

2.3 El repositorio

Subversion es un sistema centralizado para compartir información. La parte principal de Subversion es el repositorio, el cual es un almacén central de datos. El repositorio guarda información en forma de *árbol de archivos*—una típica jerarquía de archivos y directorios. Cualquier número de *clientes* puede conectarse al repositorio y luego leer o escribir en esos archivos. Al escribir datos, un cliente pone a disposición de otros la información; al leer datos,

el cliente recibe información de otros. La figura 1: Un sistema cliente/servidor típico, ilustra ésto.

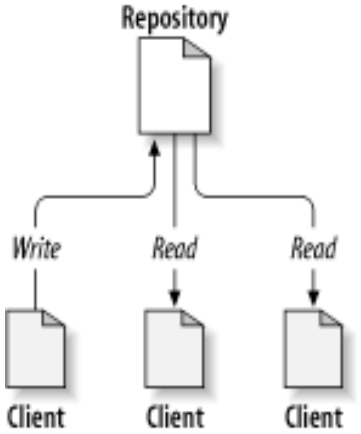

Ilustración 1: Un sistema cliente/servidor típico

Entonces, ¿qué tiene ésto de interesante?. Hasta ahora, suena como la definición del típico servidor de archivos. Y, de hecho, el repositorio *es* una especie de servidor de archivos, pero no del tipo habitual. Lo que hace especial al repositorio de Subversion es que *recuerda todos los cambios* hechos sobre él: cada cambio a cada archivo, e inclusive cambios al propio árbol de directorios, tales como la adición, borrado y reubicación de archivos y directorios.

Cuando un cliente lee datos del repositorio, normalmente sólo ve la ultima versión del árbol de archivos. Sin embargo, el cliente también tiene la posibilidad de ver estados *previos* del sistema de archivos. Por ejemplo, un cliente puede hacer consultas históricas como, "¿Qué contenía este directorio el miércoles pasado?" Esta es la clase de preguntas que resulta esencial en cualquier *sistema de control de versiones*: sistemas que están diseñados para registrar y seguir los cambios en los datos a través del tiempo.

2.4 Modelos de versionado

La misión principal de un sistema de control de versiones es permitir la edición colaborativa y la compartición de los datos. Sin embargo, existen diferentes sistemas que utilizan diferentes estrategias para alcanzar este objetivo.

2.4.1 El problema de compartir archivos

Todos los sistemas de control de versiones tienen que resolver un problema fundamental: ¿Cómo permitirá el sistema a los usuarios el compartir información, pero al mismo tiempo impedirá que se pisen los callos mutuamente de forma accidental? Es muy sencillo para los usuarios el sobreescribir accidentalmente los cambios de los demás en el repositorio.

Considere el escenario mostrado en la Ilustración 2: El problema a evitar. Suponga que tenemos dos colaboradores, Juan y Carmen. Cada uno de ellos decide editar el mismo archivo del repositorio al mismo tiempo. Si Juan guarda sus cambios en el repositorio en primer lugar, es posible que (unos momentos más tarde) Carmen los sobreescriba accidentalmente con su propia versión del archivo. Si bien es cierto que la versión de Juan

no se ha perdido para siempre (porque el sistema recuerda cada cambio), cualquier cambio que Juan haya hecho *no* estará presente en la versión más reciente de Carmen porque, para empezar, ella nunca vio los cambios de Juan. El trabajo de Juan sigue efectivamente perdido —o al menos ausente en la última versión del archivo—y probablemente por accidente. ¡Esta es definitivamente una situación que queremos evitar!

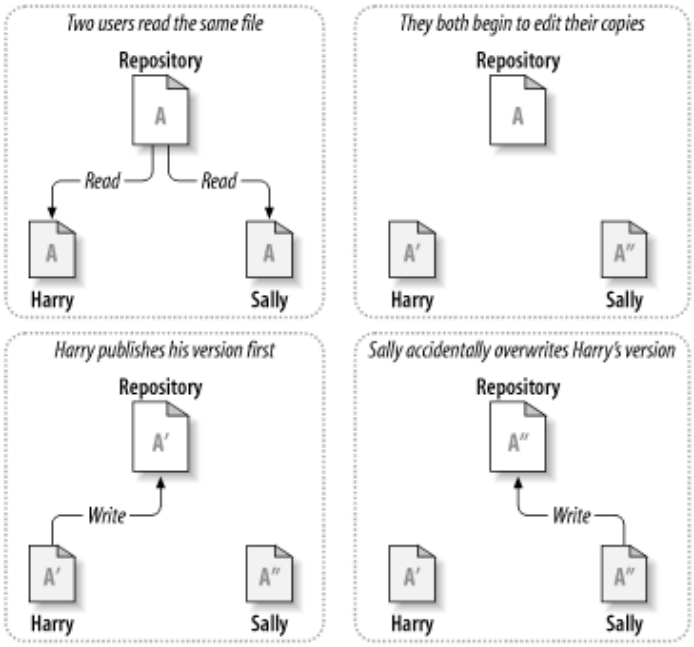

Ilustración 2: El problema a evitar

2.4.2 La solución bloqueo-modificación-desbloqueo

Muchos sistemas de control de versiones utilizan un modelo de *bloqueo-modificación-desbloqueo* para atacar este problema. En un sistema como éste, el repositorio sólo permite a una persona modificar un archivo al mismo tiempo. Juan debe “bloquear” primero el archivo para luego empezar a hacerle cambios. Bloquear un archivo se parece mucho a pedir prestado un libro de la biblioteca; si Juan ha bloqueado el archivo, entonces Carmen no puede hacerle cambios. Por consiguiente, si ella intenta bloquear el archivo, el repositorio rechazará la petición. Todo lo que puede hacer es leer el archivo y esperar a que Juan termine sus cambios y deshaga el bloqueo. Tras desbloquear Juan el archivo, Carmen puede aprovechar su turno bloqueando y editando el archivo. La Ilustración 3: La solución bloqueo-modificación-desbloqueo demuestra esta sencilla solución.

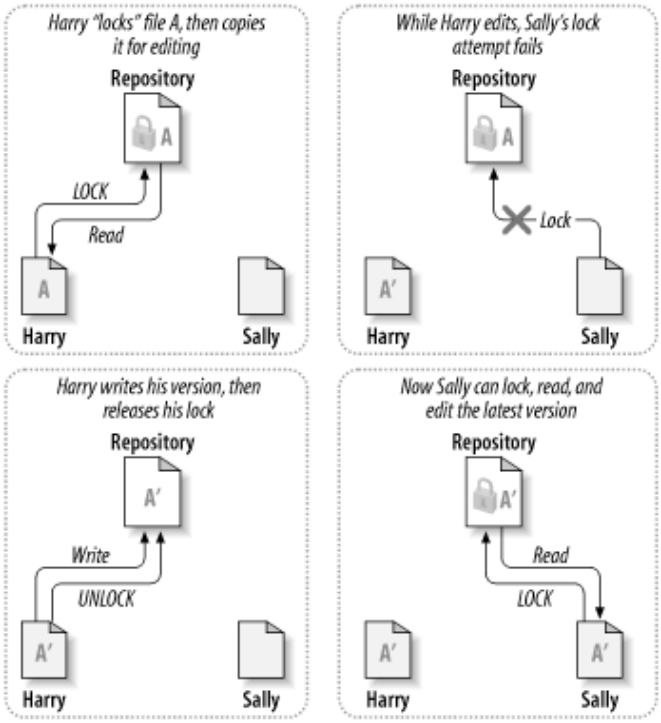

Ilustración 3: La solución bloqueo-modificación-desbloqueo

El problema con el modelo bloqueo-modificación-desbloqueo es que es un tanto restrictivo y a menudo se convierte en un obstáculo para los usuarios:

- *Bloquear puede causar problemas administrativos.* En ocasiones Juan bloqueará un archivo y se olvidará de él. Mientras tanto, como Carmen está aún esperando para editar el archivo, sus manos están atadas. Y luego Juan se va de vacaciones. Ahora Carmen debe conseguir que un administrador deshaga el bloqueo de Juan. La situación termina causando muchas demoras innecesarias y pérdida de tiempo.
- *Bloquear puede causar una serialización innecesaria.* ¿Qué sucede si Juan está editando el inicio de un archivo de texto y Carmen simplemente quiere editar el final del mismo archivo? Estos cambios no se solapan en absoluto. Ambos podrían editar el archivo simultáneamente sin grandes perjuicios, suponiendo que los cambios se combinaran correctamente. No hay necesidad de turnarse en esta situación.
- *Bloquear puede causar una falsa sensación de seguridad.* Imaginemos que Juan bloquea y edita el archivo A, mientras que Carmen bloquea y edita el archivo B al mismo tiempo. Pero suponga que A y B dependen uno del otro y que los cambios hechos a cada uno de ellos son semánticamente incompatibles. Súbitamente A y B ya no funcionan juntos. El sistema de bloqueo se mostró ineficaz a la hora de evitar el problema—sin embargo, y de algún modo, ofreció una falsa sensación de seguridad. Es fácil para Juan y Carmen imaginar que al bloquear archivos, cada uno está empezando una tarea segura y aislada, lo cual les inhibe de discutir sus cambios incompatibles desde un principio.

2.4.3 La solución copiar-modificar-mezclar

Subversion, CVS y otros sistemas de control de versiones utilizan un modelo del tipo *copiar-modificar-mezclar* como alternativa al bloqueo. En este modelo, el cliente de cada

usuario se conecta al repositorio del proyecto y crea una *copia de trabajo* personal—una réplica local de los archivos y directorios del repositorio. Los usuarios pueden entonces trabajar en paralelo, modificando sus copias privadas. Finalmente, todas las copias privadas se combinan (o mezclan) en una nueva versión final. El sistema de control de versiones a menudo ayuda con la mezcla, pero en última instancia es un ser humano el responsable de hacer que ésto suceda correctamente.

He aquí un ejemplo. Digamos que Juan y Carmen crean sendas copias de trabajo del mismo proyecto, extraídas del repositorio. Ambos trabajan concurrentemente y hacen cambios a un mismo archivo A dentro de sus copias. Carmen guarda sus cambios en el repositorio primero. Cuando Juan intenta guardar sus cambios más tarde, el repositorio le informa de que su archivo A está *desactualizado*. En otras palabras, que el archivo A en el repositorio ha sufrido algún cambio desde que lo copió por última vez. Por tanto, Juan le pide a su cliente que *mezcle* cualquier cambio nuevo del repositorio con su copia de trabajo del archivo A. Es probable que los cambios de Carmen no se solapen con los suyos; así que una vez que tiene ambos juegos de cambios integrados, Juan guarda su copia de trabajo de nuevo en el repositorio. En Ilustración 4: La solución copiar-modificar-mezclar y Ilustración 5: La solución copiar-modificar-mezclar (continuación) muestran este proceso.

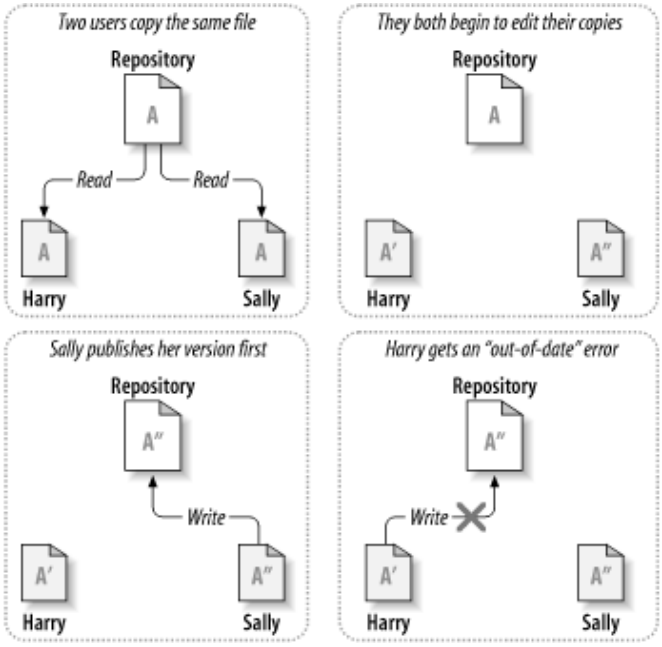

Ilustración 4: La solución copiar-modificar-mezclar

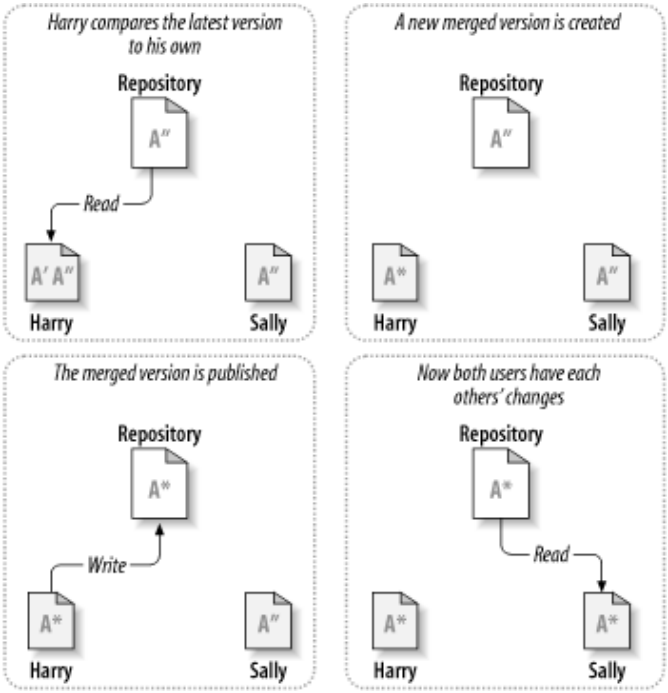

Ilustración 5: La solución copiar-modificar-mezclar (continuación)

¿Pero qué ocurre si los cambios de Carmen *sí* se solapan con los de Juan? ¿Entonces qué? Esta situación se conoce como *conflicto* y no suele suponer un gran problema. Cuando Juan le pide a su cliente que mezcle los últimos cambios del repositorio en su copia de trabajo, su copia del archivo A se marca de algún modo para indicar que está en estado de conflicto: Juan podrá ver ambos conjuntos de cambios conflictivos y escoger manualmente entre ellos. Observe que el programa no puede resolver automáticamente los conflictos; sólo los humanos son capaces de entender y tomar las decisiones inteligentes oportunas. Una vez que Juan ha resuelto manualmente los cambios solapados—posiblemente después de discutirlos con Carmen—ya podrá guardar con seguridad el archivo mezclado en el repositorio.

La solución copiar-modificar-mezclar puede sonar un tanto caótica, pero en la práctica funciona extremadamente bien. Los usuarios pueden trabajar en paralelo, sin tener que esperarse el uno al otro. Cuando trabajan en los mismos archivos, sucede que la mayoría de sus cambios concurrentes no se solapan en absoluto; los conflictos son poco frecuentes. El tiempo que toma resolver los conflictos es mucho menor que el tiempo perdido por un sistema de bloqueos.

Al final, todo desemboca en un factor crítico: la comunicación entre los usuarios. **Cuando los usuarios se comunican pobremente, los conflictos tanto sintácticos como semánticos aumentan. Ningún sistema puede forzar a los usuarios a comunicarse perfectamente, y ningún sistema puede detectar conflictos semánticos. Por consiguiente, no tiene sentido dejarse adormecer por la falsa promesa de que un sistema de bloqueos evitará de algún modo los conflictos; en la práctica, el bloqueo parece inhibir la productividad más que otra cosa.**

2.5 Subversion en acción

2.5.1 Copias de trabajo

Ya ha leído acerca de las copias de trabajo; ahora demostraremos cómo las crea y las usa el cliente de Subversion.

Una copia de trabajo de Subversion es un árbol de directorios corriente de su sistema de archivos local, conteniendo una colección de archivos. Usted puede editar estos archivos del modo que prefiera y si se trata de archivos de código fuente, podrá compilar su programa a partir de ellos de la manera habitual. Su copia de trabajo es su área de trabajo privada: Subversion nunca incorporará los cambios de otra gente o pondrá a disposición de otros sus cambios hasta que usted le indique explícitamente que lo haga.

Tras hacer algunos cambios a los archivos en su copia de trabajo y verificar que funcionan correctamente, Subversion le proporciona comandos para “publicar” sus cambios al resto de personas que trabajan con usted en su proyecto (escribiendo en el repositorio). Si las demás personas publican sus propios cambios, Subversion le proporciona comandos para mezclar estos cambios en su directorio de trabajo (leyendo del repositorio).

Una copia de trabajo también contiene algunos archivos extra, creados y mantenidos por Subversion para ayudarle a ejecutar estos comandos. En particular, cada directorio de su copia de trabajo contiene un subdirectorio llamado `.svn`, también conocido como el *directorio administrativo* de la copia de trabajo. Los archivos en cada directorio administrativo ayudan a Subversion a reconocer qué archivos contienen cambios no publicados y qué archivos están desactualizados con respecto al trabajo hecho por los demás.

Para conseguir una copia de trabajo, debe ejecutar primero un ***check out*** de algún subárbol del repositorio. (El término inglés “check out” puede sonar como si tuviera algo que ver con bloquear o reservar recursos, pero no es así; tan sólo crea una copia privada del proyecto para usted).

Para publicar sus cambios a otros, usted puede utilizar el comando **commit** de Subversion.

La salida del comando **svn update** indica que Subversion actualizó el contenido de una copia de trabajo local con los datos de un repositorio.

2.5.2 URLs del repositorio

A los repositorios de Subversion se puede acceder a través de diferentes métodos—en el disco local, o a través de varios protocolos de red. Sin embargo, la ubicación de un repositorio es siempre un URL. La tabla 2-1 describe la correspondencia entre los diferentes esquemas de URL y los métodos de acceso disponibles.

Esquema	Método de acceso
file:///	acceso directo al repositorio (en disco local)
http://	acceso vía protocolo WebDAV a un servidor Apache que entiende de Subversion
https://	igual que http://, pero con cifrado SSL.
svn://	acceso vía un protocolo personalizado a un servidor svnserve.
svn+ssh://	igual que svn://, pero a través de un túnel SSH.

Tabla 1: URLs de Acceso al Repositorio

En general, los URLs de Subversion utilizan la sintaxis estándar, permitiendo la especificación de nombres de servidores y números de puertos como parte del URL. Recuerde que el método de acceso `file:` es válido sólo para ubicaciones en el mismo servidor donde se ejecuta el cliente—de hecho, se requiere por convención que la parte del

URL con el nombre del servidor esté ausente o sea `localhost`.

2.5.3 Revisiones

Una operación svn **commit** puede publicar los cambios sobre cualquier número de ficheros y directorios como una única transacción atómica. En su copia privada, usted puede cambiar el contenido de los ficheros, crear, borrar, renombrar y copiar ficheros y directorios, y luego enviar el conjunto entero de cambios como si se tratara de una unidad.

En el repositorio, cada cambio es tratado como una transacción atómica: o bien se realizan todos los cambios, o no se realiza ninguno. Subversion trata de conservar esta atomicidad para hacer frente a posibles fallos del programa, fallos del sistema, problemas con la red, y otras acciones del usuario.

Cada vez que el repositorio acepta un envío, éste da lugar a un nuevo estado del árbol de ficheros llamado *revisión*. A cada revisión se le asigna un número natural único, una unidad mayor que el número de la revisión anterior. La revisión inicial de un repositorio recién creado se numera con el cero, y consiste únicamente en un directorio raíz vacío.

La Ilustración 6: El repositorio ilustra una manera interesante de ver el repositorio. Imagine un array de números de revisión, comenzando por el 0, que se extiende de izquierda a derecha. Cada número de revisión tiene un árbol de ficheros colgando debajo de él, y cada árbol es una “instantánea” del aspecto del repositorio tras cada envío.

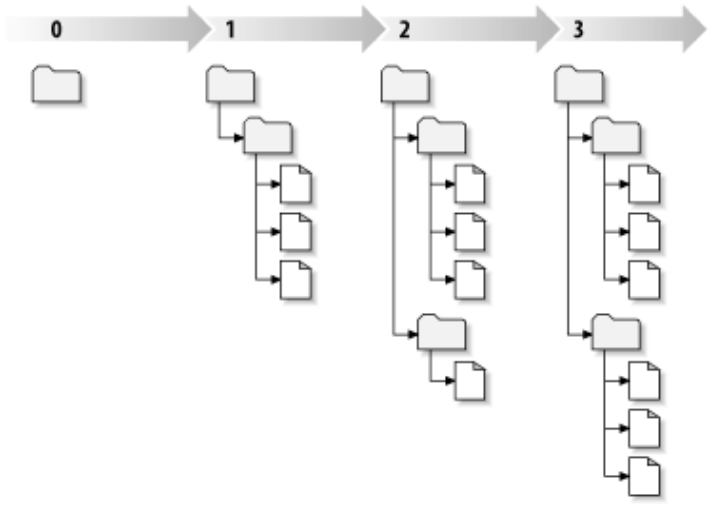

Ilustración 6: El repositorio

2.6 Ciclo básico de trabajo

El ciclo de trabajo típico se parece a esto:

- **Actualizar su copia de trabajo local**
  - svn **update**

    Cuando se trabaja en un proyecto con un equipo, usted querrá actualizar su copia de trabajo local para recibir cualquier cambio hecho desde su última actualización por otros desarrolladores en el proyecto. Use **svn update** para poner a su copia de trabajo local en sincronía con la última revisión en el

repositorio. Es muy recomendable hacer una actualización de nuestra copia de trabajo local antes de comenzar a trabajar.

- **Hacer cambios**
  - svn **add** foo

    Programa añadir `foo` al repositorio. Cuando haga su próximo envío, `foo` se convertirá en hijo de su directorio padre. Fíjese que si `foo` es un directorio, todo por debajo de `foo` será programado para la adición . Si solo quiere añadir el propio `foo`, pase la opción `--non-recursive` (`-N`) .
  - svn **delete**

    Programa borrar `foo` del repositorio. Si `foo` es un fichero, se borrará inmediatamente de su copia de trabajo local. Si `foo` es un directorio, este no es borrado, pero Subversion lo programa para borrarlo. Cuando envíe sus cambios, `foo` será borrado de su copia de trabajo y del repositorio.
  - svn **copy** foo bar

    Crea un nuevo objeto `bar` como duplicado de `foo`. `bar` es automáticamente programado para la adición. Cuando `bar` es añadido al repositorio en el siguiente envío de cambios, su historia de copia es registrada (como que originalmente viene de `foo`).svn copy no crea directorios intermedios.
  - svn **move** foo bar

    Este comando funciona exactamente igual que svn copy foo bar; svn delete foo. Esto es, se programa `bar` para la adición como una copia de `foo`, y se programa `foo` para la eliminación. svn move no crea directorios intermedios.

- **Examinar sus cambios**
  - svn **status**

    Si ejecuta **svn status** en lo alto de su copia de trabajo sin argumentos, detectará todos los cambios de fichero y árbol que usted ha hecho. Este ejemplo está diseñado para mostrar todos los códigos de estado diferentes que **svn status** puede devolver.
  - svn **diff**

    Otra manera de examinar sus cambios es con el comando **svn diff**. Puede descubrir *exactamente* cómo ha modificado cosas ejecutando **svn diff** sin argumentos, el cual imprime los cambios de los ficheros en formato unificado del diff
  - svn **revert**

    Subversion invierte el fichero a un estado pre-modificado reescribiéndolo con la copia “prístina”. Pero también observar que **svn revert** puede deshacer *cualquier* operación programada—por ejemplo, usted puede decidir que no quiere añadir un nuevo fichero después de todo.
- **Fusionar los cambios de otros en su copia de trabajo**
  - svn **merge**

    Aplica las diferencias de dos fuentes diferentes a una copia de trabajo. Se utiliza principalmente para resolver conflictos.

- svn **resolved** foo

  Indica que los conflictos sobre el fichero foo ya han sido resueltos.

- **Enviar sus cambios**

  - svn **commit**

    Envía todos sus cambios al repositorio. Cuando usted envía un cambio, necesita proveer un *mensaje de registro*, describiendo su cambio. Su mensaje de registro será adjuntado a la nueva revisión que ha creado.

3 SVN en el IAR

Originalmente la necesidad de implementar el servicio SVN en el IAR provino del área de Transferencia de Tecnología ya que era necesario establecer un “espacio” común para trabajar en la documentación de sus proyectos, el Área de Sistemas entonces implementó el sistema de control de versión SVN en el servidor tux sobre la plataforma httpd *apache*, heredando todas las características de control de acceso que éste último posee.

Debido a las características extensibles del servidor *apache* es posible tener dos árboles de repositorios completamente independientes uno del otro, ésto permitió que el Área de Transferencia y el Área Observatorio tengan sus árboles de repositorios completamente independientes el uno del otro, evitando que se solapen los proyectos que puedan tener mismo nombre.

3.1 WebSVN

WebSVN es un servicio que permite explorar los repositorios SVN con un navegador web como *Mozilla Firefox*, permite tener una vista rápida de los repositorios disponibles, ver el historial de cambios y hasta descargar archivos puntuales del repositorio (sin necesidad de tener que hacer una copia de trabajo local), se puede acceder al servicio WebSVN del Área Observatorio entrado en la siguiente URL:

http://www.iar.unlp.edu.ar/svnobs

Al entrar en ésta página, el sistema pedirá que se autentique.

Es importante tener en cuenta que la finalidad de éste servicio es netamente de exploración (sólo lectura) es decir: no se pueden hacer modificaciones en los repositorios.

3.2 Repositorios del Área Observatorio

En la sección anterior vimos que se puede acceder al área de sólo lectura de los repositorios para poder explorarlos, pero para poder interactuar libremente con un repositorio es necesario otra URL:

http://www.iar.unlp.edu.ar/svn/obs/*nombre_del_repositorio*

dónde *nombre_del_repositorio* será el nombre del repositorio con el que queremos trabajar. En las próximas secciones veremos cómo es una sesión de trabajo con un respositorio SVN. Obsérvese que en la URL hay una barra “/” entre svn y obs.

4 Interacción con los repositorios del IAR

4.1 Cliente SVN para *Microsoft Windows®: ToroiseSVN* (La tortugüita)

SVN es Software de Código Abierto, ésto quiere decir entre otras cosas que está disponible para casi todas las plataformas que existen hoy en día. *Microsoft Windows®* no es la excepción, ***ToroiseSVN*** es la implementación de cliente SVN más utilizada en esta plataforma, apodado cariñosamente por los integrantes del Área de Transferencia: la tortugüita.

En las próximas sub-secciones haremos la importación (*checkout*) inicial de un repositorio y trabajaremos con él, para luego exportar (*commit*) los cambios realizados, no contemplaremos la instalación del software *TortoiseSVN* ya que lo consideramos un paso trivial que está fuera del alcance del presente documento.

4.2 Checkout: importar un repositorio completo para generar una copia de trabajo local.

- Creamos un directorio donde guardaremos todas las copias de trabajo locales, podemos llamarlo “repos” por ejemplo, dentro de nuestro nuevo directorio “repos” hacemos click con el botón derecho y elegimos “SVN Checkout...”

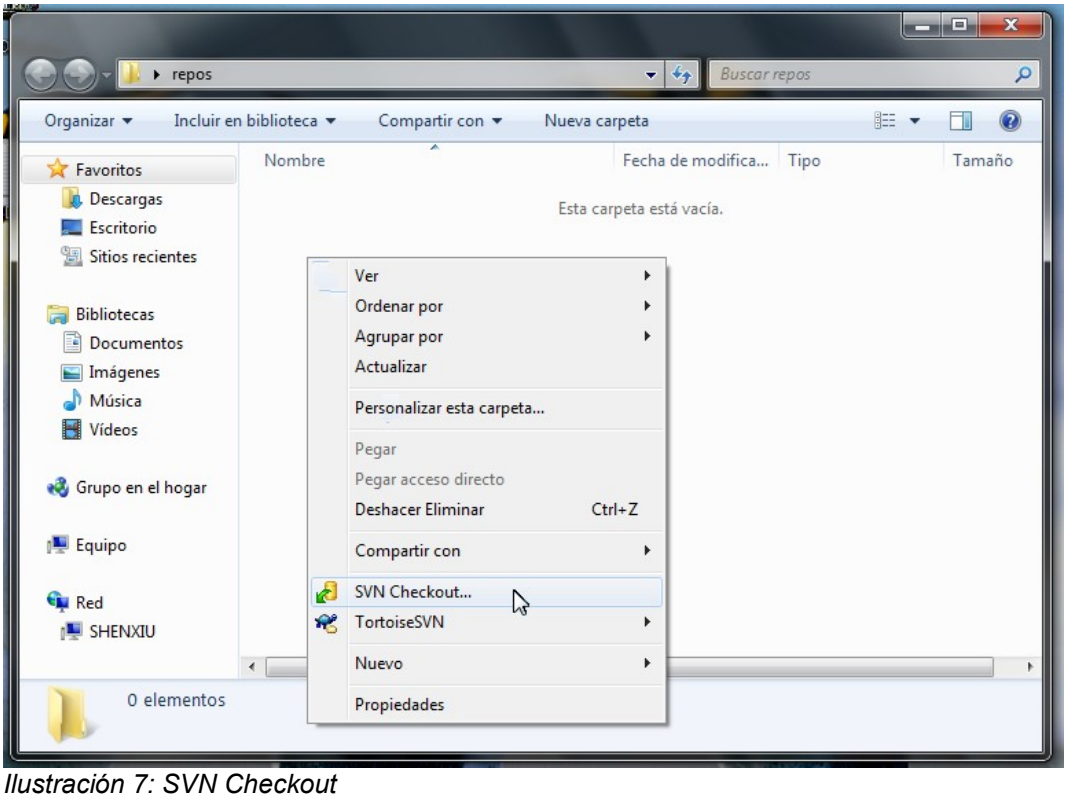

Ilustración 7: SVN Checkout

- Se nos presentará entonces el cuadro de diálogo de Checkout de repositorios en el cual completaremos sólo los campos:
  - *URL of the repository*: es la URL del repositorio que queremos hacer checkout,

en éste caso será el repositorio de proyectos edilicios del IAR y su URL es: http://www.iar.unlp.edu.ar/svn/obs/EDL

- *Checkout directory:* el *path* completo en el cual se guardará la copia local.

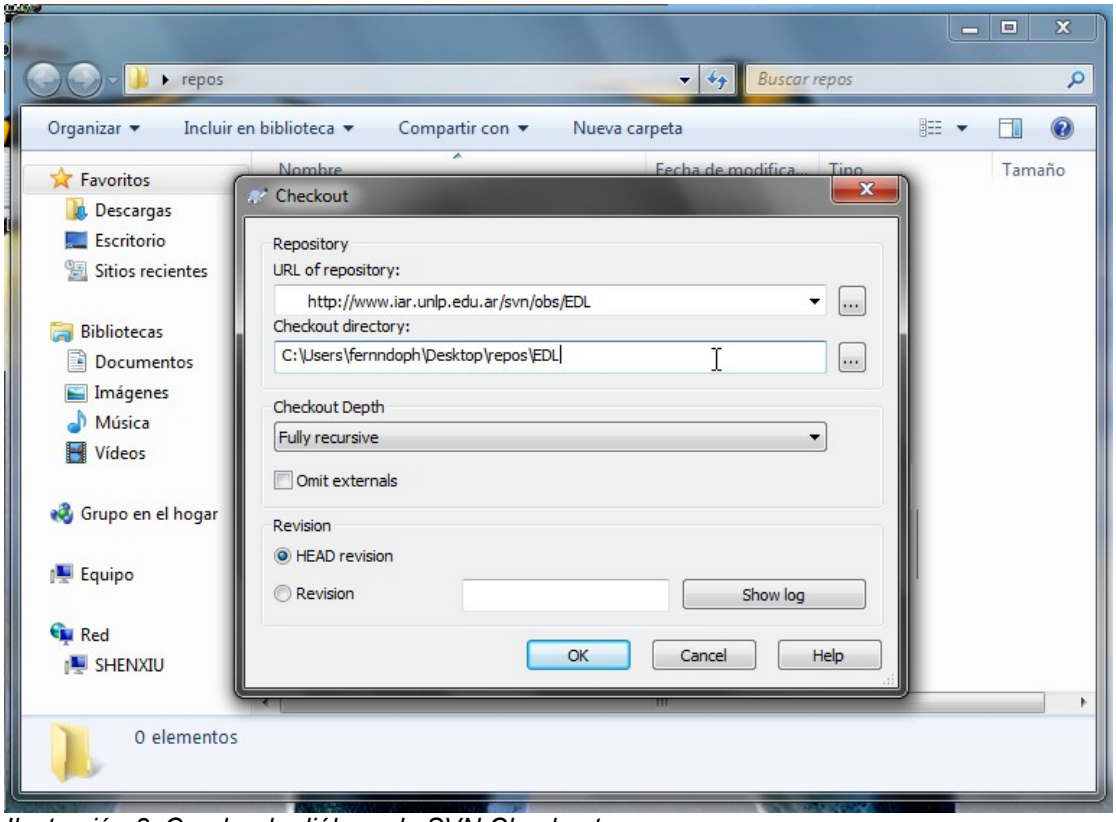

Ilustración 8: Cuadro de diálogo de SVN Checkout

- Al hacer click en el botón “OK” del cuadro anterior (Ilustración 8: Cuadro de diálogo de SVN Checkout), se abrirá una nueva ventana (Ilustración 9: SVN Checkout pregunta usuario y contraseña) en la cual tendremos que tipear nuestro nombre de usuario y contraseña, podemos hacer click en la casilla de veridicación “*Save authentication”* para que el cliente *TortoiseSVN* recuerde los datos de autenticación y no los pregunte cada vez que contactamos el servidor SVN.

Ilustración 9: SVN Checkout pregunta usuario y contraseña

- Hacemos click en “OK” y veremos como comenzará el proceso de *Checkout* del repositorio a nuestra copia local:

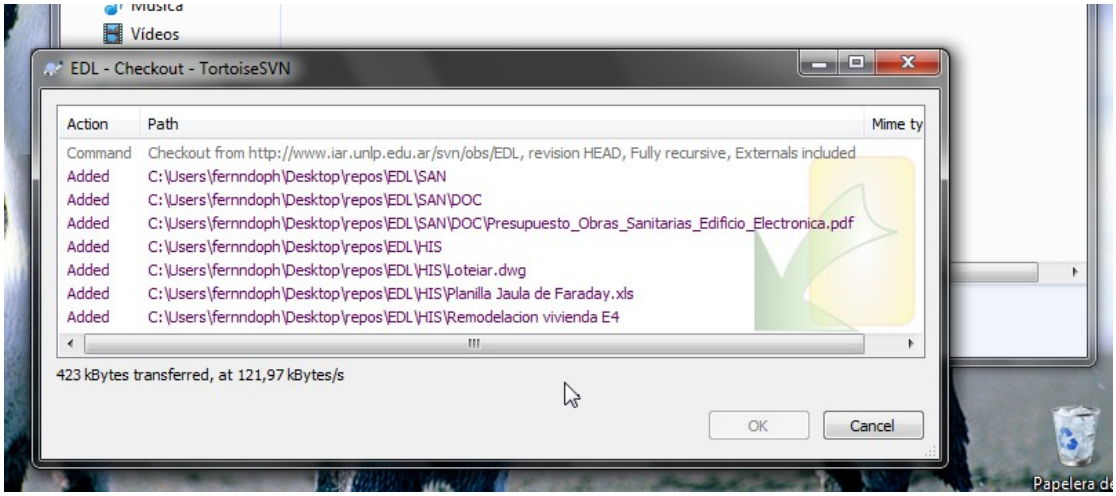

Ilustración 10: Checking out...

- El proceso de *Checkout* puede tardar unos segundos o varios minutos ya que depende de: qué tan grande es el repositorio y qué tan rápida es nuestra conexión a la red. Una vez finalizado el proceso de *Checkout*, se presentará un pequeño informe de la tarea realizada:

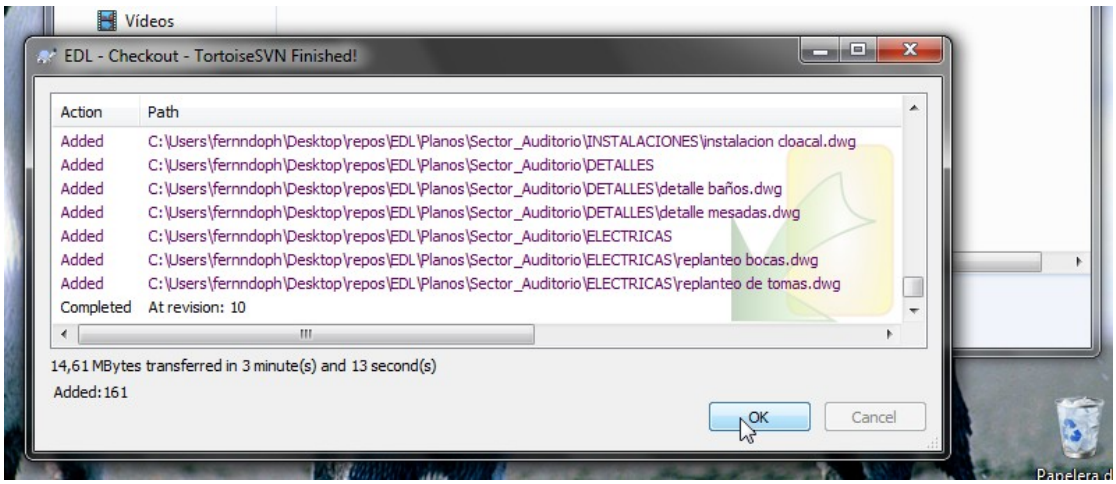

Ilustración 11: Checkout terminado

- Podemos ver en Ilustración 11: Checkout terminado, que hemos hecho el Checkout de la reivisión 10 del repositorio EDL el cual tiene un tamaño de casi 15 MegaBytes y posee 161 archivos, para completar la operación tardó 3 minutos.
- Finalmente hacemos click en “OK” para cerrar esta ventana y veremos que ahora el ícono del directorio EDL tiene una marca verde, ésto quiere decir que el repositorio está debidamente actualizado respecto del servidor:

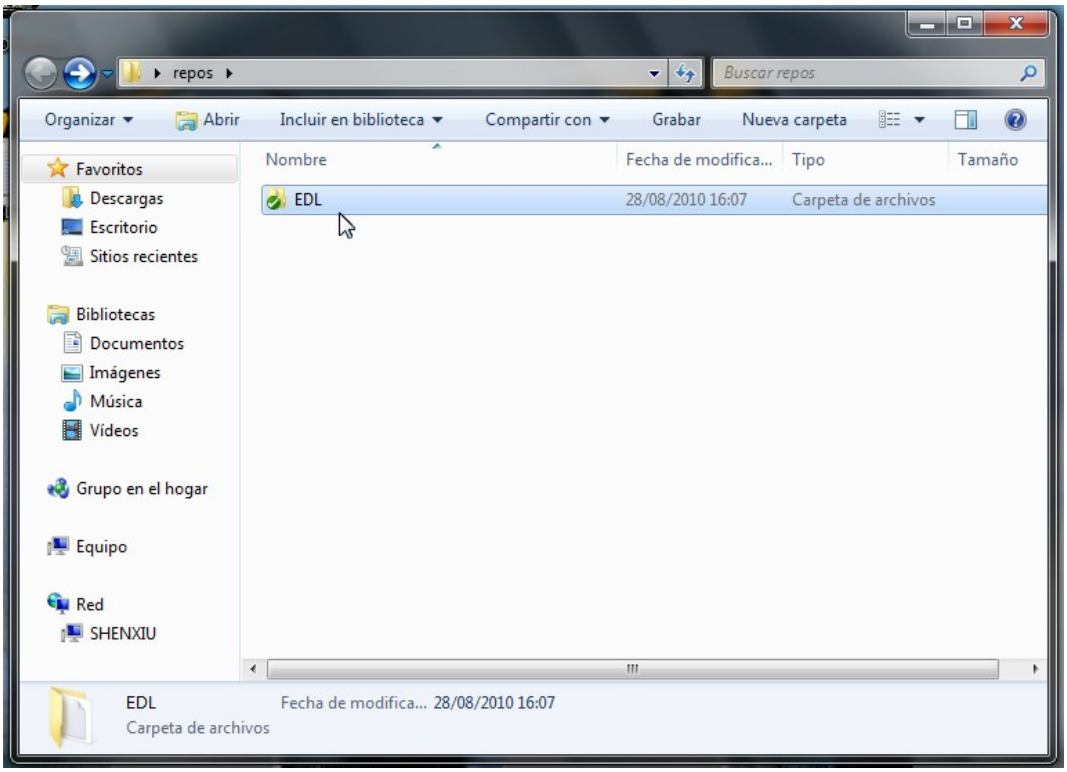

Ilustración 12: Repositorio correctamente actualizado

4.3 La copia de trabajo local

Hemos hecho el *Checkout* inicial del repositorio y de esta manera hemos creado nuestra *copia de trabajo local* en inglés: *Working copy.* La copia de trabajo local de un repositorio no es muy diferente a cualquier arbol de directorios; podemos copiar, mover, modificar, borrar y realizar todas las operaciones que usualmente se hacen con los archivos de un directorio. Los cambios realizados en los directorios y archivos de nuestra copia de trabajo local no se verán reflejados en el repositorio hasta que el usuario indique mediante un comando que así sea.

4.4 Commit: Modificación y publicación de los cambios de un archivo.

Ahora bien, realizaremos un cambio menor en un documento aprovechando un error que cometí al generar dicho documento: olvidé colocar la fecha correctamente en la tabla de revisiones. Para ésto entonces vamos al directorio REU y abrimos el archivo *OBS-EDL-00102-MI-A00-19-AGO-2010 Auditorio-SConferencias-GSanitario.odt,* una vez hechas las modificaciones necesarias, veremos que al cerrar el archivo el ícono del archivo que editamos recién ha cambiado a color rojo.

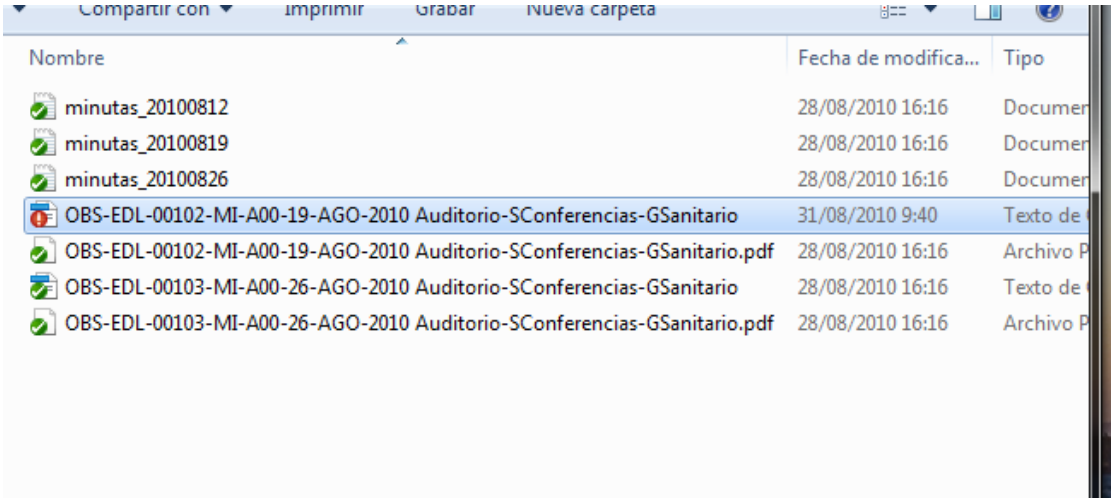

Ilustración 13: Copia de trabajo local no sincronizada con el repositorio.

Como se dijo en 4.3 La copia de trabajo local, la copia de trabajo local permanecerá en éste estado (descronización respecto del repositorio) hasta que el usuario mediante el comando *commit* decida publicar el cambio realizado.

Nos disponemos entonces a publicar el cambio: haciendo click con el botón derecho y seleccionando *“SVN Commit...”.*

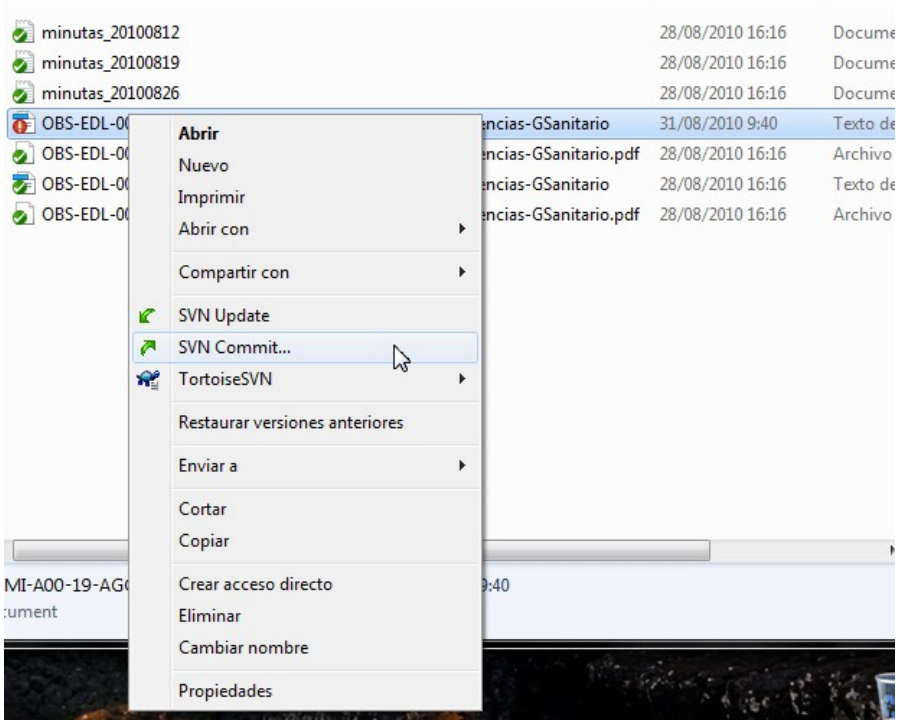

Ilustración 14: SVN Commit

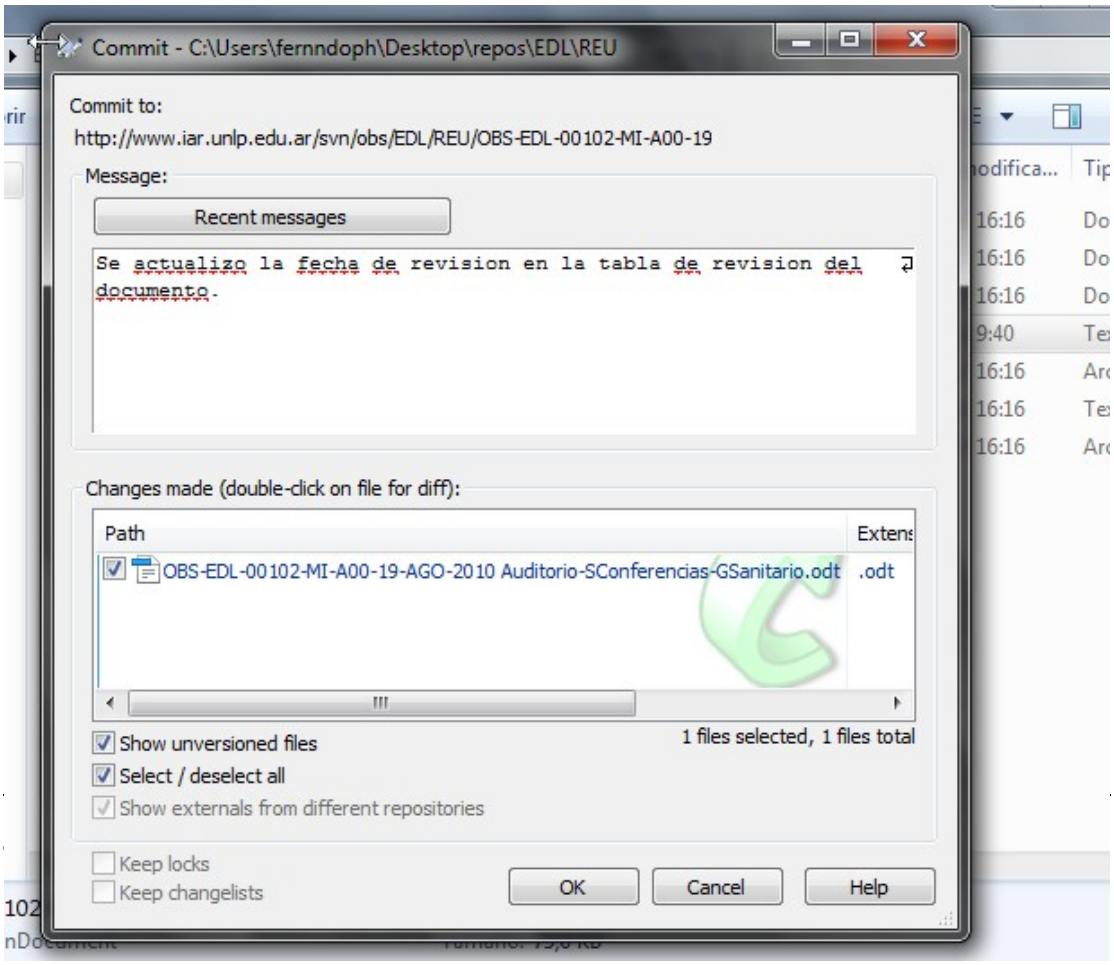

Ilustración 15: Cuadro de diálogo, SVN Commit

La Ilustración 15: Cuadro de diálogo, SVN Commit nos permite verificar los cambios hechos en la copia de trabajo local antes de publicar los cambios en el repositorio. Vemos entonces en la parte inferior de la imagen *“Chages made”* veremos los archivos que fueron modificados. Ahora, en la parte superior *“Message”* se completará con una breve explicación de lo realizado. El mensaje debe ser breve pero muy explícito sobre el cambio realizado ya que permitirá mantener un historial de cambios claro conforme evoluciona el repositorio con el contenido de los usuarios. Cuando hemos completado el campo *“Message”*, haremos clicke en *“Ok”* y el proceso de publicación comenzará, finalizando con la siguiente imágen:

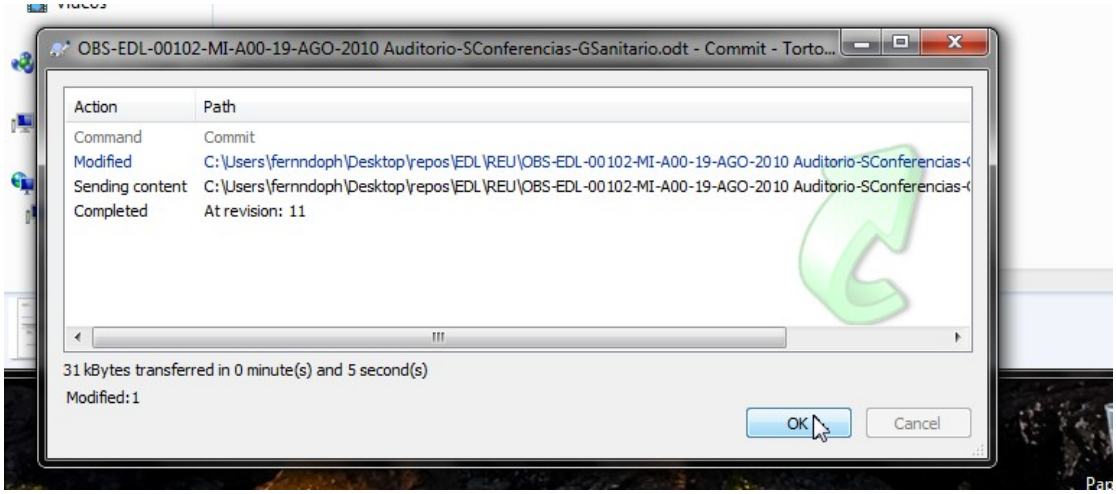

Ilustración 16: Commit realizado

La breve bitácora presentada en la imágen anterior nos permite verificar los cambios que han sido publicados en el repositorio. Hacemos click en OK y veremos como ahora el color del ícono del documento modificado, vuelve a ser verde, con lo cual podemos verificar que nuestra copia local está sincronizada respecto del repositorio.

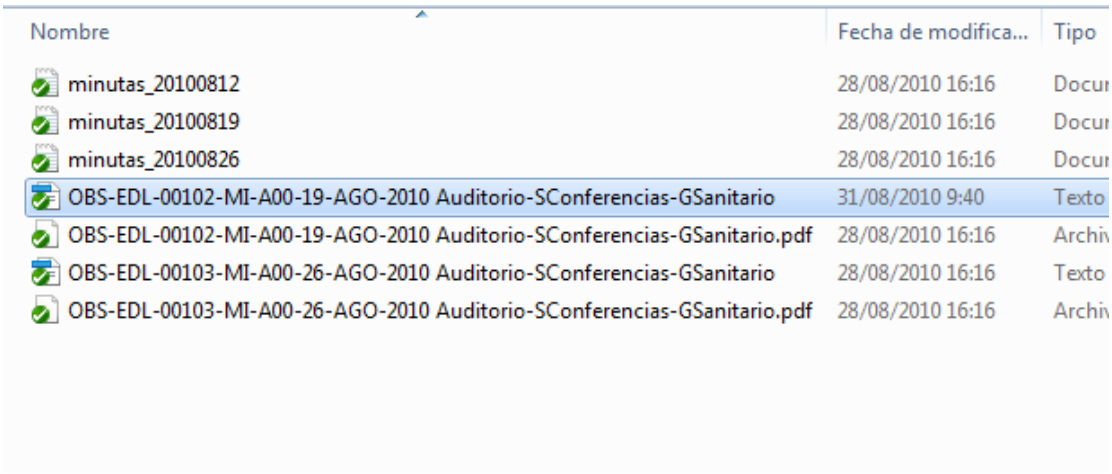

Ilustración 17: Sincronizado

5 Conclusión

Se presentó brevemente el sistema de control de versión *Subversión* así como también la interacción con los repositorios para los proyectos del IAR.

6 Referencias

[1] http://svnbook.red-bean.com/
[2] http://svnbook.red-bean.com/nightly/es/
[3] http://svnbook.red-bean.com/nightly/en/svn-book.pdf
[4] http://svnbook.red-bean.com/nightly/es/svn-book.pdf

[5] http://tortoisesvn.tigris.org/
[6] http://tortoisesvn.net/downloads

7 Control de cambios

Versión	Autor	Descripción
20100609	FPH	Versión inicial del documento