Subversion

Alejandro Ramírez [mailto:jano@1x4x9.info]

22 marzo 2004

Historial de revisiones

Revisión 1	22 marzo 2004
Revisión 2	24 julio 2004
Revisión 3	31 julio 2004

Tabla de contenidos

Introducción

Este artículo es un tutorial del sistema de control de versiones Subversion. Se incluyen ejemplos para Windows y *nix. No se asumen conocimientos previos. Visita 1x4x9.info [http://www.1x4x9.info] para obtener la última versión.

La referencia definitiva sobre Subversion es el libro electrónico "Version Control with Subversion [http://svnbook.red-bean.com/]", que se distribuye gratis en Internet, y que publica O'Reilly en digital [http://safari.oreilly.com/] y papel. Dicho libro contiene toda la información necesaria para instalar, usar, administrar, y desarrollar Subversion. Este documento solo cubre una mínima parte de su contenido.

Sistemas de control de versiones

Un sistema de control de versiones es *un software que administra el acceso a un conjunto de ficheros, y mantiene un historial de cambios realizados*. El control de versiones es útil para guardar cualquier documento que cambie con frecuencia, como una novela, o el código fuente de un programa.

Normalmente consiste en una copia maestra en un repositorio central, y un programa cliente con el que cada usuario sincroniza su copia local. Esto permite compartir los cambios sobre un mismo conjunto de ficheros. Además, el repositorio guarda registro de los cambios realizados por cada usuario, y permite volver a un estado anterior en caso de necesidad.

Pero, ¿que hacer cuando dos usuarios intentan modificar el mismo fichero?. Existen dos estrategias:

- *Bloqueos*: el usuario bloquea el fichero durante su edición, evitando el acceso concurrente de otros usuarios. Existen varios problemas: el usuario que acapara ficheros, el interbloqueo entre usuarios que necesitan varios ficheros, y la falta de concurrencia.
- *Merge* (fusión de cambios): los ficheros se acceden concurrentemente. Los cambios realizados sobre un mismo fichero son fusionados inteligentemente por el sistema. El único problema es el intento de fusión de cambios incompatibles, que ha de solucionarse manualmente.

Existen multitud de sistemas de control de versiones [http://dmoz.org/Computers/Software/Configuration_Management/Tools/], pero sin duda, el más popular es CVS (Concurrent Versions System). CVS tuvo el merito de ser el primer sistema usado por el movimiento de código abierto para que los programadores colaboraran remotamente mediante el envío de parches. Es de uso gratuito, código abierto, y emplea fusión de cambios.

Subversion se creó para igualar y mejorar la funcionalidad de CVS, preservando su filosofía de desarrollo. Su desarrollo comenzó en el año 2000 como proyecto de código abierto esponsorizado por CollabNet [http://www.collab.net/]. El líder del equipo de desarrollo fue Karl Fogel, autor de Open Source Development with CVS [http://cvsbook.red-bean.com/], y fundador de Cyclic Software (compañía de desarrollo y soporte comercial para CVS, hoy adquirida por SourceGear [http://www.sourcegear.com/]). La versión 1.0 fue publicada en febrero del 2004. Emplea licencia Apache/BSD.

Hay una comparativa entre 13 sistemas de control de versiones en better-scm.berlios.de [http://better-scm.berlios.de/comparison/comparison.html], y otra entre Subversion y CVS en wiki.gnuarch.org [http://wiki.gnuarch.org/moin.cgi/SubVersionAndCvsComparison]. Este último web es también el wiki de GNU Arch [http://www.gnu.org/software/gnu-arch/], un sistema de control de versiones de código abierto que no he probado.

¿Por qué son necesarios?

Supongamos que estamos haciendo un programa de cierto tamaño en colaboración con otra persona. Lo más primitivo es compartir cambios usando ficheros comprimidos. Pero este sistema es propenso a errores: ¿estamos enviando todo el código?, ¿estamos sobrescribiendo algún cambio?, ¿que ficheros debemos actualizar?, ¿quien tiene la versión maestra del código?

Todos los sistemas de control de versiones tienen ciertas características que acaban con estas preocupaciones. Esto es lo que aporta un sistema de control de versiones a un equipo:

- Actualiza ficheros modificados. El cliente recorre nuestro código y sincroniza nuestra copia local con el repositorio.
- Copias de seguridad centralizadas. Solo el administrador debe preocuparse de realizar copias de seguridad en el repositorio. Esto se automatiza fácilmente con una tarea cron [http://www.gnu.org/directory/cron.html] o similares.
- Historial de cambios. El repositorio guarda registro de todos los cambios realizados. Es posible recuperar cualquiera de las versiones anteriores de cualquier fichero. Si alguien borra todos los

ficheros, podemos volver atrás y recuperar su contenido.

- Acceso remoto. Es posible acceder remotamente al repositorio. No es necesario que el equipo este dentro de la misma LAN.

- Seguridad. Es posible otorgar diferentes permisos sobre diferentes ramas del proyecto. Por ejemplo, estableciendo permiso universal de lectura, y permiso de escritura solo a ciertos usuarios.

Desarrollar un proyecto de software implica invertir mucho tiempo y dinero. No proteger nuestra inversión con un sistema de control de versiones es irresponsable y denota un grave desconocimiento del desarrollo de software.

¿Por qué SVN?

Si eres desarrollador seguramente conocerás CVS. Se emplea en la mayoría de proyectos comerciales, y prácticamente en todos los de código abierto. Pero últimamente se habla de un sustituto de CVS. La fundación Apache por ejemplo, ya permite que sus proyectos migren a Subversion [http://www.apache.org/dev/cvs2svn.html] si lo desean. Esta sección se explica porque es necesario el cambio.

Subversion soluciona estos problemas del CVS:

- No registra cambios en la estructura de directorios: no es posible mover, renombrar, ni copiar. Estas operaciones se consiguen eliminando y añadiendo, pero con esto perdemos el historial de cambios. Este defecto se debe a que CVS usa internamente el sistema de almacenamiento de RCS [http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?RCS] , que solo registra cambios de contenido en ficheros individuales.

- Es necesario interrumpir el acceso al repositorio para crear copias de seguridad.

- No permite "conjuntos de cambios". Cuando un desarrollador sube un conjunto de cambios, se van subiendo uno a uno, quizás al mismo tiempo que otro desarrollador hace lo mismo. Al no ser una operación atómica, nadie puede asegurar que el estado del repositorio tras su commit, sea el mismo que el estado que probó en local, y por tanto, el proyecto puede estar en un estado que nadie ha probado. Además, deshacer un conjunto de cambios requiere recorrer el repositorio entero comparando las fechas.

- Almacena ficheros binarios enteros (no sus diferencias entre versiones). Esto consume espacio en disco y ancho de banda.

- No usa la red eficientemente. Las diferencias entre versiones solo se envían desde el servidor al cliente, cuando el cliente sube sus cambios envía ficheros enteros.

- El código fuente es difícil de mantener. CVS comenzó como un conjunto de scripts shell que usaban RCS e implementaban algoritmos desarrollados entre los años 60-80. El resultado actual es producto de sucesiones de parches, y no tiene un diseño fácil de entender o mejorar. Esto dificulta su evolución. La idea de crear un nuevo CVS desde cero, surgió en la propia compañía que ofrecía soporte comercial para el CVS.

Aumenta la funcionalidad:

- Objetivo: mejorar y ampliar las prestaciones de CVS.

- Registra cambios en la estructura de directorios (permite mover y renombrar sin perder el historial). Subversion no usa RCS, sino un sistema virtual de ficheros versionado sobre una base de datos.

  El uso de la base de datos Berkeley [http://www.sleepycat.com/] permite aislamiento, atomicidad, recuperación de datos, integridad, backups en caliente, y concurrencia sin necesidad de usar

ficheros de lock. Con Berkeley DB no se pueden editar los ficheros a mano como con CVS, pero eso tampoco es necesario porque el repositorio no se corrompe. Subversion resulta más fiable que CVS sobre sistema de ficheros transaccional. Nota: Los logs de la base de datos llegan a ocupar bastante espacio, pero existe una herramienta para eliminarlos (**svnadmin**).

- Commits atómicos, se realizan todos o ninguno. Las transacciones atómicas permite identificar conjuntos de cambios. Cuando un desarrollador sube un conjunto de ficheros lo hace en una transacción atómica, de modo que todos los ficheros se etiquetan con un número de revisión en el repositorio. La atomicidad también impide que el repositorio quede en estado no compilable porque la red cae durante la subida de cambios.
- Servidor y cliente intercambian diferencias entre versiones. Al enviar una nueva versión nunca es necesario transmitir ficheros enteros.
- Pueden añadirse propiedades arbitrarias (pares de clave y valor) a ficheros y directorios.
- Interoperabilidad con WebDAV. Es posible acceder al repositorio con cualquier software que soporte dicho protocolo ("Web Folders" de Windows XP, Photoshop, etc.).
- Apache + SSL puede usarse con firewalls y proxys.
- MIME types y detección automática de ficheros binarios.
- Permite operar directamente sobre el repositorio, sin copia local.
- Permite backups en caliente.

Y mejora el rendimiento y diseño:

- Protocolo WebDAV/DeltaV para el protocolo de red.
- Arquitectura de red mejorada: Apache 2.0, envío de diffs binarios entre cliente y servidor, datos comprimidos con mod_deflate.
- Se basa en APIs C bien definidas y documentadas. CVS en cambio, se construyo mediante sucesiones de parches.
- Usa la biblioteca Apache Portable Runtime, que permite portar la capa de red a varios sistemas operativos.
- El cliente es una pequeña aplicación que usa una biblioteca de alto nivel.
- Versiona todos los ficheros guardando comprimidas sus diferencias.
- No es necesario duplicar el código en el repositorio para crear ramas. Subversion usa copia perezosa, solo se crea un nuevo fichero cuando es modificado. Mientras tanto, el fichero de la nueva rama, esta implementado como un enlace al fichero original. En contraste, CVS tarda por ejemplo 40 minutos en crear un tag de release en el servidor de GCC. Es decir, en Subversion la operación es O(1), mientras que en CVS es O(n) (lineal respecto al tamaño del repositorio).
- No es necesario conexión a red para ciertas operaciones: **status**, **diff**, **revert**. Esto se debe a que la copia local contiene una copia del fichero original presente en el repositorio. Este comportamiento ahorra ancho de banda a costa de mayor espacio en disco.

Aprender Subversion desde CVS

La filosofía de trabajo es la misma entre los dos sistemas, los comandos básicos (**checkout**, **add**, **commit**, **update**, etc.) también. Pero algunas cosas han cambiado (a mejor):

- Subversion proporciona comandos con nuevas funcionalidades: **copy**, **move**, **merge**, **resolve**, **mkdir**, **propset**, **propget**, **proplist**, **propdel**, **propedit**, **revert**, **switch**, **info**.
- Numeración de versiones. Con Subversion los números de versión son globales para todo el repositorio. No hay un número de versión por fichero.
- Autentificación:
  - CVS usa un modo de autentificación propio, con un protocolo propio. Este protocolo debe usarse con SSH tunneling [http://www.tigris.org/nonav/scdocs/ddSSHGuideCygwin.html], o implementaciones CVS de kerberos o GSS para que sea seguro.
  - Subversion se usa normalmente con HTTP + autentificación BASIC o SSL. También incluye un servidor propio que podemos usar con SSH tunneling.
- Dejan de existir los conceptos de módulo, ramas, y etiquetas, como entidades separadas:
  - Ramas y etiquetas (branches y tags): Internamente no existen tales conceptos, si quieres una rama la creas a partir de un enlace a un número de revisión del código (*se comparte el historial*, no es necesario copiar el proyecto entero). Una etiqueta es una rama a la que no se le añaden más cambios.
  - Módulos: En Subversion solo existen directorios, no hay concepto de módulo.
- Palabras clave. Las palabras clave de CVS se expanden automáticamente. Esto fuerza al usuario a desactivar este comportamiento explícitamente, corriendo peligro de destruir ficheros binarios. En Subversion, este comportamiento debe ser activado explícitamente.
- En Subversion existen varios protocolos de acceso: HTTP, SVN, file:///.

Si ya conoces CVS, cambiar a Subversion es sencillo.

Instalación

La distribución de Subversion incluye un cliente remoto (**svn**), un servidor (**svnserve**), y varias utilidades. En este documento se ilustra el uso de Apache como servidor porque permite más posibilidades que svnserve. La única ventaja de svnserve es que se configura más fácilmente que Apache. Por ejemplo, asegurar la comunicación en Apache requiere certificados SSL, mientras que con svnserve basta usar SSH.

Instalar Subversion

Puedes encontrar paquetes para Windows, Debian, RedHat, SuSE, Mandrake, pkgsrc ({Net,Open,Free}BSD, Linux, Solaris, etc.), Solaris, y OSX, a través de la página http://subversion.tigris.org/project_packages.html.

Aquí se describe la instalación en Debian y Windows. Tras este paso podremos usar Subversion desde consola. Más adelante activaremos el acceso HTTP que nos permitirá compartir el repositorio con otros desarrolladores, y acceder con programas que usan este protocolo (Eclipse, TortoiseSVN, etc.).

Debian

Existe un paquete subversion [http://packages.debian.org/subversion] versión 1.0.5-1 unstable y 1.0.0-1 testing. La última versión liberada del proyecto a 27 de julio es 1.1.0-rc1. Hay una versión 1.0.5-0 portada a woody aquí [http://people.debian.org/~cjwatson/subversion-woody/].

Aviso

Las versiones hasta la 1.0.5 (incluida) presentan una vulnerabilidad en el módulo mod_authz_svn. En la versión 1.0.5-1 ya esta solucionado.

Aviso

La instalación unstable y testing requiere libc6.1 (2.3.2.ds1-11) [http://packages.debian.org/unstable/base/libc6.1], una biblioteca usada por multitud de programas del sistema. Dependiendo de lo actualizado que este tu sistema, podría requerir una actualización masiva.

En cualquier caso, la instalación es la misma:

```
debian:/mnt# wajig install subversion

Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  db4.2-util libapr0 libneon24 libsvn0 libswig1.3.21
Suggested packages:
  subversion-tools
The following NEW packages will be installed:
  db4.2-util libapr0 libneon24 libsvn0 libswig1.3.21 subversion
0 upgraded, 6 newly installed, 0 to remove and 233 not upgraded.
Need to get 1304kB of archives.
After unpacking 3711kB of additional disk space will be used.
Do you want to continue? [Y/n]
Get:1 http://ftp.us.debian.org unstable/main db4.2-util 4.2.52-16 [60,6kB]
Get:2 http://ftp.us.debian.org unstable/main libapr0 2.0.50-5 [124kB]
Get:3 http://ftp.us.debian.org unstable/main libneon24 0.24.6.dfsg-1 [80,4kB]
Get:4 http://ftp.us.debian.org unstable/main libswig1.3.21 1.3.21-5 [160kB]
```

Nota

wajig es un frontend simplificado para varios comandos `apt-*` y **dpkg**. Prueba **apt-get install wajig** y **wajig help**. O si no quieres complicarte, usa el **apt-get install** de toda la vida.

Opcionalmente podemos instalar subversion-tools, unos scripts que facilitan la administración del repositorio:

```
debian:/# wajig install subversion-tools

Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  libconfig-inifiles-perl python2.3-subversion rcs
The following NEW packages will be installed:
  libconfig-inifiles-perl python2.3-subversion rcs subversion-tools
0 upgraded, 4 newly installed, 0 to remove and 233 not upgraded.
Need to get 1095kB of archives.
After unpacking 3346kB of additional disk space will be used.
```

Windows

De la página de descarga [http://subversion.tigris.org/servlets/ProjectDocumentList?folderID=91], hay dos versiones:

- "Windows installer with the basic Subversion Win32 binaries". Versión autoinstalable.
- svn-win32-x.x.x.zip. Versión en zip, probablemente más reciente. Para instalarla es necesario

descomprimir, añadir el directorio bin al PATH, y establecer esta variable de entorno: `APR_ICONV_PATH=D:\programas\Subversion\iconv`.

Aviso

Como regla general, evita los directorios con espacios cuando uses versiones Windows de herramientas Unix. Casi siempre deberían funcionar sin problemas, yo lo he hecho por probar y funciona, pero no es recomendable.

Aviso

Si usas Windows XP necesitas el Service Pack 1.

Tras instalar deberíamos poder abrir una consola tras la instalación y operar con Subversion en local:

```
D:\>svn
Type 'svn help' for usage.
```

Crear un repositorio

El repositorio es el árbol de directorios donde Subversion almacena nuestros ficheros y cambios.

Para crear un repositorio vacío usamos la herramienta de administración **svnadmin**:

- Debian:

  ```
  mkdir -p /var/local/repos
  svnadmin create /var/local/repos

  # doy permisos al servidor web
  chown -R www-data:www-data /var/local/repos
  ```

- Windows:

  ```
  # en Windows quiero crearlo en d:\repositorio, así que me pongo en d:\ y ejec
  svnadmin create repositorio
  ```

Si no hay problemas tampoco veremos aparecer mensajes por consola. Esto puede sorprender a usuarios Windows, pero en Unix, muchos comandos se comportan así.

El repositorio tiene este aspecto en el sistema de ficheros de Debian:

```
debian:/# ls -la /var/local/repos
total 36
drwxr-sr-x    7 www-data www-data     4096 2004-07-27 20:00 .
drwxrwsr-x    3 root     staff        4096 2004-07-27 20:00 ..
drwxr-sr-t    2 www-data www-data     4096 2004-07-27 20:00 conf
drwxr-sr-t    2 www-data www-data     4096 2004-07-27 20:00 dav
drwxr-sr-t    2 www-data www-data     4096 2004-07-27 20:00 db
-r--r--r--    1 www-data www-data        2 2004-07-27 20:00 format
drwxr-sr-t    2 www-data www-data     4096 2004-07-27 20:00 hooks
drwxr-sr-t    2 www-data www-data     4096 2004-07-27 20:00 locks
-rw-r--r--    1 www-data www-data      376 2004-07-27 20:00 README.txt
```

Lo que vemos son ficheros y directorios de una instancia de la base de datos Berkeley. Esta es una novedad frente al CVS: los proyectos no se guardan en el sistema de ficheros, sino en una base de datos. Esto asegura que cualquier transacción que realicemos

- es atómica,
- sus resultados consistentes,
- aislados (independientes de otras transacciones),
- y duraderos (su efecto es permanente).

O dicho de otro modo, cualquier operación con Subversion posee las propiedades ACID (Atomic, Consistent, Isolated, Durable) propias de toda base de datos.

> **Nota**
>
> La información esta más segura en una Base de Datos que en un sistema de ficheros. Pero si llegados a este punto te asusta no ver físicamente tus ficheros, espero que eso refuerce en ti la sana costumbre de hacer backups. :-)

NUNCA será necesario operar manualmente sobre el contenido del repositorio. En su lugar debemos usar las herramientas que proporciona Subversion. Por tanto, no realizaremos más visitas a este directorio.

Acceso remoto con Apache

Apache es un servidor web que podemos configurar para acceder al repositorio usando el protocolo HTTP. El acceso remoto con Apache ofrece mayor flexibilidad que **svnserve**, porque podemos usar todos los módulos y scripts para Apache, como por ejemplo ViewCVS (que sí, también funciona con Subversion).

Instalar Apache

Debian

En Debian Apache viene en sabores moderno, tradicional, y suicida:

- apache2-mpm-worker [http://packages.debian.org/unstable/net/apache2-mpm-worker]: High speed threaded model for Apache2.

  The worker MPM provides a threaded implementation for Apache2. It is considerably faster than the traditional model, and is the recommended MPM. Worker generally is a good choice for high-traffic servers because it has a smaller memory footprint than the prefork MPM.

- apache2-mpm-prefork [http://packages.debian.org/unstable/net/apache2-mpm-prefork]: Traditional model for Apache2.

  This Multi-Processing Module (MPM) implements a non-threaded, pre-forking web server that handles requests in a manner similar to Apache 1.3. It is appropriate for sites that need to avoid threading for compatibility with non-thread-safe libraries. It is also the best MPM for isolating each request, so that a problem with a single request will not affect any other. It is not as fast, but is considered to be more stable.

- apache2-mpm-perchild [http://packages.debian.org/unstable/net/apache2-mpm-perchild]: Experimental High speed perchild threaded model for Apache2.

  Perchild is the grown up, mac daddy version of suexec for apache2. Rather than execute a cgi script as a given user, perchild forks a process for each vhost, then su's to the correct user/group for that vhost. Each process then uses a thread model

similar to that of the worker mpm. THIS MPM IS NOT CURRENTLY EXPECTED TO WORK CORRECTLY, IF AT ALL. IT IS UNDER VERY HEAVY DEVELOPMENT. This mpm is still highly experimental, and should be used with care.

Yo he escogido la opción moderna:

```
debian:/# wajig install apache2-mpm-prefork

Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  apache2-common
Suggested packages:
  apache2-doc
The following NEW packages will be installed:
  apache2-common apache2-mpm-prefork
0 upgraded, 2 newly installed, 0 to remove and 233 not upgraded.
Need to get 1062kB of archives.
After unpacking 3830kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Para que apache transforme las operaciones HTTP de los clientes de Subversion, en operaciones sobre el repositorio, es necesario instalar el módulo libapache2-svn [http://packages.debian.org/unstable/net/libapache2-svn]:

```
debian:/# wajig install libapache2-svn

Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  libapache2-svn
0 upgraded, 1 newly installed, 0 to remove and 233 not upgraded.
Need to get 57.0kB of archives.
After unpacking 229kB of additional disk space will be used.
```

Windows

Para acceder al repositorio usando un cliente HTTP (por ejemplo el plugin subclipse de Eclipse [www.eclipse.org]), necesitaremos instalar Apache2.0.48 (no valen el 1.x ni el 2.0.46). La instalación es trivial, descarga y ejecuta apache_2.0.48-win32-x86-no_ssl.msi [http://apache.rediris.es/httpd/binaries/win32/apache_2.0.48-win32-x86-no_ssl.msi]. Siempre se instala en un directorio Apache2, por tanto, si lo quieres en `d:\httpd\Apache2`, indica `c:\httpd` como directorio de instalación. Por defecto se añadirá como servicio y al iniciar Windows veremos aparecer un monitor de Apache la bandeja, al lado del reloj.

Configurar Apache

Aquí se explica como configurar el módulo para que opere con nuestro repositorio, y como restringir el acceso usando autentificación básica.

Apache permite otras posibilidades que aquí no se mencionan, como por ejemplo, comprobar los passwords contra base de datos, crear grupos de usuarios, permitir el acceso de los usuarios solo a ciertas ramas del repositorio, autentificar por IP, y otras. Para más información sobre restricciones de acceso podemos consultar el apartado seguridad en algún tutorial acerca de Apache [http://httpd.apache.org/docs-2.0/misc/tutorials.html].

Debian

Para configurar el módulo mod_dav_svn, edita `/`

`etc/apache2/mods-available/dav_svn.conf`. Comenta estas líneas:

```
#<Location /svn>
#</Location>
```

y añade estas otras:

```
<Location /repos>
    DAV svn
    SVNPath /var/local/repos
</Location>
```

La directiva Location [http://httpd.apache.org/docs/mod/core.html#location] de Apache asocia el bloque que contiene a una URL determinada. Aqui por ejemplo, la configuración del módulo dav_svn queda asociado a la dirección http://localhost/repos. Perfectamente podríamos haber usado `Location /cualquier/otra/ruta`, y dejar el repositorio accesible en http://localhost/cualquier/otra/ruta.

Si acabas de instalar Apache, probablemente ya este en ejecución (prueba **ps -A | grep apache2**). Reinícialo para que lea los cambios:

```
debian:/# /etc/init.d/apache2 restart
Restarting web server: Apache2.
```

Vamos a http://127.0.0.1/repos, y vemos el repositorio:

```
Revision 0: /

Powered by Subversion version 1.0.5 (r9954).
```

Windows

Primero copiamos el fichero `mod_dav_svn.so` de la instalación de Subversion (`C:\Program Files\Subversion\httpd\mod_dav_svn.so`) al directorio `modules` de la instalación de apache (`C:\Program Files\Apache2\modules\`).

En el fichero `httpd.conf` de la instalación de Apache busca la siguiente línea:

```
#LoadModule dav_module modules/mod_dav.so
```

y cambiala por esto:

```
LoadModule dav_module modules/mod_dav.so
LoadModule dav_svn_module modules/mod_dav_svn.so
```

Suponiendo que nuestro repositorio este en `d:\repositorio` ponemos esto al final de `httpd.conf`:

```
<Location /repos>
  DAV svn
  SVNPath d:/repositorio
</Location>
# observa que hemos usado d:/ y no d:\
```

La directiva Location [http://httpd.apache.org/docs/mod/core.html#location] de Apache asocia el bloque que contiene a una URL determinada. Aqui por ejemplo, la configuración del módulo dav_svn queda asociado a la dirección http://localhost/repos. Perfectamente podríamos haber usado `Location /cualquier/otra/ruta`, y dejar el repositorio accesible en http://localhost/cualquier/otra/ruta.

Reiniciamos apache usando el monitor. Si el monitor indica que la operación ha fallado, conviene abrir una consola en el directorio bin de la instalación de Apache (en mi ordenador es

`C:\Program Files\Apache\bin`) y ejecutar lo siguiente:

```
D:\programas\Apache2\bin>apache -e debug
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module access_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module actions_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module alias_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module asis_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module auth_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module autoindex_modul
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module cgi_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module dav_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module dav_svn_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module dir_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module env_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module imap_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module include_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module isapi_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module log_config_modu
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module mime_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module negotiation_mod
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module setenvif_module
[Wed Mar 03 22:22:54 2004] [debug] mod_so.c(290): loaded module userdir_module
```

Vemos que entre otros ha cargado el módulo dav_svn_module. En este punto la consola se queda bloqueada, Apache espera peticiones. Ya deberías poder acceder por red, comprueba la dirección http://127.0.0.1/repos/. Deberías ver algo como

Figura 1. Repositorio vacío

Pulsamos Ctrl+C para cerrar la consola, y arrancamos Apache desde el monitor, o desde la consola de servicios de Microsoft. Desgraciadamente, a mí me falló en ambos casos el servicio, y cuando consulte el visor de eventos del sistema encontré este mensaje:

```
The Apache service named  reported the following error:
>>> Cannot load C:/Program Files/Apache/modules/mod_dav_svn.so into server:
The specified module could not be found.
```

Gracias a mis dotes adivinatorias y a que examiné `mod_dav_svn.so` con DependencyWalker [http://www.dependencywalker.com/], descubrí que el módulo DAV_SVN no encontraba los ficheros `libapr.dll`, `libaprutil.dll`, y `libhttpd.dll`, que son las bibliotecas de la APR (Apache Portable Runtime Library). Para arreglarlo las copié de `C:\Program Files\Apache\bin` al directorio `modules`.

Por desgracia seguía sin funcionar, y DependencyWalker no daba más pistas. En la lista de correo leí que debía copiar los ficheros `libdb42.dll`, `libeay32.dll`, `ssleay32.dll`, del `bin/` de la instalación de Subversion al directorio `modules` de Apache.

Como aun así seguía sin funcionar copie todas las .dll de subversion/bin al modules/ de Apache, en algún caso, sobreescribiendo la versión de Apache que había copiado antes. No es un método muy científico pero funcionó. Supongo que el módulo de Subversion prefería las librerías de Apache contra las que había sido compilado.

Por cierto, no encontré referencias a este problema en el manual de Subversion.

Restringir el acceso

Aquí se explica como configurar Apache para restringir el acceso usando autentificación básica (basic authentication) o SSL.

La autentificación básica no es segura porque envía los passwords a través de la red codificados en base64 (un algoritmo que proporciona muy poca protección). Estos passwords pueden capturarse con un sniffer y desencriptarse con paciencia. Por tanto:

- Si el servidor esta expuesto a Internet, convendrá usar SSL para operaciones de escritura y asegurar Apache [http://www.securityfocus.com/infocus/1694] contra ataques.
- Si lo vamos a usar en un entorno seguro (una intranet) y solo nos interesa identificar a los usuarios que realizan cada cambio en el repositorio, basta la autentificación básica.

Debian

Volvemos a editar la configuración del módulo (`/etc/apache2/mods-available/dav_svn.conf`):

```
<Location /repos>
    DAV svn
    SVNPath /var/local/repos
    AuthType Basic
    AuthName "Subversion repository"
    # fichero de passwords
    AuthUserFile /etc/subversion/passwd
    # permito las opciones GET PROPFIND OPTIONS REPORT a usuarios anonimos
    <LimitExcept GET PROPFIND OPTIONS REPORT>
        # pero requiero autentificación para el resto de operaciones (las de es
        Require valid-user
    </LimitExcept>
</Location>
```

Luego crea un usuario, por ejemplo 'jano':

```
debian:/# htpasswd2 -c /etc/subversion/passwd jano
New password:
Re-type new password:
Adding password for user jano
```

Y reinicia:

```
debian:/# /etc/init.d/apache2 restart
Restarting web server: Apache2.
```

Windows con Autentificación básica

El procedimiento es casi el mismo que veíamos con Debian. Creamos el fichero de passwords:

```
D:\programas\Apache2\bin>htpasswd.exe -nb jano secreto > usuarios.txt
Automatically using MD5 format.
```

Añadimos la configuración necesaria a `httpd.conf`:

```
<Location /repos>
  DAV svn
```

```
  SVNPath d:/repositorio

  # Autentificación básica.
  AuthType Basic
  AuthName "Repositorio Subversion"
  AuthUserFile D:/programas/Apache2/bin/usuarios.txt

  # Solo permitimos usuarios identificados en el fichero de passwords.
  #require valid-user

  # Solo permitimos usuarios identificados en el fichero de passwords
  # con nombre 'jano' o 'alicia'.
  #require user jano alicia

  # La lectura es libre pero solo permitimos usuarios identificados
  # en el fichero de passwords para otras operaciones.
  <LimitExcept GET PROPFIND OPTIONS REPORT>
    Require valid-user
  </LimitExcept>

</Location>
```

Y reiniciamos:

```
net restart apache
```

Acceso remoto con svnserve

svnserve es un servidor incluido en Subversion cuya función es acceder al repositorio usando un protocolo TCP/IP propio. Las URLs son similares a las que usamos en Apache, pero comienzan por `svn://` o `svn+ssh://`. svnserve consume menos recursos que Apache, pero es menos potente.

Instalar svnserve

Si ya has instalado Subversion, ya tienes el programa svnserve. El propio comando nos muestra las opciones de arranque:

```
$ svnserve --help
Usage: svnserve [options]

Valid options:
  -d [--daemon]            : daemon mode
  --listen-port arg        : listen port (for daemon mode)
  --listen-host arg        : listen hostname or IP address (for daemon mode)
  --foreground             : run in foreground (useful for debugging)
  -h [--help]              : display this help
  -i [--inetd]             : inetd mode
  -r [--root] arg          : root of directory to serve
  -R [--read-only]         : deprecated; use repository config file
  -t [--tunnel]            : tunnel mode
  -X [--listen-once]       : listen once (useful for debugging)
```

Windows

No es aconsejable usar svnserve en Windows porque necesita fork para gestionar el acceso concurrente, pero Windows no lo implementa. Desconozco que problemas pueden darse. En Windows podemos usar este wrapper [http://dark.clansoft.dk/~mbn/svnservice/] para instalarlo como servicio.

Si tenemos CygWin instalado, instalamos **xinetd** en la sección Net y hacemos

```
/usr/sbin/inetd --install-as-service
net start inetd
```

También podemos arrancar inetd desde consola manualmente en depuración con

```
/usr/sbin/inetd -d
```

Para SSH también necesitaremos OpenSSH y OpenSSL. Instalalos y comprueba que que estan ahi:

```
$ ssh -version
OpenSSH_3.8p1, SSH protocols 1.5/2.0, OpenSSL 0.9.7d 17 Mar 2004
Bad escape character 'rsion'.
```

Consulta tus dudas en la lista de correo de Subversion. Por ejemplo en http://www.contactor.se/~dast/svnusers/.

Linux

Por defecto, el servidor escucha en el puerto 3690 reservado por la IANA [http://www.iana.org/assignments/port-numbers] para el protocolo svn.

En modo independiente la consola queda bloqueada:

```
$ svnservce -d
```

Para lanzarlo con inetd:

```
$ svnserve -i
( success ( 1 2 ( ANONYMOUS ) ( edit-pipeline ) ) )
```

y añadimos esto a `/etc/inetd.conf`:

```
svn stream tcp nowait svnowner /path/to/subversion/bin/svnserve -i
```

Importar un proyecto

Ya tenemos el repositorio funcionando en red. Para iniciar un proyecto simplemente creamos un subdirectorio donde iremos añadiendo ficheros. Durante el resto del documento usare un miniproyecto casero llamado "awpool".

```
# Creo un proyecto llamado "awpool" en el repositorio
svn mkdir http://127.0.0.1:80/repos/awpool -m "Creo el proyecto"
```

Si ya tienes un proyecto y quieres importarlo necesitareis una copia libre de directorios `CVS/` (si es que usábamos CVS). Lo más sencillo es eliminar los directorios e importar en el CVS.

Si vais a conservar el repositorio CVS convendrá etiquetar la versión a partir de la cual has migrado a Subversion. Por ejemplo, vamos a etiquetarlo como "release1":

```
# Suponiendo que aun no tengo en mi disco el CVS, lo hago ahora
mkdir /home/jano/tmp
cd /home/jano/tmp
cvs co awpool

# y lo etiqueto como 'release1'
cvs tag release1
```

En mi caso no quería perder mi copia local del CVS, así que me baje esa release exacta exportando. En CVS exportar significa bajarte el código sin control de versiones, o sea, sin los directorios `CVS/`:

```
mkdir /home/jano/release1
cvs export -r release1 awpool
```

De un modo u otro, deberíamos acabar con el código listo para ser importado.

```
# Creo un directorio para la versión principal del proyecto.
# A veces se hacen versiones (branches) de un proyecto, y lo principal se pone
# en un directorio que por convención se llama trunk.
svn mkdir http://127.0.0.1:80/repos/awpool/trunk -m "version principal"

# importo el código que copie (o exporte) antes en /home/jano/release1,
# o sea, el directorio con el código a importar, y lo meto en trunk
cd /home/jano/release1
svn import . http://127.0.0.1:80/repos/awpool/trunk -m "importacion inicial"

# en Windows me coloque dentro del directorio con el proyecto a importar y tecl
# svn import . http://127.0.0.1:80/repos/awpool/trunk -m "importacion inicial"
```

La operación anterior tarda un rato, veremos desfilar todos los ficheros que están importándose. Si olvidaste eliminar la versión compilada de tú programa (ooops!!) también se añadirá y tendrás que borrarla luego. Cuando termina, puedes conectarte por web para ver el resultado, o hacer checkout (descargarlo) desde otro directorio:

```
mkdir /home/jano/awpool
cd  /home/jano/awpool
svn checkout http://127.0.0.1:80/repos/awpool/trunk .
```

La copia local generada tiene un subdirectorio `.svn` junto a cada directorio. Es el equivalente a los antiguos `CVS/`, pero su contenido tiene un formato distinto.

Guardar la versión principal en un subdirectorio trunk no es obligatorio, pero ese y otros, son los elementos acostumbrados en un repositorio:

- `trunk/` Por convención suele usarse para el código principal del proyecto. Cuando la gente dice "get the head of the trunk", se refieren a que te bajes la última versión de este directorio.
- `branches/` Ramas del desarrollo. Copias del código principal con cambios experimentales.
- `tags/` Versiones del código listas para empaquetar como entregables.

Uso

Como puedo ...?

Bajar el proyecto (checkout)

Primero nos bajamos el proyecto que creamos antes, o algún otro que encontremos por ejemplo en http://svn.apache.org/repos/asf/. Yo bajé el que cree antes:

```
# creamos un directorio para albergar el proyecto
mkdir /home/jano/awpool

# nos situamos en él
cd /home/jano/awpool

# y bajamos el proyecto
svn checkout http://127.0.0.1:80/repos/awpool/trunk .
```

Esta copia local contiene un subdirectorio `svn` en cada directorio, lo que permite que sea gestionada con el cliente de Subversion.

Actualizar la copia local (update)

Si queremos actualizar nuestra copia local con los cambios que otros usuarios hayan enviado al repositorio usamos **update**:

```
# por defecto es recursivo
svn update
At revision 4.

# para actualizar solo un directorio
svn -N update
At revision 4.

# para un fichero especifico
svn update GenericsTest.java
At revision 2.
```

Resolver conflictos

Acabo de modificar un fichero en local, y voy a actualizarlo para luego hacer commit (no quiero machacar los cambios de nadie). Pero al actualizar Subversion muestra una C de conflicto junto al fichero:

```
debian:/# svn update .
C  GenericsTest.java
Updated to revision 5.
```

Además han aparecido varias versiones del fichero en disco:

```
debian:/# ls
GenericsTest.java         # el fichero con mis cambios y marcas de conflicto
GenericsTest.java.mine    # el fichero con mis cambios
GenericsTest.java.r4      # la revisión antes de mis cambios
GenericsTest.java.r5      # la última revisión que acabo de actualizar
```

Todo esto ocurre porque desde la última vez que bajé este fichero, alguién más lo cambio, y al actualizarme con el servidor he recibido un trozo de código que entra en conflicto con mis cambios. O dicho de otra manera, un cambio que de haberse aplicado sobreescribiría mis propios cambios, y yo perdería información. Afortunadamente siempre podemos actualizar sin miedo, porque Subversion suspende la fusión de cambios y nos advierte del conflicto.

Vamos a abrir el fichero para ver que aspecto tienen las "marcas de conflicto":

```
   private static void print(Type t) {
      if (t instanceof TypeVariable) {
         print((TypeVariable)t);
<<<<<<< .mine
      } else if (t instanceof WildcardType) {
         print((ParameterizedType)t);
=======
      } else if (t instanceof WildcardType) {
         print((WildcardType)t);
>>>>>>> .r2
      } else {
         System.out.println(t);
      }
   }
```

Observa que hay un trozo `<<< .mine mi-código ===` y un trozo `=== código-revisión-2 >>> .r2`. Se corresponden a mi código y al código de la revisión 2 que acabo de bajarme.

Para solucionar el conflicto puedo:

- Descartar mis cambios locales, o los cambios que baje del servidor. Esto se hace sobreescribiendo el fichero original GenericsTest.java por alguna de las copias que han aparecido.
- Solucionar el conflicto editando a mano el fichero:

```
private static void print(Type t) {
   if (t instanceof TypeVariable) {
      print((TypeVariable)t);
   } else if (t instanceof WildcardType) {
      print((WildcardType)t);
   } else {
      System.out.println(t);
   }
}
```

  Oops! el cambio correcto no era el mío, asi que he dejado lo que ya había.

Tras solucionar el conflicto elimino los ficheros sobrantes y hago commit:

```
rm *.mine *.r4 *.r5
svn ci GenericsTest.java
```

Como ves, normalmente se soluciona intuitivamente editando el fichero a mano. Si quieres más detalles consulta el manual de Subversion:
http://svnbook.red-bean.com/html-chunk/ch03s05.html#svn-ch-3-sect-4.4.

Enviar modificaciones (commit)

Al contrario que con CVS, los commit son atómicos. Es decir, cuando subes varios ficheros llegan en bloque (todos o ninguno), y si dos desarrolladores suben sus cambios a la vez primero se graba todo el bloque de uno, y luego el del otro. Los cambios nunca se mezclan.

Es buena práctica pensar por adelantado en el mensaje de log que vamos a usar, y hacer commits de cambios relacionados. De ese modo es más fácil revisar los cambios buscando errores. Conviene definir una variable **SVN_EDITOR=kedit** (o **SVN_EDITOR=notepad**) para usarlo como editor por defecto. De no hacerlo, tendremos que añadir el flag **-m 'mensaje'** para el obligado comentario que debe acompañar todo commmit.

Para enviar modificaciones locales al servidor:

```
# envía las modificaciones encontradas en todos los subdirectorios
$svn commit

# actualiza todos los *.txt del directorio actual y lo que haya en el directori
$svn commit *.txt src
```

El cliente svn lanzará el editor configurado, escribimos el mensaje de log, grabamos, cerramos el editor, y el commit se realiza. Un ejemplo de la salida en Windows:

```
$svn commit *.txt src
Sending        STATUS.txt
Sending        src/java.head/overview.html
Transmitting file data ..
Committed revision 4.
```

Si alguno de los ficheros que enviamos ha sido modificado en el repositorio, es obligado que actualicemos dichos cambios (usando **update**) antes de hacer **commit**. Esto evita que machaquemos los cambios de otros usuarios.

Examinar cambios (status)

El comando status muestra las modificaciones que hemos hecho en la copia local desde el último checkout o commit. Es útil hacerlo antes de un commit para hacernos idea de lo que poner en el log. Por defecto, el cliente no accede a la red durante la ejecución de este comando.

```
svn status
```

Este ejemplo muestra los posibles estados devueltos:

```
$ svn status
  L     ./abc.c               # fichero bloqueado
M       ./bar.c               # modificado localmente
 M      ./baz.c               # modificado localmente en sus propiedades, no en
?       ./foo.o               # fichero ignorado
!       ./foo.c               # fichero borrado por el usuario u otro programa
~       ./qux                 # versionado como fichero pero es directorio (o vi
A  +    ./moved_dir           # añadido con historial de procedencia
M  +    ./moved_dir/README    # añadido con historial y tiene modificaciones loc
D       ./stuff/fish.c        # preparado para borrado
A       ./stuff/loot/bloo.h   # preparado para añadir
C       ./stuff/loot/lump.c   # conflictos con un update
    S   ./stuff/other_dir     # directorio cambiado a rama
G       ./foo.c               # modificado localmente e incorporados cambios rem
```

Recuperar una revisión (update)

Al hacer un update/commit, nos indica un número de versión. Para machacar una copia local con una revisión anterior hacemos:

```
$svn update -r3
U  STATUS.txt
U  src\java.head\overview.html
Updated to revision 3.

# y de vuelta a la revisión 4
$svn update -r4
U  STATUS.txt
U  src\java.head\overview.html
Updated to revision 4.
```

Hemos deshecho los cambios que se hicieron tras la revisión 3. Observa que ahora he colocado una opción a la derecha del comando en vez de a la izquierda. En Subversion no importa donde colocar las opciones.

Mover un fichero o directorio (move)

Es posible mover o renombrar ficheros o directorios sin perder el historial de cambios. El comando **move** puede usarse para renombrar o para mover de un directorio a otro.

```
# primero se marca para su renombrado
svn move LEEME.txt README.txt
A         README.tx
D         LEEME.txt

# comprobamos los cambios pendientes
svn status
D      LEEME.txt
A  +   README.txt

# y ejecutamos el cambio
svn commit -m "Fichero renombrado"
Deleting       LEEME.txt
Adding         README.txt
```

```
Committed revision 5.
```

También se puede operar directamente en el servidor usando HTTP:

```
$svn move http://localhost/repos/awpool/trunk/README.txt
          http://localhost/repos/awpool/trunk/README
```

Ver logs (log)

Con SVN es posible obtener mensajes de log:

```
# para todas las revisiones:
svn log

# idem con información adicional
svn log --verbose

# para la revisión 4
svn log -r 4

# para un fichero concreto (deberemos estar en
# el directorio del fichero o especificar el path completo)
svn log README.txt
```

También podemos mostrar log operando directamente sobre el servidor, sin necesidad de habernos bajado nada:

```
$svn log http://localhost/repos/awpool
```

Ver el autor de cada cambio en un fichero (blame)

Si con log veíamos el historial de revisiones de un fichero, con blame vemos el fichero entero y el autor de cada cambio sobre cada parte del fichero:

```
svn blame http://ruta/al/fichero
```

Crear un changelog (log)

```
# entro en el proyecto
cd /home/jano/work/trunk

# Creo un fichero de nombre arbitrario Changelog con todos los mensajes de log
# desde la revisión 1 (-r1) hasta la última revisión (HEAD).
svn log -r1:HEAD >> ChangeLog

# Hago add porque es un fichero nuevo
svn add ChangeLog

# subo el fichero para añadirlo a la distribución
svn commit ChangeLog -m 'Actualizo ChangeLog'
```

Eliminar un fichero o directorio (rm, remove, del, delete)

```
# Se hace como en CVS, primero se marca para eliminarlo
svn rm fichero

# y luego se hace commit para eliminarlo en el repositorio.
# Si es un directorio, la eliminación se realizará recursivamente por defecto.
svn commit
```

```
# del, delete, remove, y rm, son seudonimos del mismo comando
```

Si ejecutamos el comando sobre una URL la eliminación será inmediata. Después de ejecutar rm y commit, los ficheros/directorios, habrán desaparecido de la copia local y el HEAD del repositorio, pero permanecerán sus revisiones anteriores.

Añadir un fichero o directorio (add)

```
# añadir un directorio recursivamente (por defecto no se añaden subdirectorios)
svn add directorio -R
svn commit

# añadir un fichero
svn add fichero
svn commit
```

Deshacer cambios (revert)

```
# no funciona recursivamente a menos que usemos -R
svn revert
```

Deshace los cambios locales, incluyendo modificaciones, cambios de propiedades, y planificados de cambios (add, rm, para los que aun no hayamos hecho commit).

Crear diffs (diff)

Los diffs (diferencias) son los datos diferentes entre dos versiones de un fichero (o conjuntos de ficheros), de modo que aplicando esa diferencia es posible pasar de un fichero a otro. Un diff se aplica usando el comando GNU **patch**. SVN usa el formato unificado diff [http://www.gnu.org/software/diffutils/manual/html_node/Unified-Format.html].

```
# Obtener los diffs de las modificaciones de la copia local
svn diff

# diff entre las revisiones 14 y 18 para ficheros *.java y *.xml
svn diff -r14:18 *.java *.xml > /home/jano/tmp/r14-a-r18.patch

# diff entre un fichero local "foo.c" y su última revisión
svn diff --revision HEAD foo.c

# lo mismo que el anterior pero sin tener en cuenta nuestras modificaciones loca
svn diff --revision BASE:HEAD foo.c
```

El parametro **--revision** también acepta fechas, no solo números de revisión.

Crear una rama

Una buena práctica de la ingeniería del software es mantener los proyectos en estado compilable y funcional en todo momento. Pero cuando necesitamos introducir un cambio extenso en un proyecto, corremos el riesgo de "dejar en obras" gran parte de la aplicación. Esto suele traducirse en la interrupción de funcionalidades críticas y molestias en el trabajo de otras personas.

Para evitar problemas conviene crear una rama (branch). Una rama es una copia del código principal que evoluciona por separado, pero que tiene un pasado común respecto al origen a partir del cual se creó. En otras palabras, un proyecto propio con el cual experimentar. Más adelante, cuando hayamos completado nuestro trabajo, fundiremos nuestros cambios con la rama principal del proyecto. SVN proporciona un comando **merge** para integrar una rama en otra.

Nota

> Suele hablarse de ramas principal y secundaria, pero internamente no existe esa distinción, son simplemente dos directorios del repositorio. De hecho, para Subversion no existe el concepto de rama, es solo un nombre que nosotros empleamos.

Un tema problemático con la fusión de cambios entre ramas, es que mientras trabajamos en una rama, el resto del equipo sigue añadiendo cambios a la principal. Sería deseable que la rama donde trabajamos integrase automáticamente los cambios producidos en el código principal para evitar discrepancias al integrar los cambios en el código principal. Esta funcionalidad esta prevista en alguna de las versiones posteriores a la 1.0, por ahora el soporte de merge es similar al del CVS.

En el siguiente ejemplo tenemos la rama principal en `/cygdrive/d/desarrollo/awpool/trunk` y crearemos una rama en `/cygdrive/d/desarrollo/awpool/branch2-awpool2`. Por cierto, estas rutas tan raras son de mi instalación de Cygwin en Windows. Podría haber usado **d:\desarrollo\...** o cualquiera ruta válida en mi disco duro.

```
cd /cygdrive/d/desarrollo/awpool
svn copy trunk branch-awpool2
svn commit -m "creo la rama branch-awpool2"
```

La copia no duplica el código en disco, tal como hace CVS, sino que crea enlaces apuntando al código existente. Lo único que ocupará espacio en disco serán los cambios que introduzcamos a partir de aquí.

Crear una etiqueta

En los sistemas de control de versiones, un tag (etiqueta) es una etiqueta que marca un punto en la evolución del código. Para Subversion un tag es un nombre que damos a una determinada revisión de un directorio del repositorio.

Para crear una etiqueta también se usa **copy**:

```
cd /cygdrive/d/desarrollo/awpool
svn copy trunk release-1.0
svn commit -m "marco el proyecto con la etiqueta release-1.0"
```

Aunque pueda sorprender a los veteranos del CVS, esto es coherente con lo que veíamos en el apartado "Branches". El comando **copy** crea un enlace a una revisión del código. Si no añadimos cambios es como si hubiéramos etiquetado una versión del código. Si añadimos cambios es como si tuviéramos una rama. Internamente, Subversion no conoce los conceptos etiqueta y rama, solo entiende de directorios del repositorio.

Crear un release en tar.gz (export)

Vamos a crear un fichero `trunk.tar.gz` que contenga un directorio trunk con todo el proyecto:

```
cd /home/jano/tmp
svn export http://localhost/repos/awpool/trunk
tar cf trunk.tar trunk
gzip -9 trunk.tar
```

Copia de seguridad (dump)

Vamos a crear una copia de seguridad del repositorio y todas sus versiones en un fichero .gz. Ocupará lo que ocupa el release que hicimos antes, más lo que ocupe la información de las revisiones.

```
svnadmin dump /home/jano/proyectos/svn | gzip -9 > dump.gz
* Dumped revision 0.
* Dumped revision 1.
* Dumped revision 2.
* Dumped revision 3.
```

```
* Dumped revision 4.
* Dumped revision 5.
```

Usando el parametro **-r** podemos volcar a partir de una revisión concreta.

Si luego queremos restaurar la copia hacemos:

```
gunzip -c dump.gz | svnadmin load /home/jano/proyectos/svn
```

o en Windows:

```
svnadmin load d:\ruta\al\repositorio < "fichero.descomprimido.dump"
```

Obtener ayuda (help)

```
# muestra los comandos disponibles
svn help

# muestra ayuda sobre un comando concreto
svn help comando
```

Desbloquear la copia de trabajo (cleanup)

Si estamos realizando cambios a la copia de trabajo e interrumpimos con Ctrl+C o se produce un cuelgue de todo el sistema, es posible que la próxima vez que accedamos nos diga que parte de nuestra copia de trabajo está bloqueada (locked). Esto se selecciona ejecutando **svn cleanup**, que terminará las operaciones interrumpidas y eliminará los bloqueos.

Eliminar un repositorio (obliterate)

Por ahora no se puede :-(. Esta capacidad se llamará "obliterate" y esta planeada para después de la versión 1.0. Lo máximo que puedes hacer ahora es

- eliminar todos los ficheros y dejar vacío el repositorio (el historial permanecerá),
- o eliminar físicamente todo el repositorio, crear uno nuevo, y restaurar las copias de seguridad.

Compartir el repositorio entre sistemas operativos

Compartir el repositorio entre sistemas operativos es posible, pero no una práctica oficialmente soportada.

Al intentar acceder a un mismo repositorio Windows en FAT32 desde linux aparece este mensaje:

```
svn: Unable to open an ra_local session to URL
svn: Unable to open repository 'file:///mnt/win5/repositorio'
svn: Berkeley DB error while opening environment for filesystem
/mnt/win5/repositorio:
Invalid argument
```

El problema es que la base de datos de Berkeley usa shared memory (un método de comunicar dos procesos usando una misma región de memoria), y almacena los punteros en ficheros cuyo formato es diferente entre windows y linux.

La solución es eliminar los ficheros `repositorio/db/__db.00*` y ejecutar **svnadmin recover /ruta/al/repositorio**.

En el futuro otra opción será usar un repositorio fsfs [http://web.mit.edu/ghudson/info/fsfs] (**svnadmin create --fs-type /ruta/al/repositorio**). Se trata de un repositorio que usa el sistema de ficheros

en vez de la base de datos de Berkeley. Es una funcionalidad de Subversion que a día de hoy (agosto de 2004) aun no es estable. Consulta la lista de desarrollo [http://subversion.tigris.org/servlets/ProjectMailingListList] de Subversion para más información.

Propiedades

Subversion permite adjuntar pares arbitrarios nombre/valor a ficheros y directorios. Estos metadatos se llaman propiedades en Subversion. Las propiedades son versioneadas, es decir, se almacenan sus sucesivos valores igual que se hace con los ficheros.

Las propiedades son modificaciones locales que no serán permanentes hasta que ejecutemos **commit**. Como cualquier modificación, las propiedades pueden verse con **diff**, **status**, **revert**.

Establecer una propiedad (propset)

Para establecer una propiedad se usa **svn propset nombre valor fichero**. Ejemplo:

```
svn propset autor jano EstoNoMeEstaPasandoAMiException.java
svn propset proposito "ilustrar el uso de propiedades" DeQueVasException.c
```

También se pueden adjuntar ficheros binarios como propiedades:

```
svn propset diagrama -F uml.png Command.java
```

Leer una propiedad (propget)

Para leer una propiedad usa **svn propget nombre fichero**. Ejemplo:

```
svn propget color Command.java
svn propget height Command.java
```

Listar propiedades (proplist)

```
# ver nombres de propiedades
svn proplist Command.java

# ver nombres y valores de propiedades
svn proplist Command.java --verbose
```

Borrar una propiedad (propdel)

Para borrar una propiedad usa **svn propdel nombre fichero**.

Editar propiedades (propedit)

Las propiedades se editan ejecutando **svn propedit nombre fichero**. Esto hará que se abran en el editor de texto que hayamos definido. Es útil sobretodo para introducir propiedades con retornos de carro.

svn:executable

Solo para ficheros. Si esta propiedad existe, significa que el fichero es ejecutable. El valor de esta propiedad es irrelevante.

svn:mime-type

Sirve para indicar el tipo mime. Normalmente no es necesario porque los comandos add e import, incorporan un algoritmo de detección.

Si establecemos el tipo mime a algo que no comience por text/, Subversion supone que es un fichero binario, si no, supone que es de texto simple.

Cuando recuperamos un fichero usando el navegador, el módulo mod_dav_svn lo envía con el valor del mime-type en la cabecera Content-type.

svn:ignore

Se aplica a un directorio y permite establecer patrones de ficheros que serán ignorados por SVN. Si queremos crearla, es conveniente usar el comando propget. Esto es similar al `.cvsignore` del CVS.

```
svn propedit svn:ignore .
*.bak
*~
```

Si nos da un error como este:

```
svn: Working copy is not up-to-date
svn: Commit failed (details follow):
svn: Cannot commit propchanges for directory '/home/jano/...
```

Debemos actualizar el directorio: **svn update directorio** antes de hacer commit a ese directorio.

svn:eol-style

Sirve para forzar los códigos de control de los saltos de línea los ficheros. Si desarrollamos desde varios sistemas operativos, podemos emplear esta propiedad para hacer siempre checkout en el salto de línea nativo del sistema que usemos.

Por defecto, los ficheros en el repositorio tienen los mismos saltos de línea que en la copia de trabajo. Los valores posibles son:

- LF
- CR
- CRLF
- native: los ficheros aparecen siempre con los saltos de línea propios del sistema operativo local. En el repositorio los ficheros se guardan con LF.

svn:keywords

Subversion sustituye cinco palabras clave, aquí se listan junto con su sinónimo:

- `LastChangedDate`, `Date` Fecha del último cambio, por ejemplo: 2002-07-22 21:42:37 -0700 (Mon, 22 Jul 2002)
- `LastChangedRevision`, `Rev` Última revisión en que fue cambiado el fichero. Por ejemplo: 18
- `LastChangedBy`, `Author` Nombre del último usuario que la cambio.
- `HeadURL`, URL de la última versión de este fichero
- `Id` Sumario de las palabras anteriores. Por ejemplo: bar 148 2002-07-28 21:30:43 epg, que significa que el fichero bar cambio en la revisión 148, por el usuario epg, en la fecha 2002-07-28 21:30:43.

Para activar las palabras clave en un fichero se usa la propiedad svn:keywords.

```
svn propset svn:keywords "palabra1, palabra2, ..." fichero
```

Ejemplo:

```
# Esto hace que en el fichero foo.c se inserte el valor de dichas palabras,
# allá donde aparezcan las cadenas $Date$, $LastChangedDate$, $Author$, y $Last
svn propset svn:keywords "Date Author" foo.c
```

Para obtener la propiedad svn:keywords se usa

```
svn propget svn:keywords fichero
```

Lo mínimo que necesita saber un cliente

```
# 1. checkout
svn checkout http://svn.example.com/repos/proyecto/trunk

# 2. actualizar cambios hechos por otros
svn update

# 3. ¿que cambios voy a enviar?
svn status

# 4. enviar mis cambios
svn commit –m "cambie tal y tal cosa"

# 5. poner ficheros y directorios bajo el control de subversion
svn add fichero1 fichero2 subdirectorio
svn commit –m "añado dos ficheros y un directorio"

# 6. borrar ficheros y directorios
svn delete fichero1 fichero2 subdirectorio
svn commit –m "borre lo de antes porque no me molaba"
```

Software relacionado

GUIs

Tabla 1. Clientes con GUI para Subversion

Nombre	Lenguaje	Portable	Licencia
jsvn [http://jsvn.alternatecomputing.com]	Java, usando el cliente de consola	Sí	Academic Free License [http://www.opensource.org/licenses/afl-2.0.php]
RapidSVN [http://rapidsvn.tigris.org/]	C++	Sí, componentes gráficos nativos	estilo Apache [http://www.opensource.org/licenses/apachepl.php]
Supervision [http://supervision.tigris.org/]	Java, cliente de consola	Sí	GPL [http://opensource.org/licenses/gpl-license.php]
svnup [http://svnup.tigris.org/]	Java: Swing, ligaduras JNI	Sí	estilo Apache [http://www.opensource.org/licenses/apachepl.

Nombre	Lenguaje	Portable	Licencia
			php]
SvnX [http://www.lachoseinteracti-ve.net/en/community/subversion/svnx/features/]	Java: Swing, ligaduras JNI	Sí	estilo Apache [http://www.opensource.org/licenses/apachepl.php]
TortoiseSVN [http://tortoisesvn.tigris.org/]	Cocoa framework y Objective-C	No, Mac OS X 10.3	??

Bibliotecas

Subversion esta compuesto por bibliotecas de funciones C que cumplen interfaces claramente definidos. Por ejemplo, existen varias bibliotecas que cumplen el interfaz de acceso al repositorio:

- libsvn_ra_dav: administra accesos usando la extensión WebDAV/DeltaV de HTTP 1.1. Es el usado para comunicarse con Apache.
- libsvn_ra_local: administra accesos mediante sistema local de ficheros (protocolo file:///)
- libsvn_ra_svn: administra accesos usando el protocolo propio SVN. Se usa con el servidor svnserve.

En total hay once bibliotecas que pueden agruparse en tres capas:

- Repositorio.
- Acceso al repositorio.
- Cliente.

En la programación de Subversion también se ha usado la biblioteca Apache Portable Runtime (APR). La APR se creó originalmente para proporcionar una API genérica independiente del sistema operativo. Para la gestión de memoria en Subversion, APR proporciona "pools de memoria", reservas de memoria de las que el programa va tomando, en vez de solicitar al sistema operativo cada pequeño fragmento que necesita.

Para usar Subversion desde lenguajes que no sean C, se proporcionan ligaduras SWIG [http://www.swig.org/]. SWIG es un compilador que permite enlazar código C/C++/Objective-C con lenguajes de script como Perl, Python, Ruby, y Tcl. Funciona en Unix, Win32 con Cygwin [http://www.cygwin.com/]/Mingw [http://www.mingw.org/], y Macintosh. Para más detalles consulta la página SWIG de compatibilidad en plataformas [http://www.swig.org/compat.html]. Otra opción para usar Subversion es ejecutar y comunicarse directamente con un cliente C desde un lenguaje no C, pero la precisión y riqueza de datos siempre será mayor usando las librerías nativas.

Si programas en Java puedes usar la Java High Level API (javahl [http://svn.collab.net/repos/svn/trunk/subversion/bindings/java/javahl/]) para comunicar Java y Subversion. javahl usa JNI para proporcionar cualquiera de las operaciones disponibles en el cliente de consola. Existen versiones Unix y Windows. Hay varios proyectos que usa javahl en http://svnup.tigris.org/.

Plugins

Tabla 2. Plugins para Subversion

Nombre	Programa	Comentarios	Licencia
AnkhSVN [http://ankhsvn.tigris.org/]	Visual Studio .NET	0.4.1, pantallazos [http://ankhsvn.tigris.org/screenshots.html]	Apache 1.1 [http://opensource.org/licenses/apachepl.php]
SCPlugin [http://scplugin.tigris.org/]	Finder OSX	Beta	MIT [http://opensource.org/licenses/mit-license.php]
Subclipse [http://subclipse.tigris.org/]	Eclipse 2.1.x, 3.0 No funciona en WASD5.	Windows, Linux, OSX. Emplea ligaduras JNI.	Apache 1.1 [http://opensource.org/licenses/apachepl.php]
SVNup [http://subclipse.tigris.org/svnant.html]	IDEA	Java: Swing, ligaduras JNI. También funciona como GUI.	estilo Apache [http://www.opensource.org/licenses/apachepl.php]
SVNAnt [http://subclipse.tigris.org/svnant.html]	Ant	Usa JNI (svnjavahl.dll en windows) o línea de comando.	Apache 1.1 [http://opensource.org/licenses/apachepl.php]

Subclipse 3 + Eclipse 3

En Windows es posible usar Subclipse contra un repositorio remoto sin instalar Subversion.

En Linux o OSX necesitaremos instalar primero Subversion. Subclipse emplea ligaduras JNI (bindings javahl [http://svn.collab.net/repos/svn/trunk/subversion/bindings/java/javahl/]). Los bindings se enlazan dinámicamente con las fuentes de Subversion, esto significa que cada versión de Subclipse se crea con respecto a una versión de Subversion, y puede dar problemas si no se emplean las versiones correspondientes.

Con conexión a red, el mejor modo de instalarlo es añadiendo el sitio de actualización de Subclipse. Esto nos permitirá actualizar cómodamente a futuras versiones. Para instalarlo haz lo siguiente:

- Ve a Help > Software Updates > Find and install.
- Selecciona Search for new features to install, y pulsa Next.
- Pulsa New Remote Site... y escribe:
  - Name: Subclipse
  - URL: http://subclipse.tigris.org/update
- Expande el nuevo nodo Subclipse, selecciona Subclipse Plugin, y pulsa Next.
- Continua pulsando Next, aceptando la licencia, aceptando el certificado, y pulsa Finish.
- Nos pide que reiniciemos eclipse, contestamos Yes.

En una LAN con más usuarios, conviene descargar el sitio de actualización de Subclipse [http://subclipse.tigris.org/servlets/ProjectDocumentList?folderID=2240] y colocarlo en un servidor local o directorio compartido. En la página de descarga se explica como [http://subclipse.tigris.org/files/documents/906/14882/README.localsite.txt]. Es el mismo procedimiento de antes solo que apuntaremos al servidor/directorio local.

Scripts

Tabla 3. Scripts para Subversion

Nombre	Lenguaje	Portable	Licencia	Sumario
CVS2SVN [http://cvs2svn.tigris.org/]	Python CGI	Sí, usa SWIG	Apache 1.1 [http://opensource.org/licenses/apachepl.php]	Crea un repositorio Subversion a partir de uno CVS.
SubWiki [http://subwiki.tigris.org/]	Python CGI	Sí, usa SWIG	Academic Free License [http://www.opensource.org/licenses/afl-2.0.php]	Wiki basado en subversion. Aun no hay entregables.
ViewCVS [http://viewcvs.sf.net/]	Python	Sí	ViewCVS License [http://viewcvs.sourceforge.net/license-1.html], (similar a la licencia MIT [http://opensource.org/licenses/mit-license.php])	Navegador del repositorio.
WebSVN [http://websvn.tigris.org/]	PHP	Sí	GPL [http://opensource.org/licenses/gpl-license.php]	Navegador del repositorio.

CVS2SVN

Permite crear un repositorio Subversion a partir de otro CVS. Funciona casi siempre, o sea, que existe algún caso en que no funciona 100% correcto. Para usarlo necesitamos instalar Python y ViewCVS (ViewCVS contiene un módulo para interpretar ficheros RCS). Su funcionamiento es simple:

```
svnadmin create /nuevo/repositorio
cvs2svn.py -s /nuevo/repositorio /cvs/repos
```

Hay más detalles en codehaus.org [http://wiki.codehaus.org/general/HowToConvertACVSRepositoryToSubversion].

ViewCVS

El soporte para Subversion en ViewCVS solo existe en el CVS. Lo he probado en Windows y el repositorio es navegable, pero ciertas operaciones producen errores que no he investigado. Si a alguien le interesa ir más allá debería empezar por consultar el error en la lista de correo de ViewCVS [http://mailman.lyra.org/mailman/listinfo/viewcvs-dev].

Windows

Suponemos instalados Apache y Subversion.

1. Instalo Python [http://www.python.org/], por ejemplo usando el ActivePython [http://www.activestate.com/Products/Download/Register.plex?id=ActivePython] (17Mb) de ActiveState y siguiendo la guía de instalación

[http://aspn.activestate.com/ASPN/docs/ActivePython/2.3/UserGuide/install.html#windows_(x86)]. También podeis instalar el de python.org (Python-2.3.3.exe [http://www.python.org/ftp/python/2.3.3/Python-2.3.3.exe], 9Mb), pero al instalar ViewCVS más adelante pide unas librerías que podéis obtener bajando las extensiones win32 de Mark Hammond [http://starship.python.net/crew/mhammond/] (pywin32-200.win32-py2.3.exe [http://prdownloads.sourceforge.net/pywin32/pywin32-200.win32-py2.3.exe?download], 3.16Mb). Yo he probado con el de Python.org. Si quieres las extensiones 2.4 necesitarás haber instalado Python 2.4 y tener la dll msvcrt71.dll en tu sistema, que puedes conseguir instalando el framework .Net (23Mb), o aqui [http://www.frenzy.hu/prog/down/crt_mfc.zip], o en Google.

2. Instalo mod_python [http://httpd.apache.org/modules/python-download.cgi] para Apache, versión 3.x. Yo usé mod_python-3.1.3.win32-py2.3.exe [http://apache.rediris.es/httpd/modpython/win/3.1.3/mod_python-3.1.3.win32-py2.3.exe] (119Ks). Hago la instalación por defecto, me pregunta donde esta Apache, y aparece un mensaje indicando como configurarlo, cosa que hago. Solo se trata de añadir esta línea en `httpd.conf` junto al resto de sentencias LoadModule:

   ```
   LoadModule python_module modules/mod_python.so
   ```

   Si además queremos probar que funciona podemos usar las instrucciones de http://www.modpython.org/live/current/doc-html/inst-testing.html. Personalmente paso.

3. Ahora necesitamos el HEAD del ViewCVS porque por ahora no hay release con soporte subversion. Suponiendo CVS instalado, los comandos son

   ```
   # password en blanco
   cvs -d :pserver:anonymous@cvs.sourceforge.net:/cvsroot/viewcvs login
   cvs -d :pserver:anonymous@cvs.sourceforge.net:/cvsroot/viewcvs co viewcvs
   ```

   Voy al directorio donde lo he bajado y ejecuto

   ```
   python viewcvs-install
   ```

   e instalo en un directorio cualquiera, por ejemplo `d:\programas\viewcvs`.

4. Modifico el `http.conf` de apache para incluir lo siguiente:

   ```
   ScriptAlias /viewcvs "d:/programas/viewcvs/www/mod_python/viewcvs.py"
   <location /viewcvs>
       AddHandler python-program .py
       PythonPath "[r'd:\\programas\\viewcvs\\lib']+sys.path"
       PythonHandler apache
       PythonDebug On
   </location>
   ```

   Observa que `d:\programas\viewcvs` debe ser el directorio donde instalaste viewcvs en el paso anterior.

5. Ya solo queda modificar `d:\programas\viewcvs\viewcvs.conf`:

   ```
   [general]
   # no voy a usar cvs
   # cvs_roots = cvs: /home/cvsroot

   # ruta a mi repositorio
   svn_roots = main: d:/repositorio

   # nombre del root por defecto, aqui solo tengo uno :-)
   default_root = main
   ```

6. Bajo el módulo svn para python (svn-win32-1.0.6_py.zip) [http://subversion.tigris.org/files/documents/15/14891/svn-win32-1.0.6_py.zip] y lo descomprimo. Copio los directorios `libsvn/` y `svn/` al directorio `d:\programas\Python23\Lib`, suponiendo python instalado en

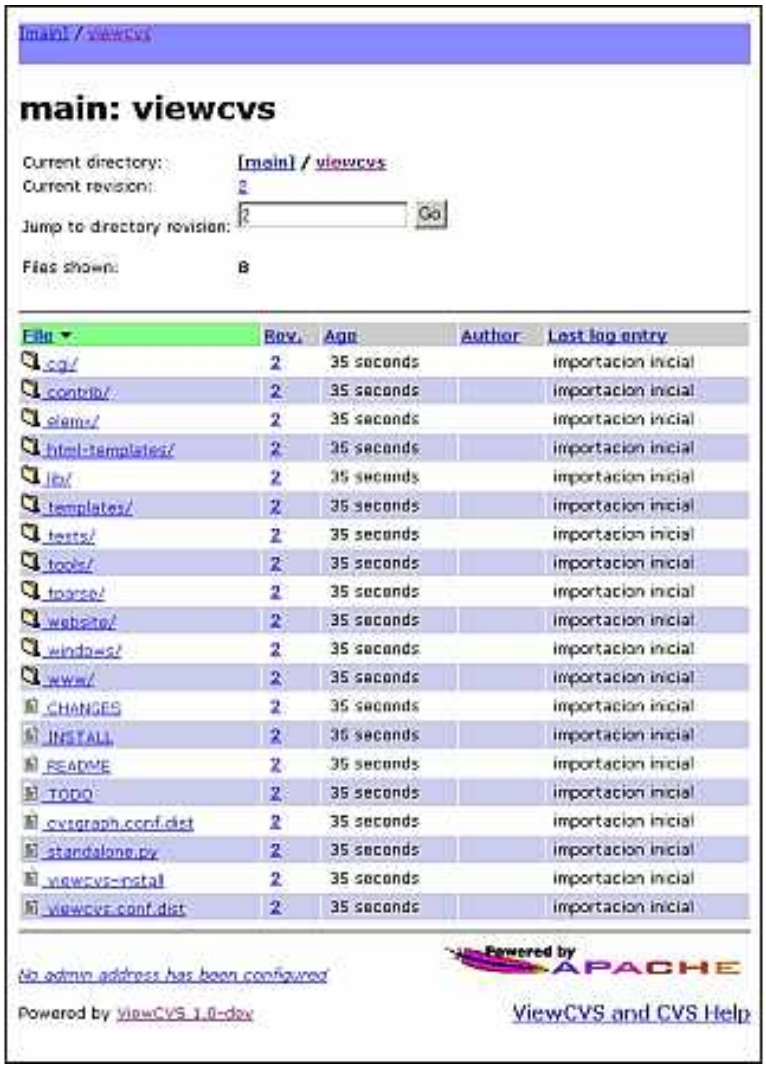
`d:\programas\Python23`.

7. Si tienes CygWin instala **diffutils**, **enscript**, **gzip**, **libiconv**, **libintl**, **rcs**, **sed**, **zlib** en versiones CygWin [http://www.cygwin.com/], GNUWin32 [http://gnuwin32.sourceforge.net/], o minGW [http://www.mingw.org/]. Puedes prescindir de este paso por ahora, el repositorio resultante será navegable, si intentamos hacer un diff y no tenemos rcs dirá algo como esto:

```
error: (2, 'CreateProcess', 'The system cannot find the file specified.')
```

8. La instalación ha terminado. Reinicio Apache, voy a http://127.0.0.1/viewcvs y he aquí el repositorio:

Figura 2.



Mirando con lupa esta imagen observamos varias cosas de interes:

- Cuando navegas aparece un número de revisión, y si pulsas en el enlace aparecen todos los ficheros pertenecientes a ese commit. Recuerda que los números de versión son globales para todo el repositorio, es decir, coinciden con el número de commits realizados.
- Todos los ficheros tienen la misma marca de tiempo y comentario. Es porque Subversion usa transacciones y constan como añadidos en el mismo instante.

- La dirección del administrador no esta configurada. Quizás nos convenga hacerlo luego. Si queremos un servidor local de correo podemos usar James [http://james.apache.org] de Apache.

Para poner colorines establece `use_enscript = 1` en `viewcvs.conf`. No lo he probado.

Para mostrar gráficamente las revisiones hay que instalar **cvsgraph** [http://www.akhphd.au.dk/~bertho/cvsgraph/], que podemos bajar de http://www.akhphd.au.dk/~bertho/cvsgraph/. Es útil sobre todo con ramas. No lo he probado.

WebSVN

WebSVN es un script PHP que usa comandos Unix. Para aumentar el rendimiento cachea el contenido. Desconozco que política de cache sigue, ni que rendimiento tendría en un sitio en Internet.

Los comentarios siguientes son de una instalación en Windows, pero se podrían adaptar sin problema a Linux. Si ya hemos instalado Subversion y Apache hacemos esto:

- Descargamos PHP [http://www.php.net/downloads.php] (7,26Mb), descomprimimos y añadimos su ruta al PATH.
- Ignoramos los comentarios del tipo "he leído por ahí que PHP no es estable en Apache2".
- Instalamos PHP en Apache2 según esta guía [http://www.php.net/manual/en/install.apache2.php] de php.net. O seguimos estas instrucciones (Windows):
  - Copiamos `php/php5ts.dll` a `windows/system32`.
  - Añadimos estas líneas al `httpd.conf` de Apache:

```
ScriptAlias /php/ "d:/programas/php"
AddType application/x-httpd-php .php
Action application/x-httpd-php "/php/php.exe"

LoadModule php5_module "d:/programas/php/php5apache2.dll"
AddType application/x-httpd-php .php
```

- Opcionalmente instalamos los comandos diff, enscript, gzip, sed, por ejemplo con CygWin [http://www.cygwin.com]. Aunque no instalemos estos comandos igualmente podremos navegar por el repositorio, pero si por ejemplo no instalamos diff, no podremos hacer diff, etc.

Ahora instalamos webSVN:

- Descargamos webSVN [http://websvn.tigris.org/servlets/ProjectDocumentList] y lo descomprimimos en `apache/htdocs` o donde quiera que apunte la la directiva DocumentRoot del `httpd.conf` de Apache.
- Renombramos `websvn/include/distconfig.inc` a `config.inc` y lo editamos como se indica en el propio fichero. Yo descomenté estas líneas:

```
$config->setServerIsWindows();
$config->setSVNCommandPath("d:\\tools\\svn\\bin");
$config->setEnscriptPath("c:\\cygwin\\usr\\bin\\enscript");
$config->addRepository("repositorio", "d:\\repositorio");
```

Ya podemos visitar http://localhost/websvn/index.php para ver nuestro repositorio.

Figura 3. websvn

Otros

Tabla 4. Scripts para Subversion

Nombre	Lenguaje	Portable	Licencia	Sumario
CruiseControl [http://cruisecontrol.sourceforge.net]	Java	Sí, requiere Java	BSD [http://opensource.org/licenses/bsd-license.php]	Proporciona *Integración Continua* [http://www.martinfowler.com/articles/continuousIntegration.html].
CruiseControl.NET [http://ccnet.thoughtworks.com/]	.Net	Requiere .Net	ThoughtWorks [http://ccnet.thoughtworks.com/?page=license.html]. Similar a la BSD [http://opensource.org/licenses/]	Proporciona *Integración Continua* [http://www.martinfowler.com/articles/continuousIntegration.html]. Es la versión .Net de CruiseControl.

Nombre	Lenguaje	Portable	Licencia	Sumario
			bsd-license.php].	

Enlaces

Página de enlaces en subversion.tigris.org: http://subversion.tigris.org/project_links.html

Subversion para no desarrolladores

Esta sección es una introducción a Subversion para no desarrolladores. Cuando tengas una idea aproximada de como funciona, te invito a que leas el documento completo.

¿Qué es?

- Subversion es un software destinado a facilitar el trabajo en equipo sobre un conjunto de ficheros.
- Cada usuario puede participar desde un ordenador conectado a Internet o una red local.
- Los usuarios se conectan al servidor para intercambiar modificaciones.
- El servidor recuerda todos los cambios realizados sobre un fichero y permite recuperar versiones anteriores.
- El acceso al servidor esta protegido mediante contraseñas.
- Es posible realizar copias de seguridad de toda la información.
- Subversion es gratuito.

¿Como funciona?

- Subversion se compone de un programa "servidor" y otro "cliente".
- El servidor contiene una copia maestra de la información a compartir.
- Los usuarios usan el cliente para descargar la información existente en el servidor.
- Cuando un usuario realiza un cambio, lo envía al servidor para que otros usuarios puedan descargarlo.
- El servidor guarda los ficheros dentro de una base de datos (no son visibles en el sistema de ficheros).

¿Qué requiere?

- Hay versiones para Windows y cualquier sistema basado en Unix.
- Puede instalarse como servidor independiente o como módulo de Apache.
- Consume pocos recursos.

- Una instalación básica solo requiere conocimientos a nivel de usuario del sistema operativo.

Un ejemplo de sesión de trabajo

Este es un ejemplo de trabajo con Subversion usando la línea de comando:

```
# Descargo (checkout) el proyecto del servidor usando
# la URL que me dió el administrador del repositorio.
svn checkout http://servidorDeSVN/repos/miProyecto

# Ahora tengo una copia del proyecto en mi ordenador y puedo trabajar en ella.
# Si quiero modificar ficheros no tengo más que editarlos.

# Si por ejemplo decido añadir ficheros o directorios hago esto:
svn add fichero1 fichero2 directorio3

# Si quisiera eliminar, haría esto otro:
svn delete ficheroA directorioB

# En cualquier momento puedo revisar los cambios que he hecho en mi copia local
svn status

# Al terminar, envío mis cambios al servidor informando de lo que he hecho.
svn commit –m "he modificado tal y cual, añadido tal y borrado cual"

# Actualizo mi copia local con los cambios subidos por otros usuarios.
svn update
```

Subversion proporciona integración con entornos de desarrollo como Eclipse, JBuilder, u otros. Normalmente ejecutaremos las mismas operaciones que acabamos de ver usando los menús de un interfaz gráfico.

Índice