Propuesta normalización repositorio subversión

Identificación

Proyecto	Normalización del Repositorio Subversión
Nombre del documento	Propuesta Normalización Repositorio Subversión
Autor	Francisco García Mateo
Versión Actual	1.0
Fecha de la versión	21/06/2007

Versiones

Versión	Fecha	Autor	Descripción
1.0	21/06/2007	Francisco García Mateo	Versión Inicial

Tabla de contenidos

1.- Introducción

Actualmente no existe una estructura definida para los repositorios que almacenan aplicaciones web. Cada estructura es definida por cada uno de los equipos de trabajo y no sigue ninguna normativa concreta. Esto origina una serie de problemas que se pueden concretar en los siguientes:

1. No existe una mínima gestión de la configuración común a todas las aplicaciones de ÁTICA
2. Los fuentes muchas veces se encuentran en el PC del programador, sin copias de seguridad y sin un correcto etiquetado y localización.
3. Es imposible crear un procedimiento medianamente automatizado para poner en línea las aplicaciones WEB.

Por ello desde la sección de sistemas se ha propuesto el incluir en un repositorio con una estructura fija todas las aplicaciones Web de manera que este repositorio permita un mínimo control de versiones y el poner en línea de manera rápida y automática las nuevas versiones de las aplicaciones.

1.1.- Objetivos

Los objetivos a conseguir entonces son los siguientes:

- Definir una estructura de directorios común a todas las aplicaciones.
- Definir criterios acerca de como utilizar la estructura de directorios.
- Conseguir una gestión de la configuración para las aplicaciones Web.
- Permitir poner en línea automáticamente las aplicaciones web a partir de la estructura de directorios propuesta.

Para conseguir estos objetivos las tareas a realizar son:

1. Identificar los componentes que han de incluirse en el repositorio.
2. Establecer criterios para identificar a los responsables de subir los artefactos al directorio, en qué momento y bajo qué condiciones.
3. Establecer la estructura del repositorio para cada aplicación.
4. Establecer procedimientos y proporcionar instrucciones de uso del repositorio.

1.2.- Ámbito

El presente documento se refiere a aplicaciones Web desarrolladas en ÁTICA.

1.3.- Referencias

Desde la sección de sistemas se hizo una primera propuesta que puede encontrarse en ./documentos de trabajo/Propuesta_Subversion_Desarrollo.odt.

Además se proporcionó un ejemplo de directorio que puede encontrarse en ./documentos de trabajo/estructura-svn.odt

Se tendrá en cuenta lo ya realizado relativo a normalización de documentación en el proyecto Normadoc, concretamente el documento organización de la “documentacion antiguav2.pdf”

Por último también se tendrá en cuenta lo desarrollado hasta este momento en el proyecto AUPA.

Se ha consultado el capítulo referido a la Process Area *Configuration Management* del libro “CMMI Guidelines for Process Integration and Product Improvement. Mary Beth Chrissis y otros. Ed. Addison-Wesley. ISBN 0321154967”.

Se ha consultado el capítulo referido a la disciplina *Configuration and Change Management* del libro “The Rational Unified Process. An Introduction. Third Edition. Philip Kruchten. Ed. Addison-Wesley. ISBN 0321197704”.

Se ha consultado el capítulo referido a la *Gestión del cambio* del libro “Ingeniería del software. Un enfoque práctico. Sexta Edición. Roger S. Pressman. Ed. McGrawHill. ISBN 0072853182”

Para la sección de Procedimientos e instrucciones se ha consultado el libro electrónico “Version Control with Subversion” que se puede encontrar en http://svnbook.red-bean.com/en/1.2/svn-book.pdf.

2.- Componentes a incluir en el repositorio

Para seleccionar los componentes que debemos incluir en un repositorio seguiremos lo recogido en la disciplina o Process Area, según la metodología utilizada, de gestión de la configuración. Según esta disciplina los distintos items de configuración que han de verse recogidos en el repositorio tenemos:

- Productos que han de entregarse al cliente. Esto incluiría lo siguiente:
  - Fuentes
  - Ejecutables
  - Manuales de instalación, de mantenimiento, funcionamiento, etc.
- Productos de trabajo interno relevantes para construir la solución final. Esto incluye:
  - Descripción de procesos
  - Planificación del proyecto
  - Artefactos de requisitos
  - Artefactos de análisis y diseño
  - Planes de test, test y resultados de los test
  - Descripción de interfaces con otros productos
- Productos externos adquiridos
- Herramientas. Cuando se utilicen herramientas es importante almacenar al menos la versión de cada herramienta utilizada. A veces la utilización de una versión distinta para crear un determinado artefacto puede generar problemas. Esto incluye compiladores, IDE's, versiones de bases de datos, etc.

3.- Criterios de uso del repositorio

Un sistema de gestión de la configuración no incluye únicamente un sistema de almacenamiento de artefactos y una herramienta para manejar dicho sistema, sino que ha de incluir procedimientos de manejo de dicho sistema que indique quien o quienes pueden acceder al repositorio y de qué manera y qué tratamiento ha de sufrir cada uno de los artefactos que son subidos al repositorio.

Hemos de tener claro en primer lugar los distintos niveles de control que necesitamos:

- Durante el comienzo del proyecto, cuando todavía no hay un producto funcionando, ni siquiera una primera versión el control sobre los artefactos sometidos a gestión de la configuración puede ser más relajado. Los distintos componentes del equipo de trabajo pueden trabajar con mayor libertad con el repositorio.

- Una vez que el producto comienza a funcionar, aunque sea una primera versión, o se establece el primer baseline, únicamente el jefe de proyecto o gestor de la configuración debería realizar cambios, previamente evaluados y aprobados formalmente, en los items de configuración del proyecto.

- Es altamente recomendable que cada participante en el proyecto tenga un repositorio privado donde pueda almacenar su trabajo diario. Esto permite conseguir dos objetivos a la vez; que los miembros del equipo tengan un lugar seguro donde almacenar su trabajo diario y puedan dar marcha atrás en caso de necesidad y por otra parte se acostumbren a realizar una gestión de la configuración de manera habitual.

Todos los miembros del equipo han de disponer de acceso libre a todos los artefactos que correspondan a los roles que desempeñen al menos en lectura, pero únicamente las personas autorizadas deben ser capaces de subir los distintos items de configuración al repositorio.

- Los miembros del equipo podrán bajar libremente, y deberán hacerlo antes de realizar cualquier modificación de artefacto, cualquier item de configuración del repositorio que sea de su competencia.

- Los miembros del equipo podrán modificarlo y almacenarlo libremente en su repositorio personal.

- Los miembros del equipo una vez modificado un artefacto y pasado los controles pertinentes lo pasarán al responsable de la gestión de la configuración.

- El gestor de la configuración subirá los items de configuración al repositorio tras realizar los controles pertinentes.

El gestor de la configuración también ha de ser el encargado de generar y recuperar versiones de la aplicación, así de decidir que versiones han de ser congeladas y/o puestas en línea.

Por último, el gestor de configuración también ha de ser el encargado de realizar informes de cada versión congelada indicando los items de configuración de la versión del programa y qué versión de cada uno forman parte de la versión de la aplicación.

4.- Estructura del repositorio para cada aplicación.

Recogiendo la información en los apartados anteriores haremos un resumen de la información que ha de recoger el repositorio.

1. Descripción de los procesos utilizados durante el desarrollo

Documentos indicando los procesos utilizados durante todo el ciclo de vida de la aplicación.

2. Planificación

Todos los documentos realizados para la planificación del proyecto han de ser recogidos, planes, actas, etc.

3. Requisitos

Cualquier artefacto de requisitos de la aplicación

4. Artefactos de análisis y diseño

Cualquier artefacto de análisis o diseño

5. Test

Planes de test, código de los test y resultados de los test.

6. Descripción de interfaces con otros productos

Si hay algún producto externo será necesario especificar claramente la interfaz entre el producto desarrollado y el producto externo.

7. Código fuente de las aplicaciones

Código fuente de todas y cada una de las versiones operativas de nuestro producto.

8. Código de productos externos adquiridos (Si corresponde)

Si existe algún producto externo y disponemos de su código debemos recogerlo en el repositorio.

9. Ejecutables de productos externos adquiridos (Si corresponde)

Los ejecutables de los productos externos también debemos incluirlos, independientemente de que dispongamos de su código.

10. Ejecutables de las distintas versiones del producto

Los ejecutables de las distintas versiones del producto no es estrictamente necesario si disponemos de los fuentes, pero en el problema concreto que nos ocupa, la necesidad de subir los ejecutables a un servidor web debemos recogerlos.

11. Manuales

Cualquier tipo de manual realizado; de instalación, de mantenimiento, de usuario, etc.

12. Herramientas utilizadas

Al menos debemos recoger la versión de todas y cada una de las herramientas utilizadas durante la construcción del producto. Esto incluye herramientas de requisitos, análisis, diseño, compiladores, IDE's, herramientas de test, etc.

Concretando todo lo anterior y suponiendo que tenemos un repositorio por cada Aplicación/proyecto proponemos la estructura de repositorios que puede verse en el Anexo I.

4.1.- Utilización de tronco, ramas y etiquetas (trunk, branches, tags)

En el repositorio deberemos utilizar los conceptos de tronco, ramas y etiquetas propio de cualquier sistema de gestión de versiones. Definiremos estos conceptos:

- Tronco o Trunk: Línea principal de desarrollo de un proyecto.
- Rama o Branch: Línea de desarrollo independiente de la línea principal, de la que se desgaja en un momento dado, pero comparte una historia común con el tronco del proyecto
- Etiqueta o Tag: Foto fija de un proyecto de desarrollo en un momento dado. Generalmente se corresponde con una versión en explotación de la aplicación desarrollada.

En nuestros proyectos, en algunos de los directorios, concretamente en src, bin, test y manuales utilizaremos estos conceptos.

- Tronco (trunk): En todos ellos tendremos un directorio llamado principal en el que se producirá la linea principal de desarrollo.
- Ramas (branch): El jefe de proyecto podrá decidir separar parte del desarrollo en una rama aparte, pero conservando la historia común del proyecto. Esto se hará, por ejemplo, cuando queremos personalizar una misma aplicación para varios clientes, el núcleo de la aplicación es común, pero habra algunas diferencias. Las ramas colgaran de src/ramas, bin/ramas, test/ramas y/o manuales/ramas.
- Etiquetas (tag): Corresponderán a versiones en explotación de la aplicación. El jefe de proyecto congelará una determinada revisión del repositorio y la etiquetará como versión.

En el apartado 6.6 veremos algunos comandos útiles para trabajar con troncos, ramas y versiones.

5.- La puesta en línea de una versión

Para poner en línea una determinada versión copiaremos el tronco o rama adecuado que cuelga del directorio .../aplicacion_proyecto/proyecto/bin a .../aplicacion_proyecto/web_nombre_de_servidor.

Por supuesto que la estructura en los troncos y ramas que hay bajo /bin han de tener una estructura J2EE.

En las secciones siguientes se dan instrucciones acerca de los comandos para utilizar el repositorio y cómo ha de hacerse esta copia dentro del repositorio con svn copy.

En la sección 7 se explicará cómo hacer exactamente cada una de las tareas para poner en línea las distintas versiones y cómo resolver cuestiones relativas al trabajo diario.

6.- Procedimientos e instrucciones de uso del repositorio

En este apartado veremos algunas de las instrucciones necesarias para manejar el repositorio y algunos ejemplos de comandos y modos de utilizarlo.

6.1.- Iniciando el trabajo

6.1.1.- Creando un directorio de trabajo en local

Un directorio de trabajo es un directorio absolutamente normal en tu máquina local en el que se ha copiado los ficheros que hay en un directorio del repositorio.

Suponiendo que dir_local es el directorio local y dir_repos es el directorio en el repositorio, crearemos una copia de trabajo en dir_local con el siguiente comando

```
C:\...\dir_local>svn checkout https://svn.atica.um.es/.../dir_repos
```

6.1.2.- Actualizando la copia de trabajo

Si disponemos de una copia de trabajo y vamos a comenzar a trabajar con un fichero determinado y queremos asegurarnos que estamos utilizando la última versión del fichero que hay en el repositorio deberemos actualizar la copia de trabajo con el comando.

```
C:\...\dir_local>svn update https://svn.atica.um.es/..../dir_repos
```

6.2.- Haciendo cambios en los ficheros y el directorio de trabajo

6.2.1.- Añadiendo ficheros al control de versiones

Al un nuevo fichero en el directorio de trabajo no está sometido a control de versiones. Para que sea sometido al control de versiones hay que indicarle expresamente a subversión que el fichero ha de ser añadido al control de versiones. Esto se hace con el siguiente comando.

```
C:\...\dir_local> svn add fichero_local
```

Si lo que añadimos es un directorio se marcará para añadir todo lo que haya en él.

6.2.2.- Eliminando ficheros del control de versiones

Si deseamos eliminar un fichero del control de versiones lo haremos con el comando

```
C:\...\dir_local> svn del fichero_local
```

Si es un fichero se borrará inmediatamente del directorio de trabajo local.[1]

6.2.3.- Copiando ficheros bajo control de versiones

Podemos copiar ficheros sometidos al control de versiones manteniendo la historia de las modificaciones sufridas por el fichero. Esto se hace con el comando

```
C:\...\dir_local> svn copy fichero_local1 fichero_local2
```

Si estamos interesado en copiar el fichero y someter la copia a control de versiones, pero no estamos interesado en mantener el historial de revisiones, simplemente haremos una copia en el S.O. de que se trate y añadiremos el nuevo fichero al control de versiones con `svn add`.

6.2.4.- Moviendo ficheros bajo control de versiones

Podemos mover un fichero sometido a control de versiones de sitio o cambiarle el nombre sin perder su historia de modificaciones. Esto se hace con el comando

```
C:\...\dir_local> svn move fichero_local1 fichero_local2
```

[1] Por supuesto que nada es borrado totalmente del repositorio. Siempre podemos consultar el repositorio y obtener versiones de los ficheros anteriores a la HEAD (la última) y podremos recuperar el fichero borrado.

Esto es equivalente a hacer

```
C:\...\dir_local> svn copy fichero_local1 fichero_local2
C:\...\dir_local> svn del fichero_local1
```

6.2.5.- Comiteando los cambios en el repositorio

Una vez realizado los cambios pertinentes en el repositorio o los ficheros de trabajo y añadidos o eliminados los ficheros del control de versiones hemos de hacer efectivos los cambios en el repositorio. Esto se hace mediante el comando

```
C:\...\dir_local> svn commit –message “Mensaje que quedará registrado en
el repositorio acerca del motivo de los cambios realizados”
```

6.3.- Comprobando el estado de la copia de trabajo

Antes de hacer un commit de los cambios en el repositorio es buena idea comprobar que es lo que se ha cambiado realmente. En las siguiente secciones veremos cómo trabajar con ficheros que se han cambiado y cómo trabajar con las distintas versiones de un mismo fichero.

6.3.1.- El estatus de la copia de trabajo

Para ver los cambios que hemos hecho en un directorio de trabajo ejecutaremos el comando

```
C:\...\dir_local>svn status
```

La información que proporciona este comando es algo así como

```
M         bar.c
?      foo.o
A  +      dir1
```

Cada fila consiste en cinco columnas con alguna información, unos espacios en blanco y el nombre de un fichero o directorio. Veremos a continuación la información más habitual que puede aparecer en cada una de las columnas.

Primera columna

Carácter	Significado
A	Ha sido encolado para añadir al repositorio
C	En estado de conflicto. Existe solapamiento de modificaciones entre la versión local y la del repositorio
D	Ha sido encolado para borrar del repositorio
M	El contenido del fichero ha sido modificado
?	El fichero o directorio no está bajo control de versiones
!	El fichero o directorio está bajo control de versiones pero se ha perdido, quizás ha sido borrado fuera del control de versiones por un comando del sistema operativo.
~	Hay un cambio de tipo en el archivo. Quizás borramos un fichero y creamos un directorio con el mismo nombre pero sin usar los comandos svn

Tercera columna

Aparece un blanco o L porque el fichero está bloqueado en el directorio .svn. Quizás está a medio realizarse un commit o el commit ha sido interrunpido a medias y es necesario desbloquear el fichero con el comando svn cleanup.

Cuarta columna

Blanco o +. Cuando el fichero está encolado para añadirse o modificarse con información adicional. Típicamente cuando se hace un svn copy o svn move. El fichero se traslada, pero con toda la historia adicional que lleva consigo.

Si añadimos en el comando la opción -u me mostrará dos columnas más; un '*' en aquellos ficheros/directorios que haya una versión más actual en el servidor que la del directorio de trabajo y el número de la revisión que tengo en el directorio de trabajo.

6.3.2.- Examinando las diferencias entre ficheros

En cualquier momento podemos ver las diferencias existentes entre los ficheros del repositorio y los modificados en la copia local. Esto se realiza mediante el comando

```
C:\...\dir_local> svn diff
```

El formato de la salida es igual al del comando diff de Unix.

6.3.3.- Volviendo a versiones anteriores de los ficheros

Podemos dar marcha atrás en los cambios hechos sobre los ficheros del directorio de trabajo. Como Subversion guarda una copia en local de los ficheros antes de ser modificados, es posible volver a su estado original con

```
C:\...\dir_local> svn revert
```

Es importante saber que esto se refiere únicamente a las modificaciones sufridas en local, pero que podemos recuperar cualquier revisión de cualquier fichero con el comando

```
C:\...\dir_local> svn update -r num_version
```

6.3.4.- Resolviendo conflictos

Es posible que al intentar subir un fichero al repositorio con svn commit alguien ya haya subido este fichero y haya hecho cambios incompatibles en él, por ejemplo haciendo cambios en las mismas líneas. En ese caso el comando da un mensaje de error y no sube el fichero modificado.

Para solucionar este conflicto ejecutaremos el comando

```
C:\...\dir_local> svn update fichero_conflictivo.txt
```

Subversion hara lo siguiente en nuestra copia de trabajo.

1. Modificará fichero_conflictivo.txt introduciendo marcadores para identificar las filas modificadas por mí y por la otra persona que entran en conflicto. Así sabré que ha modificado cada uno.
2. Creará un fichero_conflictivo.mine con el fichero tal y como quedo tras mis modificaciones.
3. Creará un fichero_conflictivo.rx (x es el número de versión del último checkout) con el fichero como estaba en su estado original en mi directorio de trabajo; es decir antes de que yo realizase ninguna modificación sobre él, tal y como estaba en el último checkout.
4. Creará un fichero_conflictivo.ry (y es el número de versión del repositorio) con el fichero como estaba en el repositorio tras las modificaciones de la otra persona.

Examinando estos ficheros tendremos que dejar el fichero_conflictivo.txt tal y como queramos que quede en el repositorio. Esto puede hacerse modificando fichero_conflictivo.txt a mano, con alguna herramienta o copiando cualquiera de los otros tres ficheros que se crearon sobre él. Una vez que fichero_conflictivo.txt está tal y como queremos que quede en el repositorio ejecutaremos el comando

```
C:\...\dir_local> svn resolved fichero_conflictivo.txt
```

Este comando elimina los ficheros .mine, .rx y .ry y resuelve el conflicto. Ahora ya podemos hacer snv commit sin ningún problema.

6.4.- Examinando la historia del repositorio

6.4.1.- Comprobar la historia de las revisiones

Podemos comprobar cada revisión, quién la subió al repositorio, cuando, que mensaje utilizó en el commit y cuales fueron los ficheros afectados. Todo esto se hace con el comando

```
C:\...\dir_local> svn log -r num_version -v
```

-r especifica la versión de la que queremos obtener información; también podremos ver información en un determinado rango de versiones con -r

version1:version2. Si lo omitimos dará información sobre todas. El comando -v da información sobre todos los ficheros/directorios implicados en el svn commit. Si no lo especificamos nos dará información únicamente sobre el mensaje enviado en el svn commit.

6.4.2.- Ver las diferencias entre versiones del repositorio

Podemos comparar distintas versiones de los ficheros que queramos. Para ello utilizamos el comando

```
C:\...\dir_local> svn diff fichero.txt
```

Este comando compara el fichero en local con la última versión del repositorio. Podemos comparar el fichero local con una versión antigua con

```
C:\...\dir_local> svn diff -r num_version fichero.txt
```

o comparar dos versiones distintas del repositorio

```
C:\...\dir_local> svn diff -r num_version1:num_version2 fichero.txt
```

El formato de la comparación es el del comando diff de Unix.

6.4.3.- Ver una versión antigua

Podemos ver el contenido de cualquier versión de cualquier fichero de manera muy sencilla con la sentencia

```
C:\...\dir_local> svn cat -r num_version fichero.txt
```

6.4.4.- Listar el contenido del repositorio en una determinada versión

Por último podemos ver el contenido del repositorio sin necesidad de hacer un checkout.

```
C:\...\dir_local> svn list -v https://svn.atica.um.es/.../repositorio
```

6.5.- Otras comandos útiles

6.5.1.- Eliminar bloqueos

Si un svn commit quedo interrumpido por alguna razón anómala algunos ficheros pueden quedar bloqueados. Esto se resuelve utilizando el comando

```
C:\...\dir_local> svn cleanup
```

6.5.2.- Hacer import del repositorio

Un import consiste en llevar una copia de un árbol de directorios no versionados a un repositorio. Esto se hace mediante el comando

```
C:\...\dir_local> svn import
```

6.6.- Trabajando con troncos, ramas y etiquetas

Una de las características más importantes de un repositorio es poder trabajar con ramas y etiquetas.

6.6.1.- Ramas

Consideremos que el directorio principal en el que estoy desarrollando mi proyecto es ..../principal. A partir de ahí, en un momento dado, queremos trabajar sobre una parte específica del proyecto sin interferir con el desarrollo principal, por ejemplo para adaptarlo a una nueva versión de GENTE, pero dejando el resto de la aplicación igual. En este caso es adecuado crear una nueva rama en el directorio correspondiente, por ejemplo .../src/ramas/rama1.

Para crear una nueva rama utilizaremos el comando

```
C:\...\dir_local>svn copy https://svn.atica.um.es/.../src/principal
https://svn.atica.um.es/.../src/ramas/rama1 -m “Creando una rama privada
de principal”
```

A partir de este momento principal y la rama1 caminan por caminos separados, pero comparten una historia común y más adelante podrán unirse de nuevo.

Si utilizamos el comando svn log podemos ver la historia de los ficheros de la rama y del principal y ver en qué revisión del repositorio se separaron ambas.

6.6.2.- Unificando las ramas y el tronco principal

Una vez que hemos modificado distintos ficheros que forman parte de la rama, es normal que queramos incorporar las modificaciones al tronco principal. El comando svn merge es capaz de:

1. Ver las diferencias que hay entre distintas versiones del repositorio.
2. Incorporar esas diferencias a uno o varios archivos.

De esta manera si creamos una nueva rama en la versión x del repositorio y modificamos un archivo en la rama en la versión x+y en el repositorio, podremos incorporar estas modificaciones al tronco principal comparando las versiones x y x+y e incorporando las diferencias a los archivos del tronco principal. Es importante indicar aquí, que los cambios se aplicarán sobre archivos que estén en un directorio de trabajo en local.

Imaginemos la siguiente situación:

1. El programador A creo una rama llamada rama1 en la revisión del repositorio 341.
2. El programador A modifica fichero1.c de la rama1 en la revisión 343.
3. El programador B modifica fichero1.c de la rama principal en la revisión 344.
4. El programador A quiere incorporar la modificación que hizo B en la rama principal en la r344 a su fichero1.c tal y como lo dejo en la r343.

En este caso el comando

```
C:\...\src\ramas\rama1>svn merge -r 343:344
https://svn.atica.um.es/.../src/principal ramas/rama1
U ramas/rama1 fichero1.c
```

actualizará la copia de trabajo con los cambios habidos en la rama principal. Más adelante deberé hacer un commit si quiero incorporarlos a la rama1 en el repositorio.

La otra situación que se puede dar es querer volver a unir la rama1 con el tronco principal. Para ello debemos crear un directorio de trabajo con la situación actual del tronco principal y después aplicar a este las modificaciones realizadas a la rama1 durante toda su vida útil. Para esto último utilizaremos el comando.

```
C:\...\src\principal>svn merge -r 341:HEAD
https://svn.atica.um.es/.../src/rama/rama1
```

Donde 341 es la revisión en que se dividió la rama del tronco principal. A partir de aquí pueden darse dos situaciones.

1. Que las modificaciones de la rama no interfieran en el tronco principal: Podemos hacer un commit con toda normalidad

   ```
   C:\...\src\principal> svn commit –m “Merge de la rama1, cambios
   r341:r345 en la rama principal”
   ```

   Es importante indicar aquí a efectos de contabilidad y administración las revisiones que se están incorporando a la rama principal.

2. Que las modificaciones de la rama interfieran en el tronco principal: En este caso deberemos resolver el conflicto tal y como se indicó más arriba.

6.6.3.- Etiquetas

No existe ninguna diferencia de trabajo entre una rama y una etiqueta, pero sí conceptual. Una etiqueta debe asimilarse a una release de la aplicación se realiza con el comando svn copy igual que las ramas. La única diferencia es que se supone que las release que se someten a explotación no deben ser modificadas nunca, aunque la política de gestión de las etiquetas será creada por el jefe de proyecto.

Las etiquetas se guardarán dentro del subdirectorio .../versiones/version x

Existen dos maneras de crear una etiqueta:

1. Copiando un directorio del repositorio:

```
C:\...\dir_local>svn copy https://svn.atica.um.es/.../src/principal
https://svn.atica.um.es/.../src/version/version1 -m “Tagging:
Creando la version1 del proyecto.”
```

2. Creando una rama a partir de un directorio de trabajo: Esta opción nos permite crear versiones más complejas a las que añadir archivos adicionales, además de los que haya en el repositorio.

```
C:\...\dir_local>svn copy dir_trabajo
https://svn.atica.um.es/.../src/version/version1 -m “Tagging:
Creando la version1 del proyecto.”
```

7.- El proceso de gestión de la configuración

En el anexo II se puede ver cual sería el proceso de gestión de la configuración para el desarrollo de una aplicación para ÁTICA. En esta sección resolveremos las principales cuestiones que le puede surgir a cualquier persona involucrada con la configuración en un proyecto software.

7.1.- ¿Por donde empiezo?

1. Solicitamos a sistemas la creación de un repositorio indicando
    - el nombre
    - si se trata de una aplicación web o c/s.
    - Usuarios que tendrán acceso a la aplicación y permisos.
2. Sistemas proporcionará una dirección del tipo https://repositorio/aplicacion_proyecto con la estructura adecuada y los permisos correspondientes. Además enviará un correo con la siguiente información:
    - La url de checkout
    - La estructura del repositorio
    - Las ACLS
    - Datos web, urls de la aplicación, como reiniciar la aplicación, etc.

7.2.- ¿Cómo creo mi directorio de trabajo?

Cliente svn

1. Situándome en el directorio padre donde quiero que se cree el directorio de trabajo haré un checkout del repositorio

```
$>svn checkout https://repositorio/aplicacion_proyecto
```

2. Puedo ver información sobre el directorio de trabajo con...

```
$>cd aplicacion_proyecto –me situo en el directorio de trabajo.
$>svn info
```

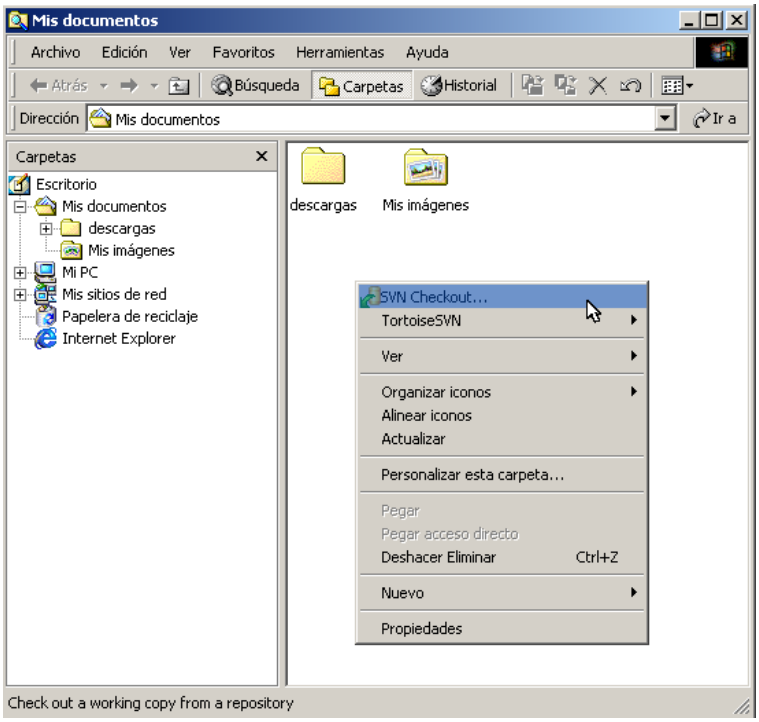
```
Path: .
URL: https://svn.atica.um.es/svn/MNCS/proyectos/aplicacion_proyecto
Repository UUID: 2939c2a9-58dc-0310-b473-f14a6506fda9
Revision: 270
Node Kind: directory
Schedule: normal
Last Changed Author: pacom
Last Changed Rev: 270
Last Changed Date: 2007-06-13 09:31:26 +0200 (mié, 13 jun 2007)
```

3. También puedo consultar la bitácora para una ruta

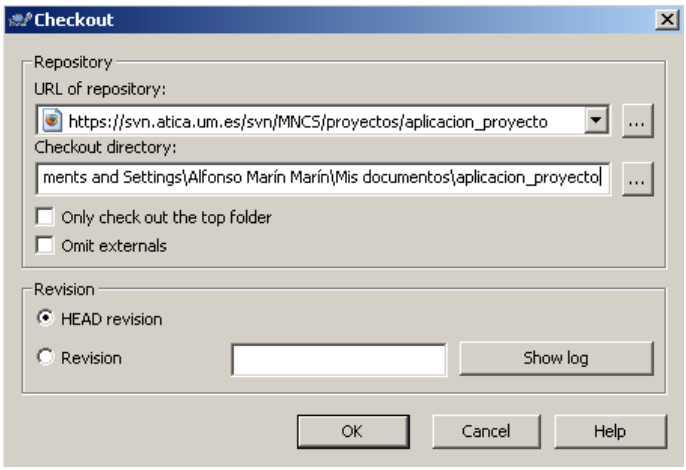
```
$>svn log
------------------------------------------------------------------------
r270 | pacom | 2007-06-13 09:31:26 +0200 (mié, 13 jun 2007) | 1 line

Creada la estructura del repositorio
```

TortoiseSvn

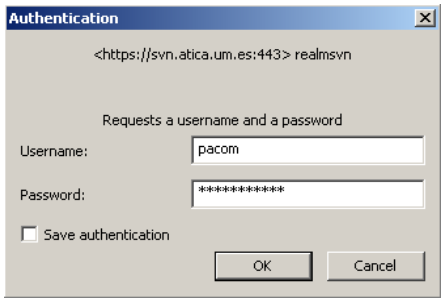
1. Situándome en el directorio padre donde quiero que se cree el directorio de trabajo haré un checkout del repositorio



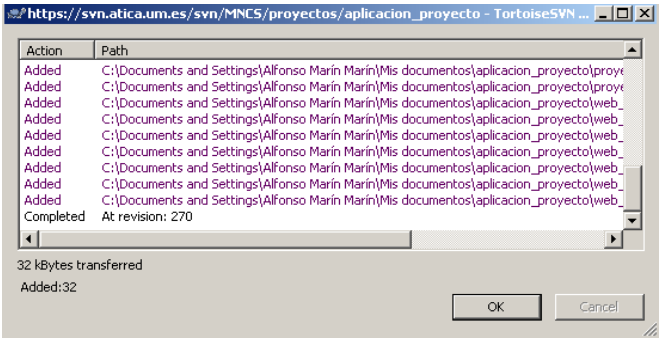
2. Indicaré la URL del repositorio y el directorio que quiero convertir en directorio de trabajo, que es en el que haré el checkout.



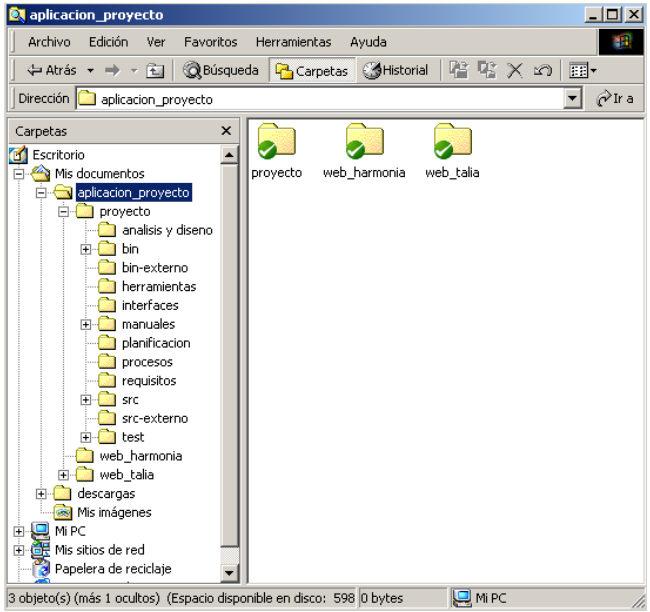
3. Me autenticaré frente al repositorio suministrando mi nombre de usuario y clave.



4. TortoiseSvn me mostrará una lista con todos los archivos y directorios que añade a mi copia local.



5. Por último, el explorador de windows me modificará los iconos de carpetas y archivos para indicarme que directorios o archivos están sincronizados con el repositorio.



7.3.- ¿Cómo bajo los ficheros de mis compañeros para actualizar mi directorio de trabajo?

Cliente svn

1. En primer lugar habré de comprobar en qué estado se encuentra mi copia de trabajo.

```
$>svn status -u
        *        proyecto/requisitos/Glosario.pdf
        *     270 proyecto/requisitos
```

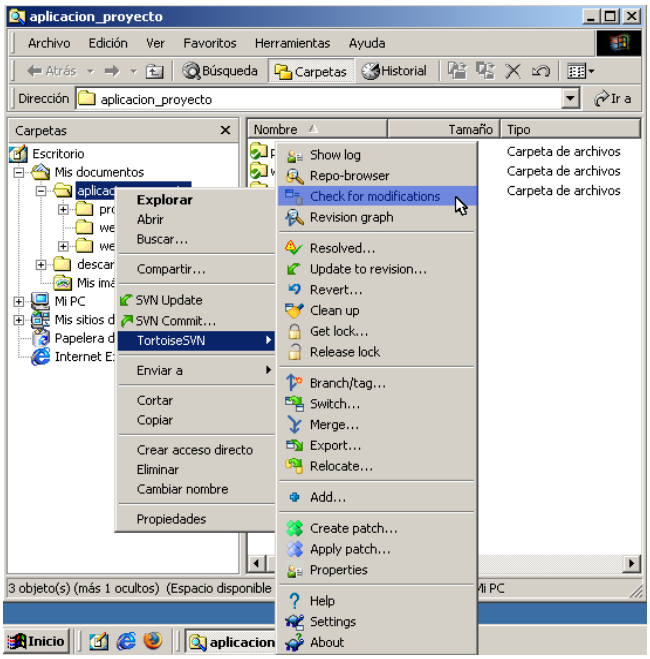
2. Podemos ver que se ha añadido Glosario.pdf al directorio proyecto/requisitos/Glosario.pdf y también que el directorio requisitos ha sido modificado y que nosotros tenemos la versión del repositorio 270.

3. En último lugar actualizaremos nuestra copia de trabajo

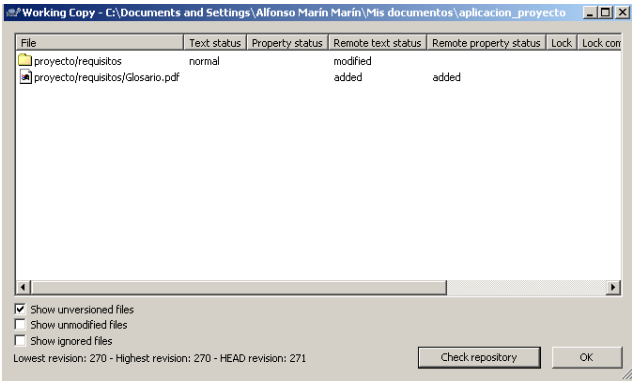
```
$>svn update
  A         proyecto/requisitos/Glosario.pdf
  Updated to revision 271.
```

TortoiseSvn

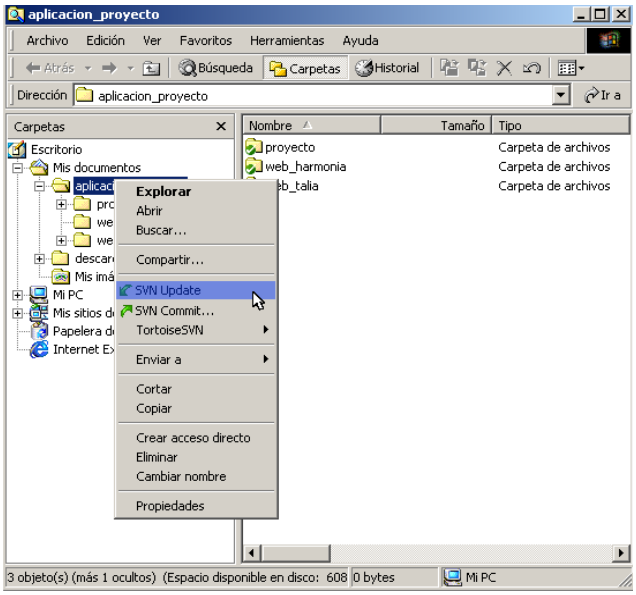
1. Pulsando con el botón derecho del ratón sobre el directorio de trabajo seleccionaré la opción TortoiseSvn -> Check for modification



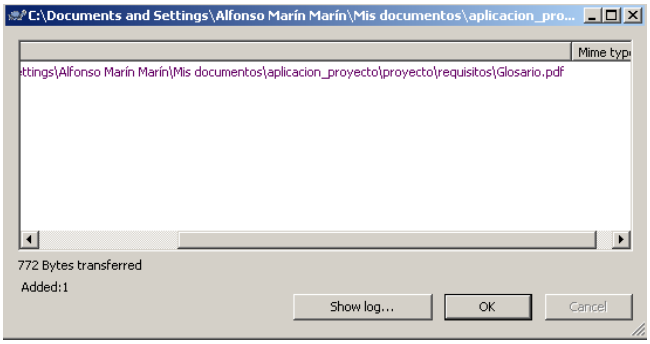
2. Tras autenticarme Tortoise me mostrará las diferencias entre el directorio de trabajo y el repositorio.



3. Con el botón derecho del ratón sobre el directorio de trabajo le indicaré que quiero actualizar el directorio de trabajo.



4. Tras autenticarme TortoiseSvn actualizará la copia de trabajo



7.4.- ¿Cómo añado ficheros en mi directorio de trabajo?

Cliente svn

En cualquier momento puedo añadir ficheros a mi directorio de trabajo como lo realizo normalmente. Para añadirlo al repositorio deberé hacerlo en dos pasos.

1. Añadirlos o eliminarlos del control de versiones. Con svn status veré si los ficheros están sometidos al control de versiones.

```
$>svn status
   ?      proyecto/requisitos/vision.pdf
```

2. La interrogación me indica que el fichero ha sido añadido al directorio de trabajo, pero no está sometido al control de versiones. Para ello utilizaré el comando svn add

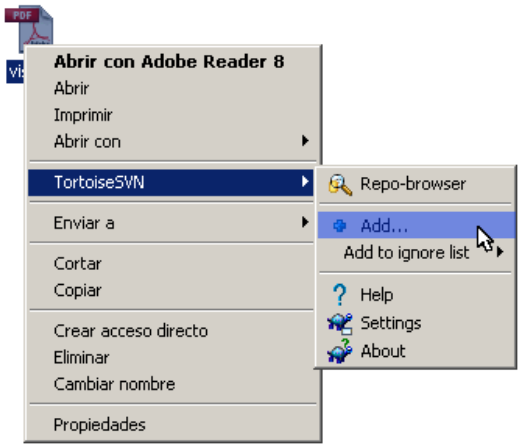
```
$>svn add proyecto/requisitos/vision.pdf
   A  (bin)  proyecto/requisitos/vision.pdf
```

3. La A me indica que el fichero ha sido añadido al control de versiones.

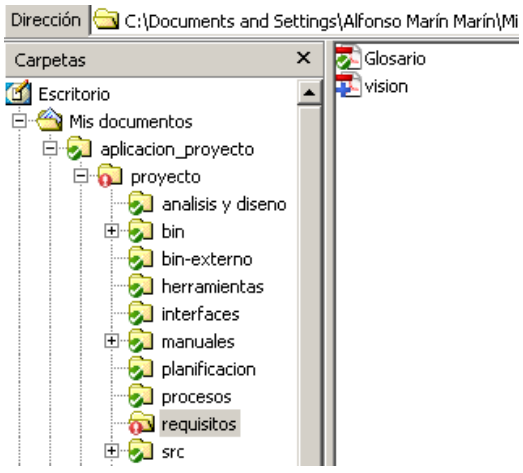
TortoiseSvn

En el explorador de windows podemos ver que el archivo vision.pdf no está sometido al control de versiones porque no está modificado su icono. Para añadirlo al repositorio deberemos hacer lo siguiente.

1. Con el botón derecho del ratón sobre el documento a añadir seleccionar TortoiseSvn -> Add



2. TortoiseSvn nos informa que ha añadido el archivo al control de versiones y el explorador de windows cambiará los iconos de todas las carpetas que contienen al archivo hasta el directorio de trabajo y el propio icono del archivo para indicar que repositorio y directorio de trabajo no están sincronizados.



7.5.- ¿Cómo elimino ficheros del directorio de trabajo?

Cliente svn

No podemos eliminar ficheros o directorios con los comandos normales del sistema operativo, ya que puede dar problemas en la sincronización entre el directorio de trabajo y el repositorio.

Para no crear este problema utilizamos el comando svn del

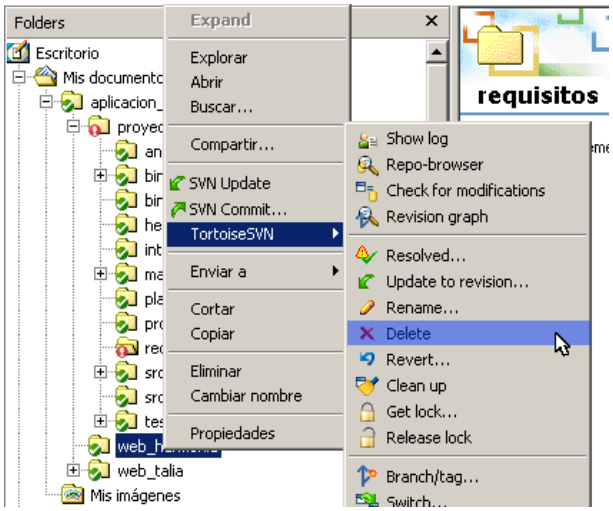
```
$>svn del web_harmonia
D         web_harmonia
```

Si hacemos un svn status -u para ver la situación en que está nuestro repositorio, podemos ver que se ha borrado el subdirectorio.

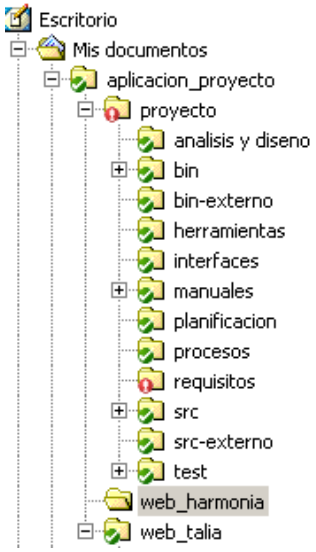
```
$>svn status -u
A                0   proyecto/requisitos/vision.pdf
D              270   web_harmonia
A                0   web_pruebas
```

TortoiseSvn

1. Con el botón derecho del ratón sobre el documento a añadir seleccionar TortoiseSvn -> Delete



2. TortoiseSvn nos informa de que ha eliminado el directorio del control de versiones cambiando el icono que tiene asociado el directorio. web_harmonia ya no tiene el signo de “todo correcto”



7.6.- ¿Cómo creo nuevos directorios en mi directorio de trabajo?

Cliente svn

1. Crearé directorios de trabajo con el comando svn mkdir

```
svn mkdir web_pruebas
   A         web_pruebas
```

2. El comando me informa de que ha creado el directorio en el directorio de trabajo y la "A" me indica que está preparado para ser añadido también al repositorio en cuanto le digamos.

TortoiseSvn

1. La creación de directorios en TortoiseSvn es igual que la de cualquier fichero. Se crea el directorio desde el sistema operativo y se añade al control de versiones con la opción del menú contextual TortoiseSvn -> Add

7.7.- ¿Para qué creo versiones?

Una versión de una aplicación está formada por el conjunto de componentes que solidariamente cumplen el objetivo definido para la aplicación. Cuando cambiamos un componente determinado de una aplicación, es frecuente que otros componentes se vean modificados. Por ello es necesario realizar una gestión de la configuración en cualquier proyecto.

Cuando vaya a poner en línea una aplicación tendré que agrupar a todos los componentes que funcionen conjuntamente y darles la estructura necesaria. Es no solo conveniente sino necesario, el saber exactamente cuales son los componentes de una versión ejecutable y el ser capaz siempre de volver a una versión determinada de una aplicación.

Esto nos permitirá cumplir algunos de los objetivos de la disciplina de la gestión de la configuración, como son identificar los componentes de trabajo que componen un baseline determinado, controlar los cambios de los items de configuración, mantener la integridad de los baseline.

7.8.- ¿Cómo creo una versión?

La forma de crear una versión en subversion es extremadamente sencilla. El conjunto de ficheros que queremos que constituyan la versión puede estar ya en el repositorio colgando de alguna carpeta o puede estar en nuestra máquina local en algún directorio de trabajo. Veremos las dos opciones.

Conviene recordar que una versión no debe componerse únicamente de ejecutables, sino que es necesario identificar otro tipo de artefactos, como son los fuentes, los requisitos, los artefactos de análisis y diseño, etc.

Opción 1: En https://repositorio/aplicacion_proyecto/proyecto/bin/principal/ está la aplicación, que hemos probado que funciona y que queremos crear una versión, concretamente la 1.0 con el contenido de esa carpeta.

Opción 2: En Mi-Version, en local, está la aplicación, que hemos probado que funciona y queremos crear la versión 1.0 con el contenido de esta carpeta.

Cliente svn

Opción 1: El ejecutable está en el repositorio.

Copiaré desde la carpeta principal a la carpeta versiones/version1.0

```
$>svn copy https://repositorio/aplicacion_proyecto/proyecto/bin/principal/
https://repositorio/aplicacion_proyecto/proyecto/bin/versiones/version1.0
-m "creo la version 1.0 de la aplicacion"

Committed revision 275.
```

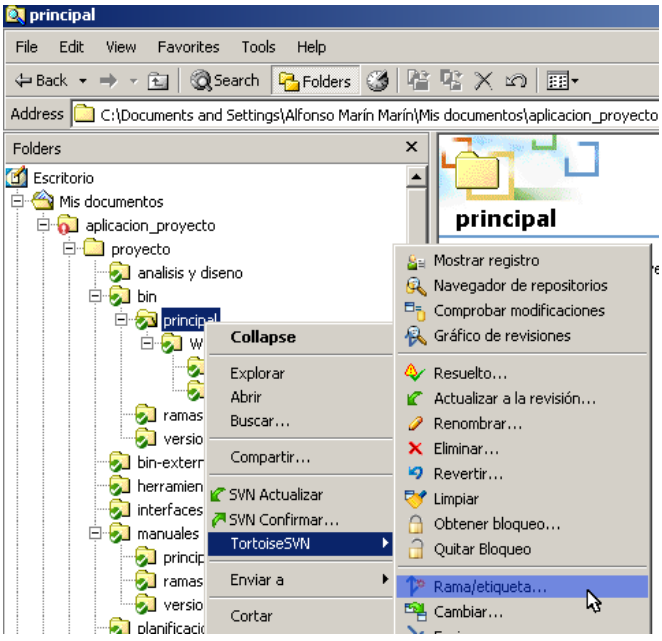
Opción 2: El ejecutable está en local

Copiaré desde la carpeta local a la carpeta versiones/version1.0

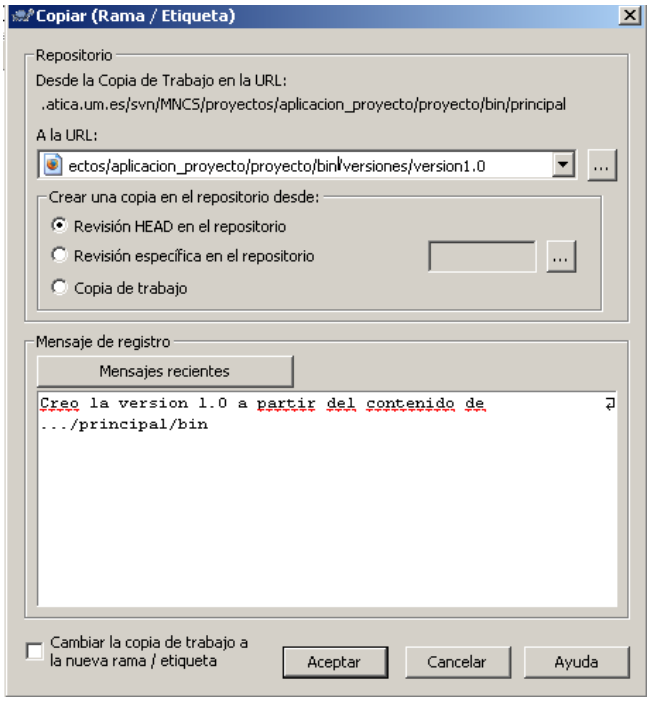
```
$> svn copy mi-version
https://repositorio/aplicacion_proyecto/proyecto/bin/versiones/version1.0
-m "creo la version 1.0 de la aplicacion"

Committed revision 275.
```

TortoiseSvn

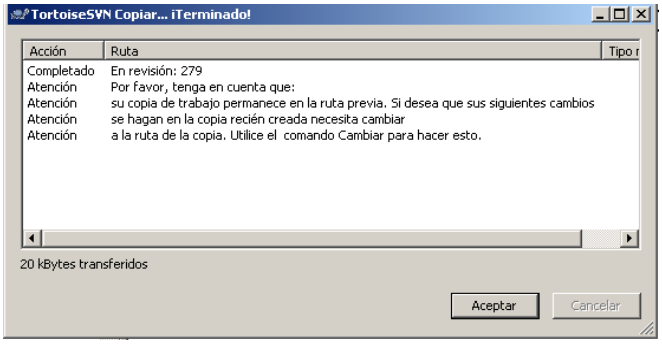
1. Me posicionaré sobre la carpeta correspondiente al directorio de trabajo donde está el ejecutable que quiero copiar y seleccionaré TortoiseSvn -> Rama/etiqueta.



2. TortoiseSvn pregunta por la carpeta en el repositorio donde quiero crear la versión.



3. TortoiseSvn nos avisará que el directorio de trabajo no está apuntando a la nueva versión. Deberemos hacer una actualización del directorio de trabajo para crear la nueva carpeta con la versión.



7.9.- ¿Para qué creo ramas?

Las ramas se utilizan para realizar modificaciones sobre partes del desarrollo ya realizadas sin interferir en el desarrollo principal.

Si queremos personalizar una aplicación ya realizada para un segundo cliente podemos crear una rama y personalizar el código mientras la rama principal del desarrollo continua siendo desarrollada.

Otra situación en la que puede ser interesante utilizar ramas es cuando es necesario adaptar un proyecto ya en marcha a cambios en interfaces de otras aplicaciones, etc.

Como regla general utilizaremos ramas cuando en un proyecto haya que continuar con la línea de desarrollo principal y haya que modificar de alguna manera lo ya desarrollarlo para hacer algún tipo de adaptación.

Características importantes de las ramas es que comparten un pasado común con el tronco principal y que, en muchos casos, las ramas pueden volver a fusionarse al tronco principal incorporando los cambios realizados en la rama.

7.10.- ¿Cómo creo una nueva rama?

Una rama se crea exactamente de la misma manera que una versión en subversión. De hecho, la única diferencia que existe entre una rama y una versión es conceptual; una rama es una línea de desarrollo independiente de la principal, pero con un pasado común, mientras que una versión es un producto definitivo y acabado que se pone en línea.

En este punto es importante recordar dos cosas. Que las ramas deben crearse colgando de la carpeta ramas “correspondiente” y que en el mensaje debe indicarse claramente que se está creando una rama. Esto es así, para que el proceso de fusión posterior con el tronco principal sea más sencillo.

7.11.- ¿Cómo veo las modificaciones realizadas en una rama?

A lo largo del ciclo de vida de una rama voy modificando distintos ficheros de la rama. En cualquier momento puedo ver las diferencias que hay entre versiones del repositorio o entre versiones de archivos concretos.

Cliente svn

1. Con el comando svn log puedo ver el ciclo de vida de la rama

```
$>svn log
```

```
------------------------------------------------------------------------
r286 | pacom | 2007-06-18 12:12:45 +0200 (lun, 18 jun 2007) | 1 line
He modificado el fichero prueba.html de la rama1
------------------------------------------------------------------------
r285 | pacom | 2007-06-18 12:11:23 +0200 (lun, 18 jun 2007) | 1 line
Creo la rama 1 a partir del contenido de /bin/principal
```

2. Puedo comprobar las diferencias existentes en el archivo prueba.html entre la versión en que fue creada la rama y la versión final.

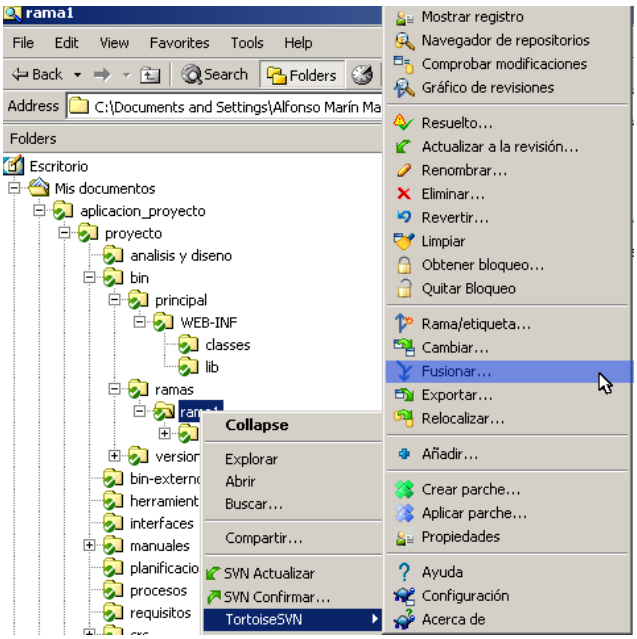
```
$>svn diff -r 285:HEAD prueba.html
```

```
Index: prueba.html
===================================================================
--- prueba.html (revision 285)
+++ prueba.html (revision 286)
@@ -1,5 +1,5 @@
 <html>
 <body>
-       Esto es una prueba
+       Esto es una prueba de ramas
 </body>
 </html>
```

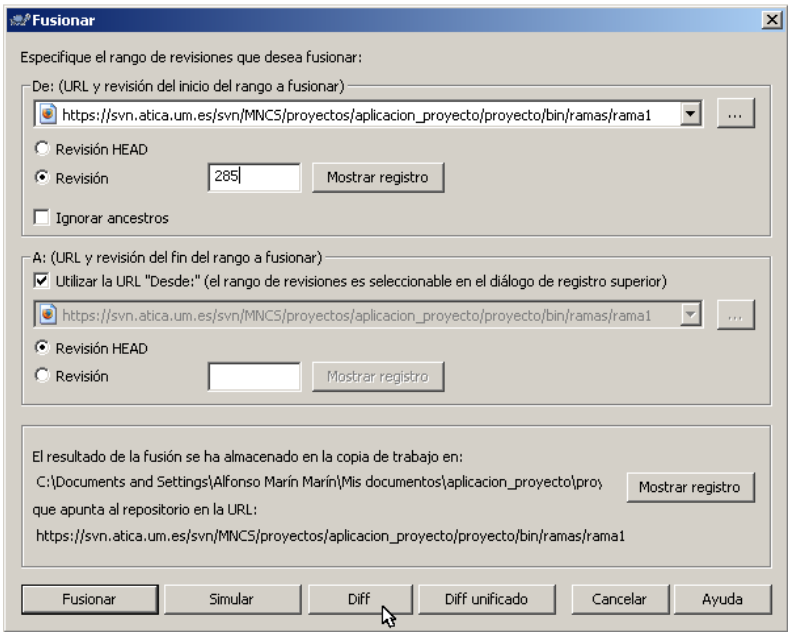
La revision HEAD es la ultima del repositorio.

TortoiseSvn

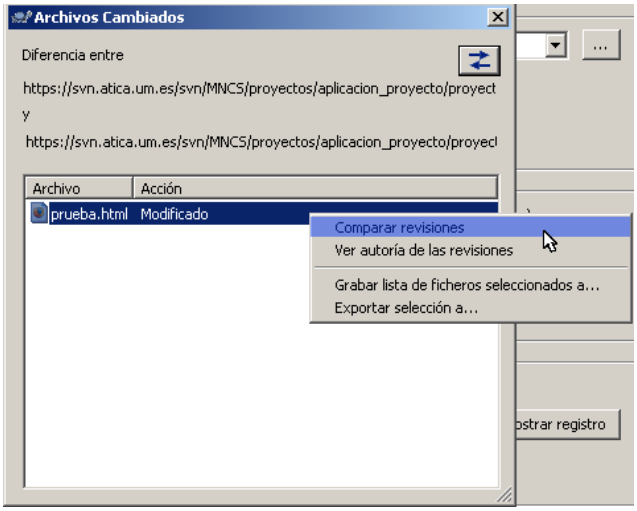
1. Selecciono el comando TortoiseSvn -> Fusionar situándome sobre la carpeta de la rama1



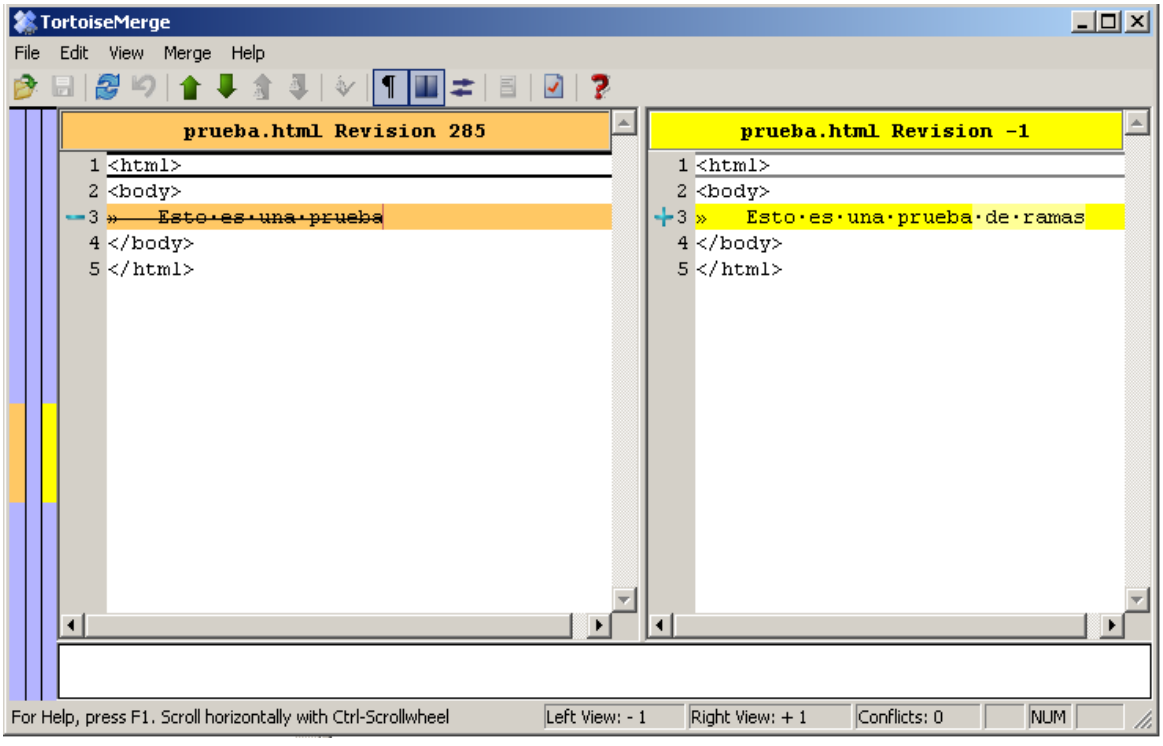
2. Selecciono las revisiones que quiero comparar y pulso el botón Diff



3. Tortoise me muestra los archivos que han sido modificados. Puedo seleccionar uno y decirle que me compare las revisiones.



4. Tortoise me mostrará las diferencias que hay entre las versiones del archivo seleccionado.



7.12.- ¿Cómo unifico una rama y el tronco principal?

En cualquier momento puedo estar interesado en incorporar los cambios realizados en la rama, y que hemos visto en el apartado anterior cómo descubrirlos.

NOTA: Es importante tener en cuenta que el fichero prueba.html del tronco principal puede haber sido modificado por otra persona. Remarco el que vamos a incorporar únicamente las modificaciones hechas en la rama manteniendo cualquier modificación que haya sufrido el fichero en el tronco principal. Otra cuestión a tener en cuenta es que la unificación se producirá en un directorio de trabajo, en nuestro caso en el .../bin/principal y posteriormente habrán de comitearse al repositorio.

Cliente svn

1. Me situo en el directorio de trabajo donde voy a fusionar el fichero del tronco principal con las diferencias entre las versiones

```
$> cd ../../principal
```

2. Indico que fusione el archivo prueba.html con las diferencias introducidas en la rama.

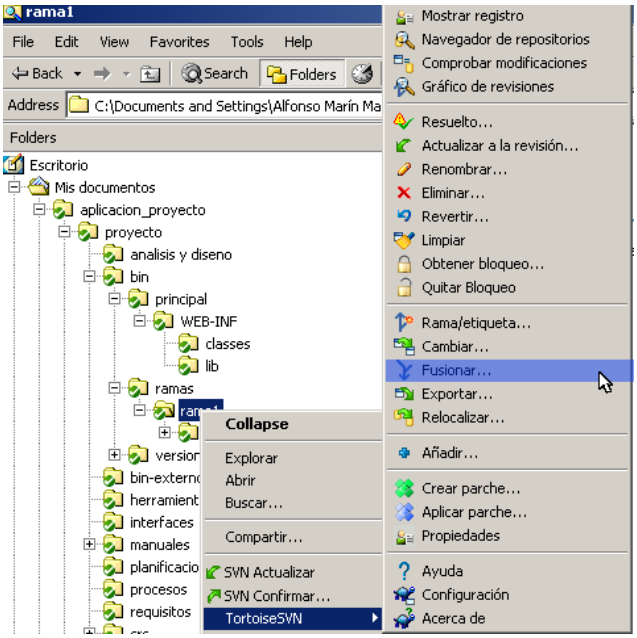
```
svn merge -r 285:HEAD
https://svn.atica.um.es/svn/MNCS/proyectos/aplicacion_proyecto/proyecto/bin/ramas/rama1/prueba.html
U    prueba.html
```

3. Actualizo el repositorio.

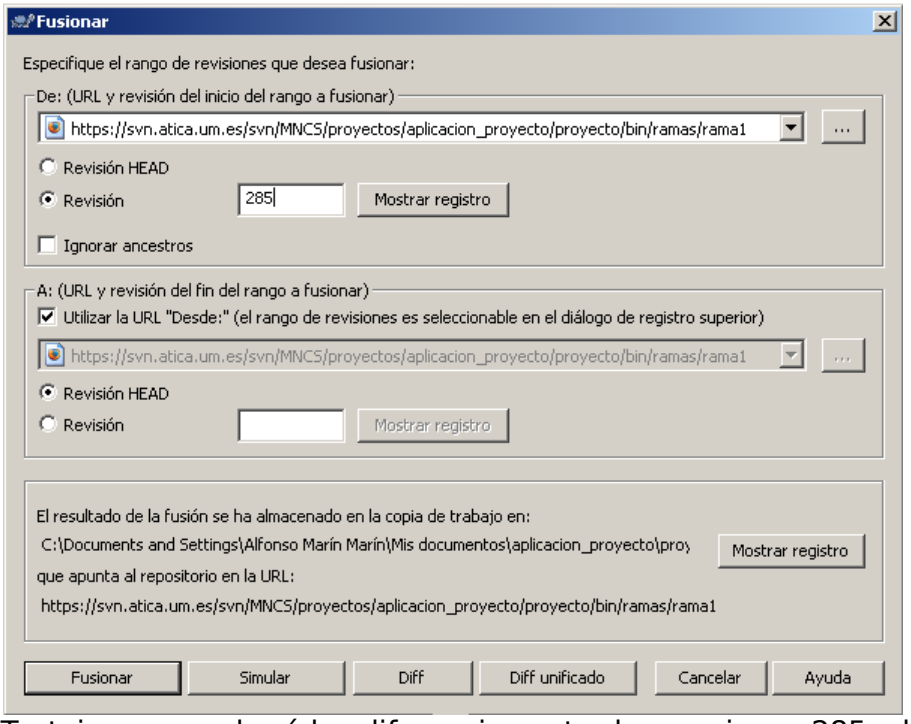
```
$>svn commit -m "Fusionado prueba.html del tronco principal con los cambios de la rama1 entre las revisiones 285 y 286 del repositorio"
Sending        proyecto/bin/principal/prueba.html
Transmitting file data .
Committed revision 287.
```

TortoiseSvn

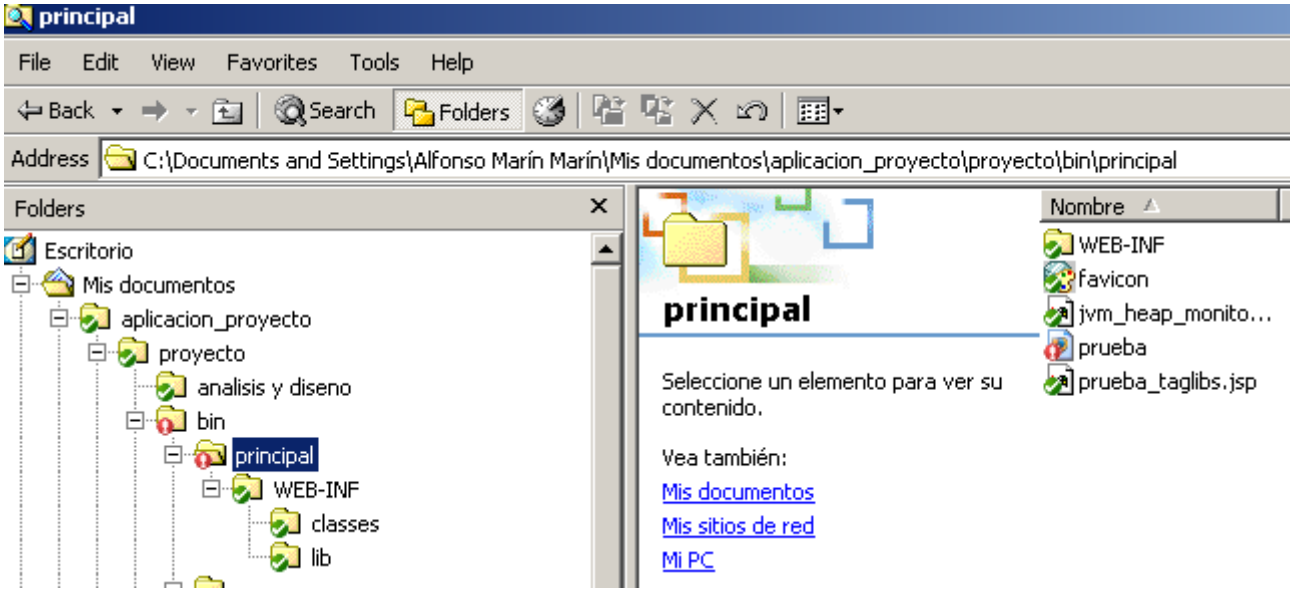
1. Selecciono el comando TortoiseSvn -> Fusionar situándome sobre la carpeta principal



2. Selecciono las revisiones que quiero fusionar. Primero pulso Simular para ver el resultado final y luego Fusionar.



3. Tortoise comprobará las diferencias entre las versiones 285 y la HEAD y las incorporará al fichero .../bin/principal/prueba.html. En el explorador de Windows veré que el archivo ha sido modificado.



7.13.- ¿Cómo veo las diferencias entre mi directorio de trabajo y el repositorio?

Las diferencias entre el repositorio y el directorio de trabajo pueden provenir de dos fuentes. La primera que hemos modificado el directorio de trabajo sin actualizar el repositorio. La segunda porque algún compañero ha modificado el repositorio.

Cliente Svn

Ejecutando el comando svn status -u

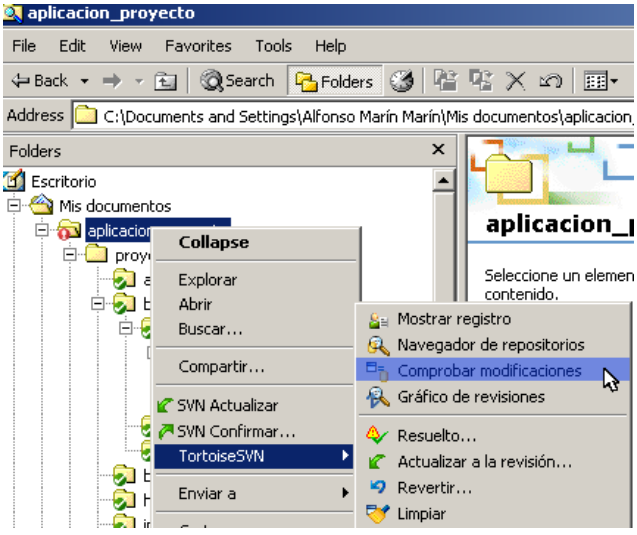
```
$>svn status -u
```

```
       *       proyecto/bin/versiones/version1.0/WEB-INF/lib
       *       proyecto/bin/versiones/version1.0/WEB-
INF/classes/TestDataSource.class
       *       proyecto/bin/versiones/version1.0/WEB-
INF/classes/HelloWorld.class
       *       proyecto/bin/versiones/version1.0/WEB-
INF/classes/SessionExample.class
       *       proyecto/bin/versiones/version1.0/WEB-
INF/classes/HelloWorldClass.class
       *       proyecto/bin/versiones/version1.0/WEB-
INF/classes/SnoopServlet.class
       *       proyecto/bin/versiones/version1.0/WEB-
INF/classes/ByeWorld.class
       *       proyecto/bin/versiones/version1.0/WEB-INF/classes
       *       proyecto/bin/versiones/version1.0/WEB-INF/web.xml
       *       proyecto/bin/versiones/version1.0/WEB-INF
       *       proyecto/bin/versiones/version1.0/jvm_heap_monitor.jsp
       *       proyecto/bin/versiones/version1.0/favicon.ico
       *       proyecto/bin/versiones/version1.0/prueba_taglibs.jsp
       *       proyecto/bin/versiones/version1.0/prueba.html
       *       proyecto/bin/versiones/version1.0
       * 274   proyecto/bin/versiones
M        274   proyecto/bin/principal/prueba.html
Status against revision:    279
```

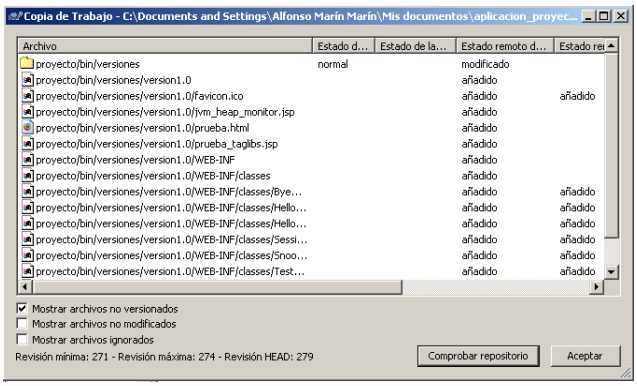
Los archivos o directorios que tienen un * son elementos con una nueva versión en el repositorio. Los que llevan un número indican cual es la versión que hay en el directorio de trabajo, y los que llevan un M indican que han sido modificados.

TortoiseSvn

1. Me posicionaré sobre la carpeta raíz del directorio de trabajo y seleccionaré TortoiseSvn -> Comprobar modificaciones.



2. TortoiseSvn me mostrará una ventana con todas las modificaciones que hay entre el repositorio y mi directorio de trabajo.


Copia de Trabajo - C:\Documents and Settings\Alfonso Marín Marín\Mis documentos\aplicacion_proyec...
Archivo
Estado d...
Estado de la...
Estado remoto d...
Estado re
proyecto/bin/versiones normal modificado
proyecto/bin/versiones/version1.0 añadido
proyecto/bin/versiones/version1.0/favicon.ico añadido añadido
proyecto/bin/versiones/version1.0/jvm_heap_monitor.jsp añadido
proyecto/bin/versiones/version1.0/prueba.html añadido
proyecto/bin/versiones/version1.0/prueba_taglibs.jsp añadido
proyecto/bin/versiones/version1.0/WEB-INF añadido
proyecto/bin/versiones/version1.0/WEB-INF/classes añadido
proyecto/bin/versiones/version1.0/WEB-INF/classes/Bye... añadido añadido
proyecto/bin/versiones/version1.0/WEB-INF/classes/Hello... añadido añadido
proyecto/bin/versiones/version1.0/WEB-INF/classes/Hello... añadido añadido
proyecto/bin/versiones/version1.0/WEB-INF/classes/Sessi... añadido añadido
proyecto/bin/versiones/version1.0/WEB-INF/classes/Snoo... añadido añadido
proyecto/bin/versiones/version1.0/WEB-INF/classes/Test... añadido añadido
Mostrar archivos no versionados
Mostrar archivos no modificados
Mostrar archivos ignorados
Revisión mínima: 271 - Revisión máxima: 274 - Revisión HEAD: 279
Comprobar repositorio
Aceptar

7.14.- ¿Cómo actualizo el repositorio con mi directorio de trabajo?

Cliente svn

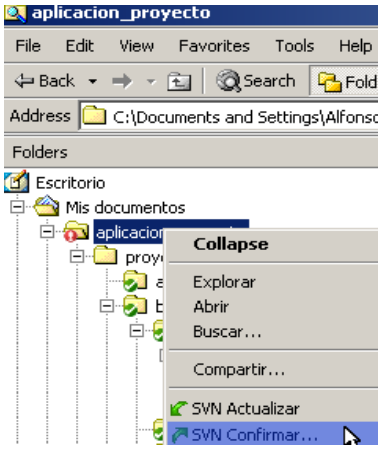
Con el comando svn commit aplico las modificaciones realizadas en mi directorio de trabajo al repositorio. Hay que tener en cuenta que estoy subiendo modificaciones del directorio de trabajo al repositorio. Que para bajar las modificaciones del repositorio al fichero de trabajo necesitaré realizar un svn update.

```
svn commit -m "Actualizacion del repositorio con las modificaciones
habidas hasta este momento"
```

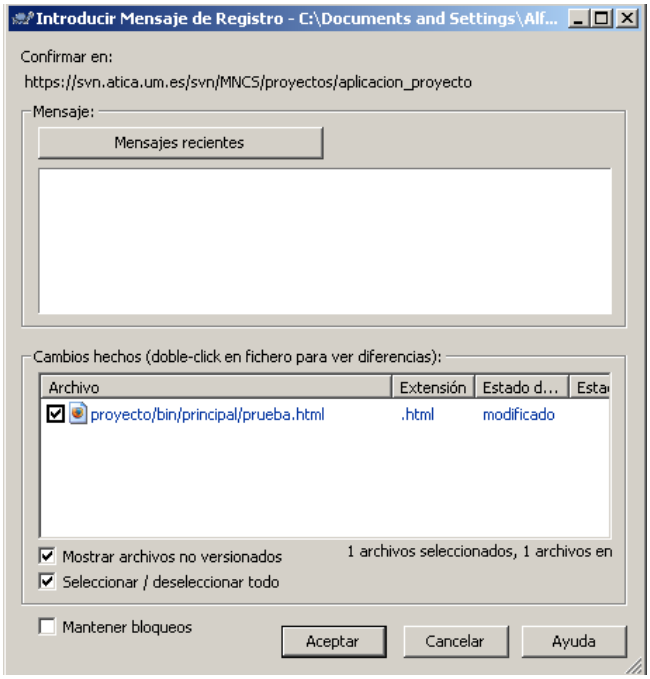
```
Sending        proyecto/bin/principal/prueba.html
Transmitting file data .
Committed revision 280.
```

TortoiseSvn

1. Me posicionaré sobre la carpeta del directorio de trabajo que quiera subir al repositorio y seleccionaré SVN Confirmar...



2. TortoiseSvn informará de los ficheros que van a ser actualizados en el repositorio.



7.15.- ¿Cómo resuelvo los posibles conflictos?

En el apartado 6.3.14 vimos como resolver conflictos. Veamos en este apartado como tratarlos con el cliente svn y con TortoiseSvn.

Cliente svn

3. Al actualizar el directorio de trabajo me muestra el conflicto en el fichero prueba.htm. Según vimos anteriormente me creara automaticamente los ficheros "prueba.mine", "prueba.r_original", "prueba.r_repositorio" que corresponden al fichero tal y como estaba en el directorio de trabajo antes de la actualización, al fichero tal y como estaba en el ultimo checkout y al fichero tal y como estaba en el repositorio respectivamente.

```
$>svn update
D    proyecto/manuales/ramas
C    proyecto/bin/principal/prueba.html
Updated to revision 281.
```

2. Si listamos el contenido del directorio vemos la situación en que quedó.

```
$> ls -l
  -rw-r--r--  1 pacom pacom  318 jun 15 10:17 favicon.ico
  -rw-r--r--  1 pacom pacom 1215 jun 15 10:17 jvm_heap_monitor.jsp
  -rw-r--r--  1 pacom pacom 1382 jun 15 10:22 prueba.htm
  -rw-r--r--  1 pacom pacom   50 jun 18 10:22 prueba. mine
  -rw-r--r--  1 pacom pacom 1228 jun 18 10:22 prueba.r274
  -rw-r--r--  1 pacom pacom 1296 jun 18 10:22 prueba.r280
  -rw-r--r--  1 pacom pacom  118 jun 15 10:17 prueba_tablibs.jsp
  -rw-r--r--  5 pacom pacom 4096 jun 15 10:17 WEB-INF/
```

3. Dejaremos el prueba.htm tal y como queremos que quede y resolvemos conflictos.

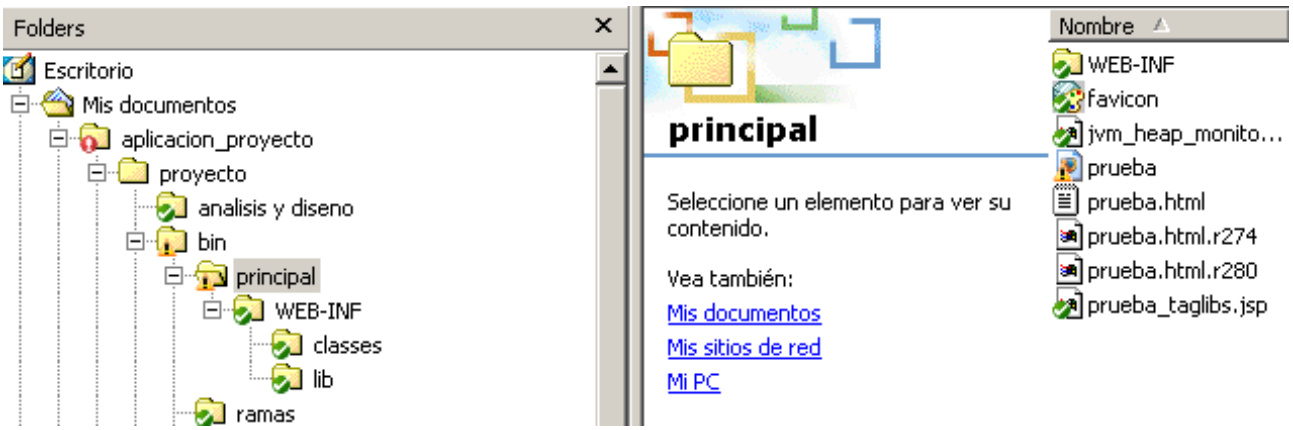
```
$>svn resolved prueba.html
  Resolved conflicted state of 'prueba.htm'
```

4. Una vez resuelto el conflicto la copia de trabajo estara lista para ser comiteado.

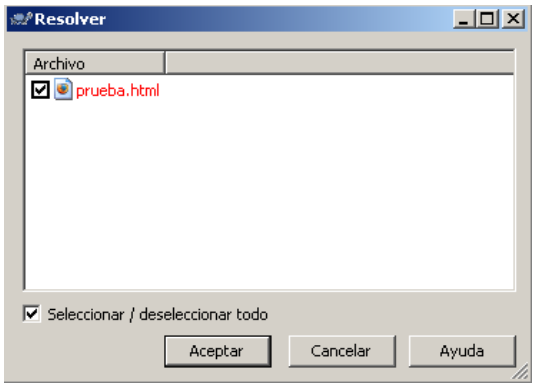
```
svn status -u
M             281   prueba.html
Status against revision:    281
```

TortoiseSvn

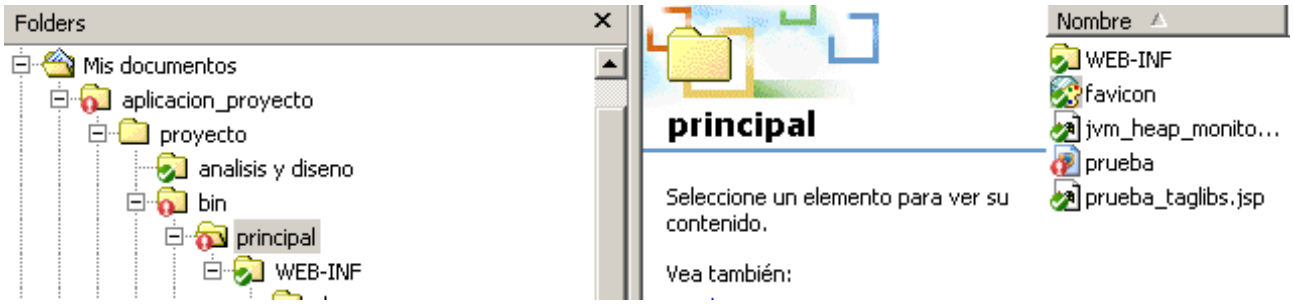
1. Al actualizar el directorio de trabajo me muestra el conflicto en el fichero prueba.htm. Según vimos anteriormente me creara automaticamente los ficheros "prueba.mine", "prueba.r_original", "prueba.r_repositorio" que corresponden al fichero tal y como estaba en el directorio de trabajo antes de la actualización, al fichero tal y como estaba en el ultimo checkout y al fichero tal y como estaba en el repositorio respectivamente.



2. Una vez que hemos dejado el fichero prueba.htm tal y como queremos que quede en el repositorio por cualquiera de los métodos posibles (modificando el fichero a mano, revirtiendo los cambios, o copiando una de los ficheros temporales sobre él) ejecutamos el comando TortoiseSvn -> Resuelto...



3. Una vez resuelto el conflicto la copia de trabajo estara lista para ser comiteada.



7.16.- ¿Cómo preparo una versión para ser puesta en línea?

Las versiones que son puestas en línea en los distintos servidores cuelgan de un directorio específico para cada uno de los servidores. Tendremos una carpeta llamada web_nombreservidor dentro del repositorio por cada uno de los servidores y es aquí donde tendremos que dejar las aplicaciones que queramos que corran en el servidor *nombreservidor*. La estructura de esta carpeta la podemos ver en el anexo I.

La idea es que colgando de .../bin/versiones/versionX.X tengamos la aplicación que queremos llevar al servidor. En el apartado 7.8 contábamos cómo preparar una versión. Una vez que la versión esté lista, con la estructura adecuada y comiteada en el repositorio tendremos que copiar el contenido de la carpeta .../bin/versiones/versionX.X a web_nombreservidor.

Cliente svn

1. Borro la aplicación que hay en línea en este instante

```
$>svn del https://repositorio/aplicacion_proyecto/web_talia\
/aplicacion_proyecto/web_aplicacion_proyecto\
-m "Borro la aplicación puesta en línea porque voy a poner otra en marcha"

Committed revision 296.
```

2. Copiamos una versión de la aplicación a la estructura recomendada por sistemas.

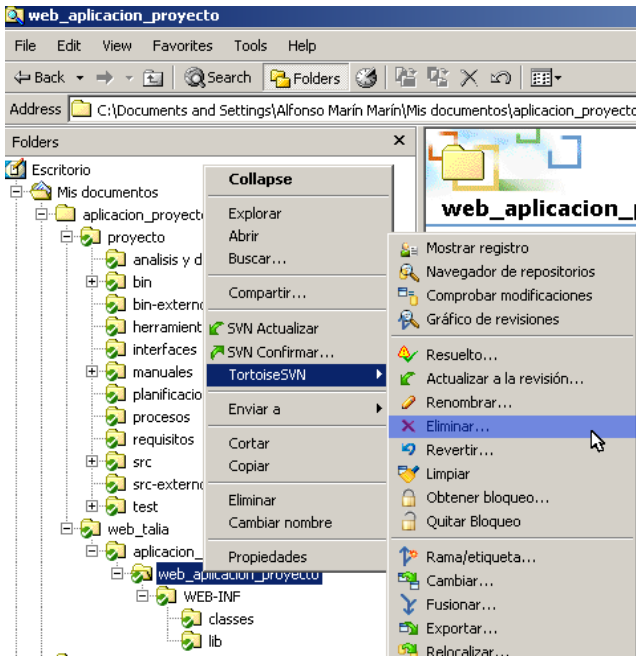
```
$> svn copy https://repositorio/aplicacion_proyecto/proyecto/bin\
/versiones/version1.0/ \
https://repositorio/aplicacion_proyecto/web_talia/aplicacion_proyecto\
/web_aplicacion_proyecto \
-m "Copio la version 1.0 para ser enviada al servidor"
```

```
Committed revision 290.
```

Nota: Es muy importante indicar en el mensaje qué versión estamos poniendo en línea, ya que más adelante necesitaré consultar esta información.

TortoiseSvn

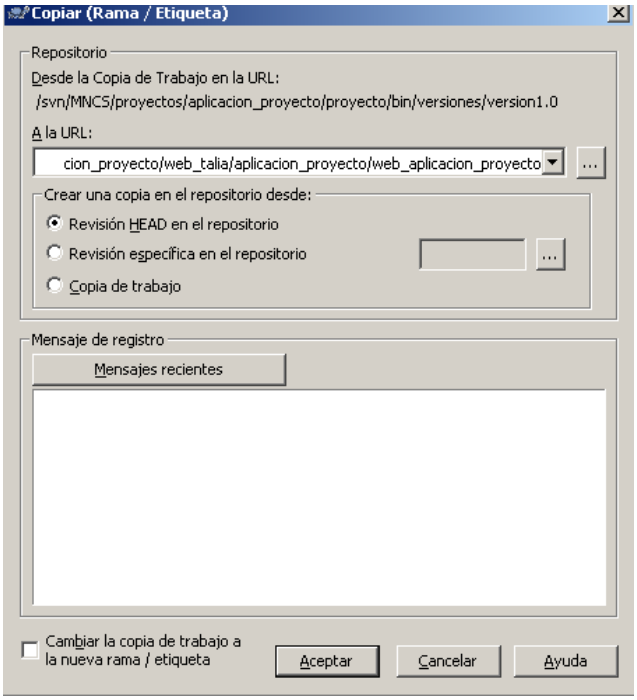
1. Borraré la aplicación que hay en línea en este instante.



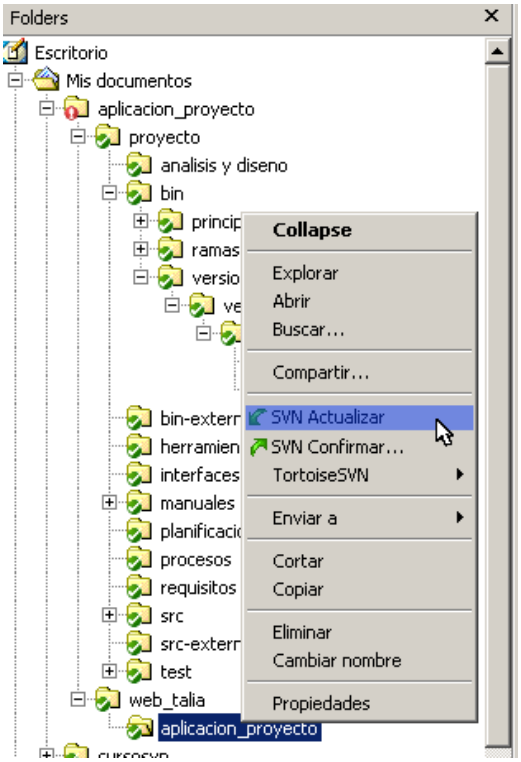
2. Haré un commit para borrarlo en el repositorio.

3. Me posicionaré sobre la carpeta correspondiente al directorio de trabajo donde está el ejecutable que quiero poner en línea y seleccionaré TortoiseSvn -> Rama/etiqueta.

4. TortoiseSvn pregunta por la carpeta en el repositorio donde quiero crear la versión. Esta será .../web_servidor/nombre_aplicacion/web_aplicacion



5. TortoiseSvn nos avisará que el directorio de trabajo no está apuntando a la nueva versión. Deberemos hacer una actualización del directorio de trabajo para crear la nueva carpeta con la versión.



7.17.- ¿Cómo hago para subir mi versión a un servidor concreto?

Subir una aplicación al servidor correspondiente, una vez que en está en el repositorio tal y como explicamos en el apartado anterior es realmente sencillo. Consiste únicamente en colocar bajo el directorio aplicacion/web_servidor/aplicacion un fichero llamado restart y que contenga una única línea con el valor *true.*

En el servidor habrá un demonio que comprobará periódicamente el valor y de este fichero. Si existe y tiene el valor *true*, copiará la aplicación al servidor de aplicaciones y la pondrá en línea. Además pondrá este valor a *false*.

7.18.- ¿Cómo reinicio una aplicación?

El reinicio de una aplicación es extremadamente sencillo. Consiste simplemente en poner el valor del fichero aplicacion/web_servidor/aplicacion/restart a true y subir dicho fichero al respositorio. El demonio del servidor de aplicaciones volverá a cargar la aplicación y a reiniciarla.

7.19.- ¿Cómo compruebo la versión que está ejecutándose?

En el apartado 7.16 explicábamos donde debíamos copiar la aplicación para que pudiera ser subida automáticamente al servidor de aplicaciones. Vimos que consistía en hacer una copia del directorio del repositorio donde se encontraba la versión que queríamos poner en línea a un directorio especial, en función del servidor en que queríamos poner en línea en la aplicación. Durante la copia incluíamos un mensaje. Es en este mensaje donde debemos especificar claramente la versión que estamos copiando. Si se trata de la 1.0, la 2.3 o la 3.5.

Estos mensajes pueden ser consultados más adelante y así saber que copia tenemos en cada momento en línea.

Cliente svn

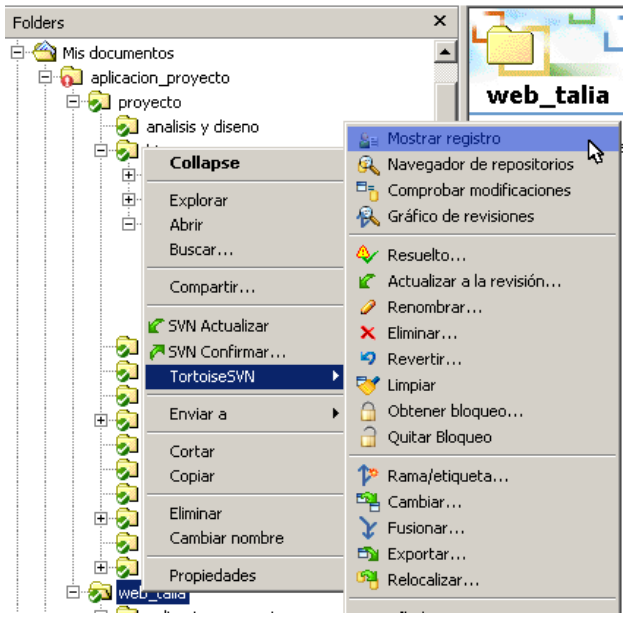
1. Consulto los log de la carpeta del repositorio donde está la aplicación que se puso en línea.

```
$>svn log https://repositorio/aplicacion_proyecto/web_talia\
\aplicacion_proyecto
```

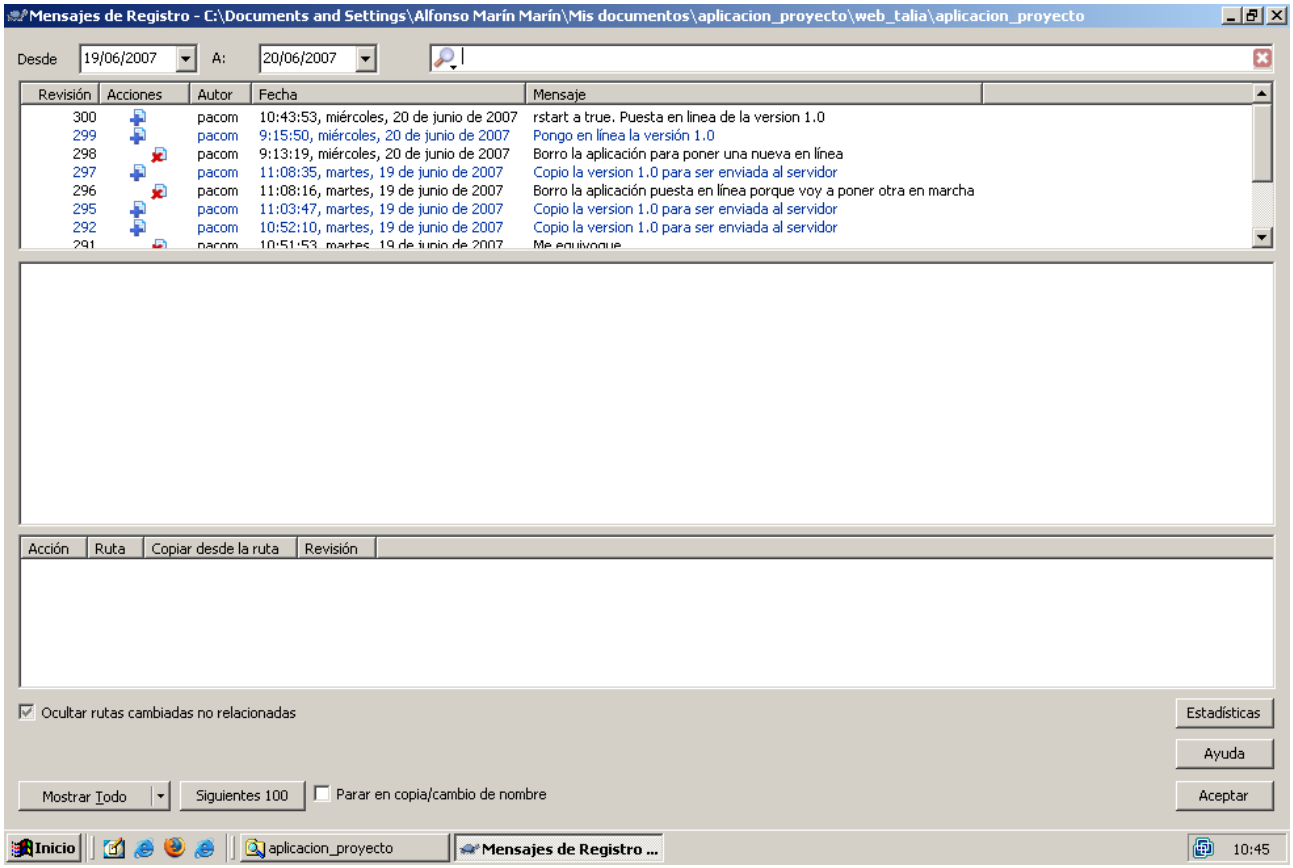
```
------------------------------------------------------------------------
r300 | pacom | 2007-06-20 10:43:53 +0200 (mié, 20 jun 2007) | 1 line
rstart a true. Puesta en linea de la version 1.0
------------------------------------------------------------------------
r299 | pacom | 2007-06-20 09:15:50 +0200 (mié, 20 jun 2007) | 1 line
Pongo en línea la versión 1.0
------------------------------------------------------------------------
r298 | pacom | 2007-06-20 09:13:19 +0200 (mié, 20 jun 2007) | 1 line
Borro la aplicación para poner una nueva en línea
```

TortoiseSvn

1. Situándome sobre la carpeta del directorio de trabajo donde copie la aplicación que luego subí al repositorio para ser copiada en el servidor de aplicaciones correspondiente pulso la opción TortoiseSvn -> Mostrar registro



2. Veo en la última revisión cual es la versión que se puso en línea


Mensajes de Registro - C:\Documents and Settings\Alfonso Marín Marín\Mis documentos\aplicacion_proyecto\web_talia\aplicacion_proyecto
Desde 19/06/2007 A: 20/06/2007
Revisión Acciones Autor Fecha Mensaje
300 pacom 10:43:53, miércoles, 20 de junio de 2007 rstart a true. Puesta en linea de la version 1.0
299 pacom 9:15:50, miércoles, 20 de junio de 2007 Pongo en línea la versión 1.0
298 pacom 9:13:19, miércoles, 20 de junio de 2007 Borro la aplicación para poner una nueva en línea
297 pacom 11:08:35, martes, 19 de junio de 2007 Copio la version 1.0 para ser enviada al servidor
296 pacom 11:08:16, martes, 19 de junio de 2007 Borro la aplicación puesta en línea porque voy a poner otra en marcha
295 pacom 11:03:47, martes, 19 de junio de 2007 Copio la version 1.0 para ser enviada al servidor
292 pacom 10:52:10, martes, 19 de junio de 2007 Copio la version 1.0 para ser enviada al servidor
Acción Ruta Copiar desde la ruta Revisión
Ocultar rutas cambiadas no relacionadas
Estadísticas
Ayuda
Mostrar Todo
Siguientes 100
Parar en copia/cambio de nombre
Aceptar
Inicio
aplicacion_proyecto
Mensajes de Registro ...
10:45

7.20.- ¿Cómo retorno a una versión anterior?

Una de las mayores ventajas de tener un gestor de versiones es que siempre podemos volver a una versión anterior. Imaginemos que en el caso anterior hemos puesto en línea la versión 1.1 pero nos hemos dado cuenta que esta versión no funciona correctamente y queremos volver a la versión 1.0. A continuación veremos como hacerlo.

Cliente svn

1. Compruebo los logs para ver en que revisión de subversión puse en línea las versiones.

```
$>svn log https://repositorio/aplicacion_proyecto/web_talia
```

```
------------------------------------------------------------------------
r306 | pacom | 2007-06-20 11:03:53 +0200 (mié, 20 jun 2007) | 1 line
Restart a true. Pongo en linea al version 1.1
------------------------------------------------------------------------
r305 | pacom | 2007-06-20 11:01:53 +0200 (mié, 20 jun 2007) | 1 line
Copio la version 1.1 para ser puesta en linea
------------------------------------------------------------------------
r304 | pacom | 2007-06-20 10:51:00 +0200 (mié, 20 jun 2007) | 1 line
Borro la version anterior para poner en linea al version 1.1
------------------------------------------------------------------------
r300 | pacom | 2007-06-20 10:43:53 +0200 (mié, 20 jun 2007) | 1 line
restart a true. Puesta en linea de la version 1.0
------------------------------------------------------------------------
r299 | pacom | 2007-06-20 09:15:50 +0200 (mié, 20 jun 2007) | 1 line
Pongo en línea la versión 1.0
```

2. Compruebo en que revisión del repositorio se puso en línea la versión 1.0. En nuestro caso concreto veo que fue en la 300. Aquí podemos darnos cuenta de la importancia de que el texto de los mensaje sea el correcto.

3. Creo una nueva carpeta de trabajo en la que me bajo los archivos de la versión 1.0

```
svn checkout -r 300
https://repositorio/aplicacion_proyecto/web_talia/aplicacion_proyecto/web_
aplicacion_proyecto/prueba.html ./r300
```

```
A    r300/web_aplicacion_proyecto/prueba.html
A    r300/web_aplicacion_proyecto/WEB-INF
A    r300/web_aplicacion_proyecto/WEB-INF/lib
A    r300/web_aplicacion_proyecto/WEB-INF/web.xml
A    r300/web_aplicacion_proyecto/WEB-INF/classes
A    r300/web_aplicacion_proyecto/WEB-INF/classes/HelloWorld.class
A    r300/web_aplicacion_proyecto/WEB-
INF/classes/SessionExample.class
A    r300/web_aplicacion_proyecto/WEB-
INF/classes/HelloWorldClass.class
A    r300/web_aplicacion_proyecto/WEB-INF/classes/SnoopServlet.class
A    r300/web_aplicacion_proyecto/WEB-INF/classes/ByeWorld.class
A    r300/web_aplicacion_proyecto/WEB-
INF/classes/TestDataSource.class
A    r300/web_aplicacion_proyecto/jvm_heap_monitor.jsp
A    r300/web_aplicacion_proyecto/favicon.ico
A    r300/web_aplicacion_proyecto/prueba_taglibs.jsp
A    r300/restart
Checked out revision 300.
```

4. Copio estos archivos sobre el directorio de trabajo donde está ahora mismo la versión 1.1 y compruebo si hay archivos que he de borrar.

5. Hago un commit de la versión 1.0.

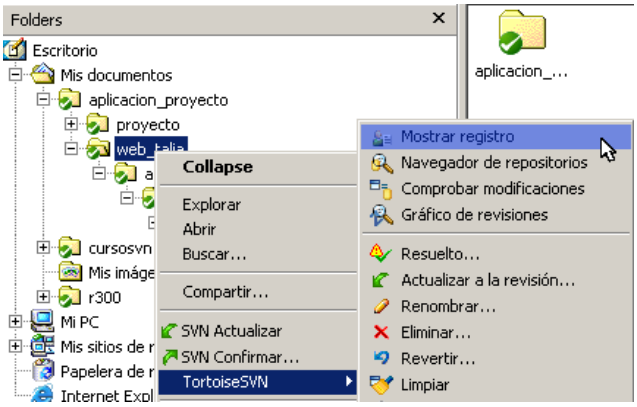
```
$>svn commit -m "Restauro la version 1.0"
```

```
Adding  (bin)  web_talia/aplicacion_proyecto/web_aplicacion_proyecto
/favicon.ico
Deleting       web_talia/aplicacion_proyecto/web_aplicacion_proyecto
/jvm_heap_monitor2.jsp
Sending
web_talia/aplicacion_proyecto/web_aplicacion_proyecto/prueba.html
Transmitting file data ..
Committed revision 309.
```

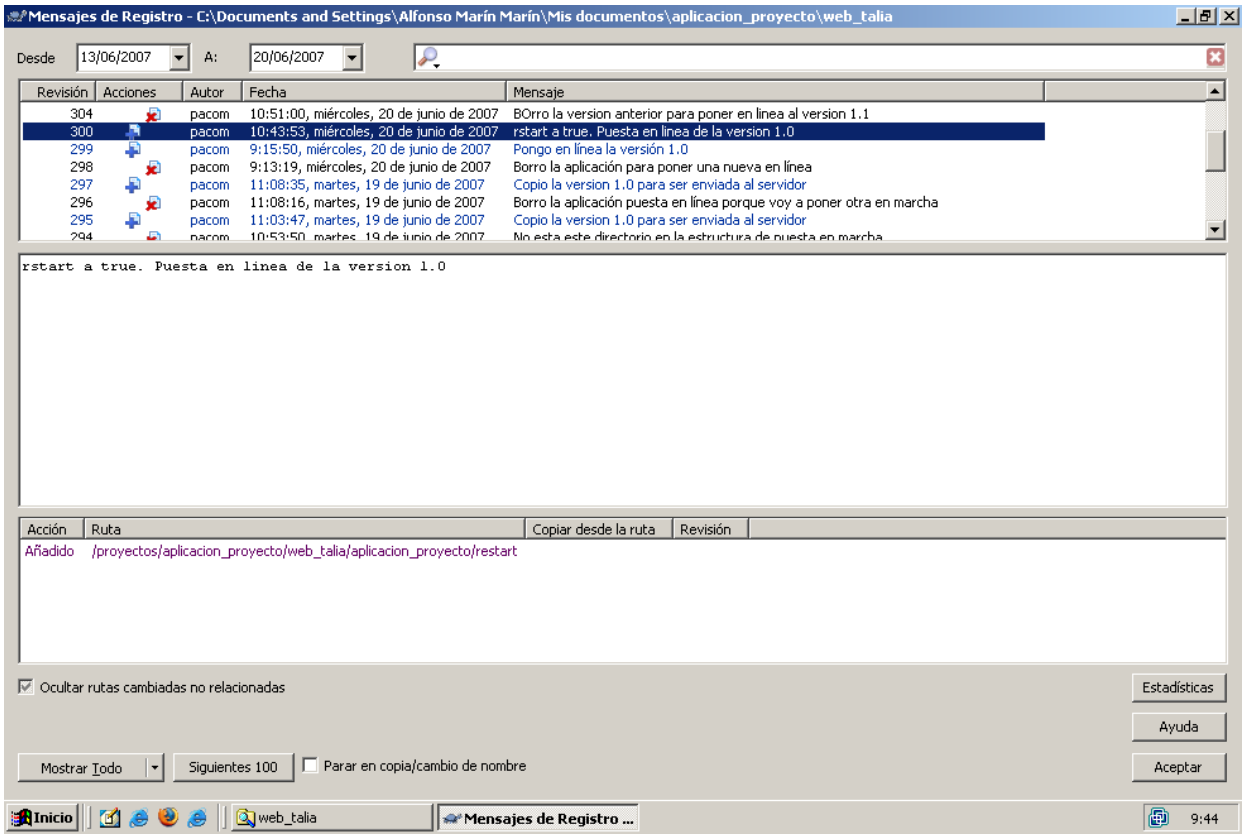
Otra opción es seguir las instrucciones del apartado 7.16 para poner una versión en línea.

TortoiseSvn

1. Compruebo los logs para ver en que revisión de subversión puse en línea las versiones. Situándome sobre la carpeta web_talia selecciono la opción TortoiseSvn -> Mostrar registro

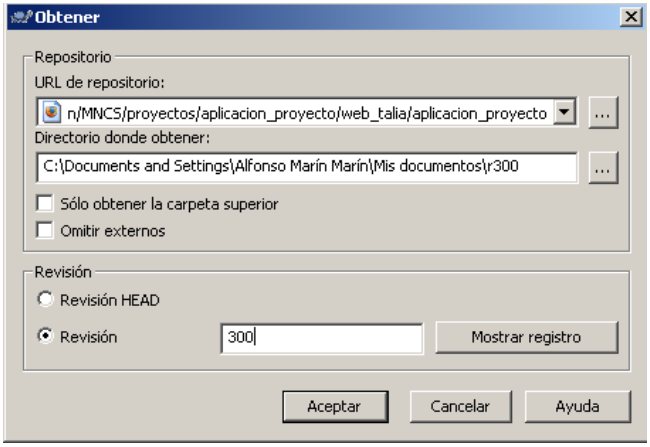


2. Compruebo en que revisión del repositorio se puso en línea la versión 1.0. En nuestro caso concreto veo que fue en la 300. Aquí podemos darnos cuenta de la importancia de que el texto de los mensaje sea el correcto.

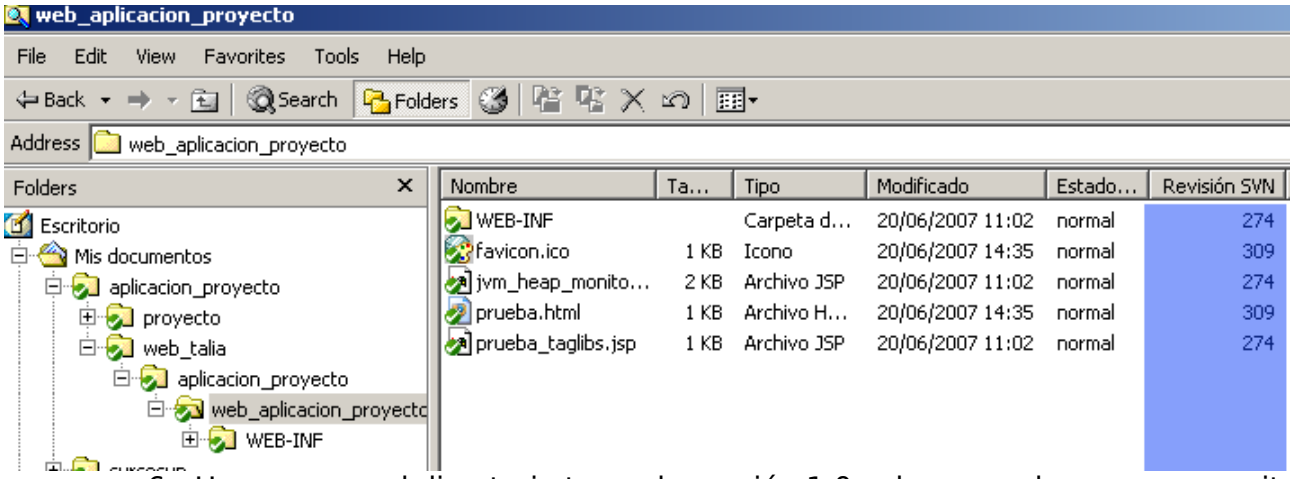


3. Creo una nueva carpeta de trabajo en la que me bajo los archivos de la versión 1.0

4. Hay que especificar claramente el directorio del repositorio donde están las aplicaciones que luego se ponen en línea, un nuevo directorio en local que sea el de trabajo y la revisión que queremos obtener.



5. Ahora copiaré los archivos de las versión 1.0 a la carpeta donde está ahora mismo la versión 1.1 y comprobaré si hay archivos que borrar, aquellos que sean nuevos de la versión 1.1, pero que no estuviesen en la 1.0. El explorador de windows nos ayuda a hacer esto, ya que hay una columna en la que indica el número de revisión en que cada archivo fue modificado.



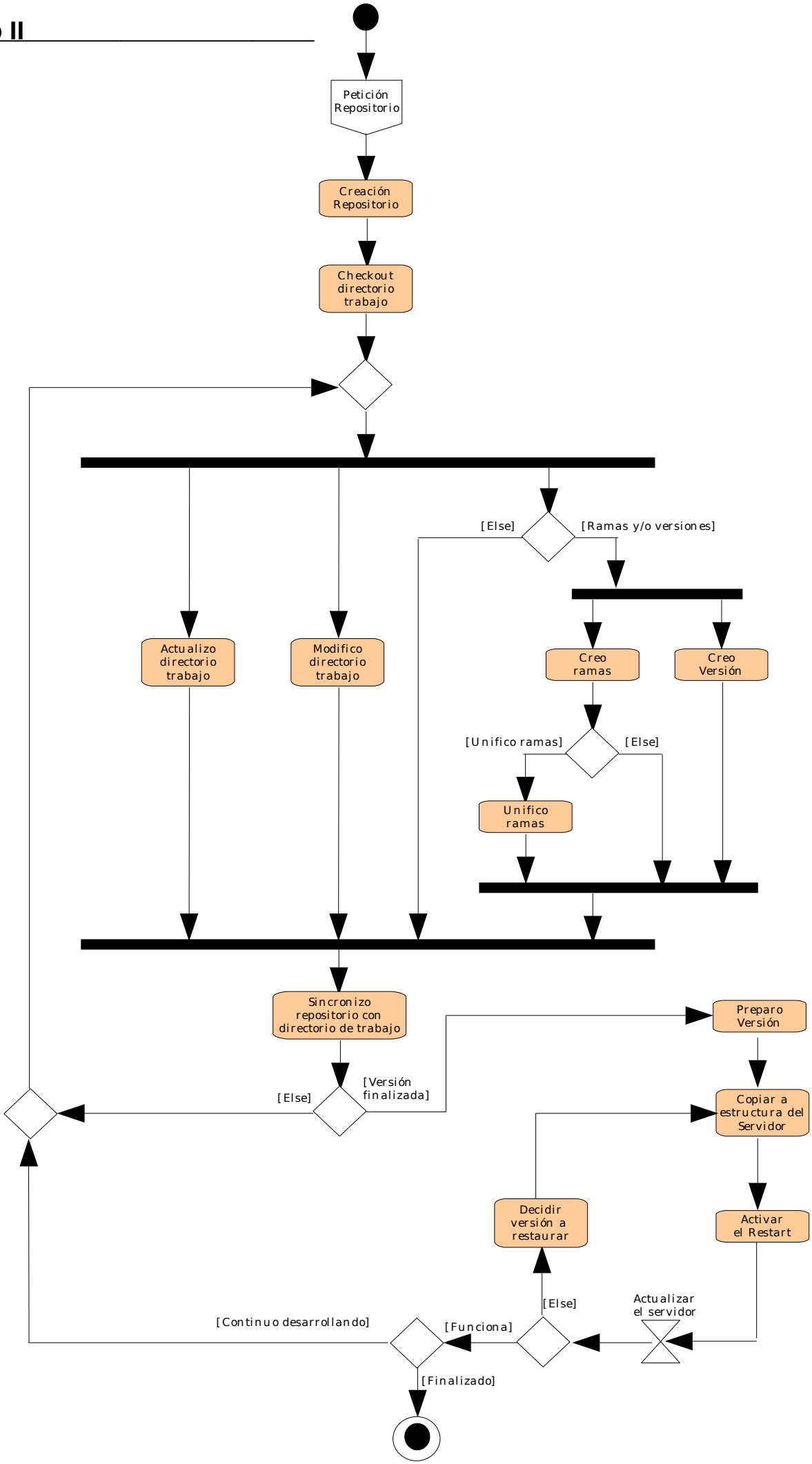
6. Una vez que el directorio tenga la versión 1.0 volvemos a hacer un commit, indicando en el mensaje que se trata de una restauración de la versión 1.0.

Anexo I

```
aplicación_proyecto
        proyecto
              procesos*
              planificacion*
              requisitos*
              analisis y diseno*
              interfaces*
              herramientas
              src-externo*
              bin-externo
              src
                 principal
                 ramas
                     rama 1
                     rama ...
                 versiones
                     version 1
                     version ...
              bin
                 principal
                 ramas
                     rama 1
                     rama ...
                 versiones
                     version 1
                     version ...
              test
                 principal
                 ramas
                     rama 1
                     rama ...
                 versiones
                     version 1
                     version ...
              manuales
                     principal
                     versiones
                           version 1
                           version ...

        web_talia (nombre servidor)
              servicio (nombre aplicación)
                     restart (true o false)
                     web_aplicación (nombre aplicación con web_delante)
                           WEB_INF
                                 classes
                                       ByeWorld.class
                                       HelloWorld.class
                                       ...

                                 lib
                                    ...
                                    ...
                                 web.xml
                           favicon.ico
                           jvm_heap_monitor.jsp
                           prueba.html
                           pruebataglibs.jsp

        web_harmonia (nombre servidor)
        ...
```

*La inclusión aquí de principal, ramas y versiones es opcional

Anexo II


Petición Repositorio
Creación Repositorio
Checkout directorio trabajo
[Else]
[Ramas y/o versiones]
Actualizo directorio trabajo
Modifico directorio trabajo
Creo ramas
Creo Versión
[Unifico ramas]
[Else]
Unifico ramas
Sincronizo repositorio con directorio de trabajo
Preparo Versión
[Else]
[Versión finalizada]
Copiar a estructura del Servidor
Decidir versión a restaurar
Activar el Restart
[Else]
Actualizar el servidor
[Continuo desarrollando]
[Funciona]
[Finalizado]