Curso Acelerado De Subversion (Segunda Parte)

Adrian Kosmaczewski

2007-01-02

antes que nada fijate que todo lo que te explique anteriormente anda.

empecemos.

primero armate una carpeta con archivos varios, en tu escritorio. pone imagenes, archivos php, javascript, html, carpetas con mas cosas adentro, lo que se te ocurra, con cuantos subniveles que quieras. cualquier disposicion estara bien. lo unico que te pido es que llames esa carpeta "proyecto", simplemente (aunque podria ser cualquier nombre, yo usare ese nombre en este texto, y asi sera mas facil seguirme; pero podria llamarse "pirulo" tranquilamente).

tambien te pedire que pongas un archivo "readme.txt" en esa carpeta, para mostrarte como se manejan los cambios en los archivos. asegurate que ese archivo tenga ya un poco de texto. tambien agregate un archivo mas, que se llame, "forradas.txt", con algo de texto adentro.

bueno, ahora tenes tu proyecto listo, tenes la primera version de tu laburo (estamos imitando un ritmo de trabajo tipico con subversion). podrias estar escribiendo un nuevo libro, un software, dibujando con el GIMP, lo que quieras. cualquier "proyecto" puede ser versionado con subversion. es decir, escribiste un par de parrafos, tenes una idea de la estructura del libro, o ya tiraste un par de lineas de codigo y pensas que esto puede evolucionar. entonces, tenes ganas de "versionarlo" para poder volver atras en caso de hacer un moco, o para explorar diferentes maneras de hacer algo, de manera "paralela". por ejemplo, yo uso subversion para el material de mi master; de esta manera, si laburo en la portatil, o en la mac de mi escritorio, siempre tengo la ultima version de lo que hago a disposicion, sin importar donde hice la ultima modificacion.

entonces ahora vamos a crear el repositorio. yo los tengo en ~/Repositories, pero en realidad pueden estar en cualquier lugar. los pongo ahi para poder bacapearlos facilmente. supondre que los pondras en ~/Repositories tambien. asi que create una carpeta "Repositories" en ~ (si no te acordas que corno significa la tilde, fijate aca). yo usare esa carpeta durante este texto, y tambien asumire que tu nombre de usuario es xxxxxx (obviamente, cuando veas xxxxxx en el texto aqui debajo, reemplazalo por tu usuario en el mac).

crear repositorios es tan facil y barato en terminos de espacio disco que se recomienda activamente tener un repositorio por proyecto, en vez de varios proyectos en el mismo repositorio. es lo mas logico y despues veras por que es altamente recomendable.

abri terminal.app y mandate con un "cd ~/Repositories" (si tipeas "cd ~/Repos" y apretas la tecla TAB el resto de la palabra se completa automaticamente).

ahora estas en la carpeta donde estaran tus repositorios. para crear un repositorio, momento clave, tipeas algo facilisimo:

```
svnadmin create Proyecto
```

ahora si te fijas, ya sea con la linea de comando o con el Finder, veras que tenes unos 4 MB de carpetas creados adentro de una carpeta llamada "Proyecto". esa carpeta es tu repositorio. ahi adentro estan las bases de datos que guardaran las 1002423 versiones de tu proyecto. el nombre "Proyecto" es algo libre, podes hacer "svnadmin create Piruleta" y te hara otro repositorio con el nombre Piruleta.

fijate que en el repositorio "Proyecto" hay un archivo llamado Readme.txt; adentro dice claramente que:

```
This is a Subversion repository; use the 'svnadmin' tool to examine
it.  Do not add, delete, or modify files here unless you know how to
avoid corrupting the repository.

Visit http://subversion.tigris.org/ for more information.
```

bueno, como veras, dice que no toques nada ya que podrias romper todo. la carpeta del repositorio, en si, nunca se toca directamente SINO a traves de los comandos svn y svnadmin. de otra manera, no se hace nada. bueno, en realidad, si, haremos algo, muy cortito y simple, para que puedas usar facilmente el repositorio.

adentro del repositorio hay una carpeta "conf" y adentro veras un archivo "svnserve.conf"; abrilo con cualquier editor de texto (vim, textmate, cualquiera) y veras que todas las lineas tienen numerales "#" delante; son comentarios. para que ande el repositorio (por medida de seguridad, un repositorio svn no anda per se, hay que "habilitarlo", que es lo que hacemos ahora) tenes que sacar el numeral de las lineas 8 y 12, y modificar la linea 12 para que todo quede asi:

(extracto)

```
### Visit http://subversion.tigris.org/ for more information.

[general] ### These options control access to the repository for
unauthenticated ### and authenticated users.  Valid values are
"write", "read", ### and "none".  The sample settings below are the
defaults.  anon-access = write # auth-access = write ### The
password-db option controls the location of the password ###
```

```
database file.  Unless you specify a path starting with a /, ### the
file's location is relative to the conf directory.  ### Uncomment
the line below to use the default password file.  # password-db =
passwd
```

la linea 12 explicitamente da derechos de escritura a los usuarios anonimos. asi de facil. despues veremos la parte de seguridad. pero por ahora, veras que nos alcanza asi para que veas el funcionamiento. como veras, al tener todo comentado, el repositorio es seguro, ya que nadie puede hacer nada con el. esto es tambien parte de la filosofia unix, asi como tuviste que habilitar php en apache usando el httpd.conf. como veras, siempre la misma idea; a priori, las puertas a los intrusos estan cerradas, pero se pueden abrir facilmente con un poco de laburo.

me seguis hasta ahora? bueno, seguimos. ahora tenemos que crear, adentro del repositorio, una estructura de base que es recomendada por los creadores de subversion, pero no obligatoria, y que es de crear, adentro del repositorio, tres carpetas donde vamos a meter la informacion: “trunk”, “tags” y “branches”. por ahora te digo que el proyecto en si ira dentro de “trunk” (creo que significa “cajon” o “baul” en ingles). en “tags”, por ejemplo, guardaremos las diferentes versiones del proyecto para poder encontrarlas mas rapidamente (tipo “version 1.0”, etc). “tags” significa “etiquetas”. “branches”, finalmente sirve para crear lo que se dice “forks” (“tenedores”), es decir, proyectos alternativos, generalmente usados para estudiar prototipos o diferentes soluciones a un problema, de manera paralela, sin obstruir el trabajo ni el contenido de “trunk”. es algo potentisimo.

PERO ATENCION! trunk, tags y branches seran carpetas “virtuales”, y no de las que se crean con el Finder; las vamos a crear desde la linea de comando de la manera siguiente:

```
svn mkdir file:///Users/xxxxxx/Repositories/Proyecto/trunk -m "Creacion de trunk"
svn mkdir file:///Users/xxxxxx/Repositories/Proyecto/branches -m "Creacion de branches"
svn mkdir file:///Users/xxxxxx/Repositories/Proyecto/tags -m "Creacion de tags"
```

como veras, svn usa los nombres de comandos tipicos unix, en este caso mkdir, para hacer operaciones simlares dentro del repositorio. el parametro “-m” toma un string de texto como parametro, que acompaña la modificacion hecha al repositorio, para que uno pueda saber que paso. svn exige que cada vez que se hace una modificacion, se use el parametro -m para indicar de manera textual lo que se hizo. este texto tiene que ser coherente, ya que se usa para saber quien hizo que, cuando, donde, etc.

si no pones el texto, svn se queja:

```
svn: Could not use external editor to fetch log message; consider
setting the $SVN_EDITOR environment variable or using the --message
(-m) or --file (-F) options svn: None of the environment variables
SVN_EDITOR, VISUAL or EDITOR is set, and no 'editor-cmd' run-time
configuration option was found
```

tambien fijate de la sintaxis `file:///` con 3 (tres) barras oblicuas, ya que se trata de una URL, y no de un mero camino de acceso. el URL indica a svn el protocolo que tiene que usarse para acceder al repositorio (http, file, svn o svn+ssh son los mas comunes, nosotros usaremos todos salvo http).

ahora tenemos:

- tu proyecto
- el repositorio listo para recibir los datos.

llego el momento de la verdad; vamos a meter la carpeta "proyecto" del escritorio en tu repositorio:

```
cd ~/Desktop
svn import proyecto file:///Users/xxxxxx/Repositories/Proyecto/trunk -m "Import inicial"
```

y listo! todo el contenido de la carpeta ~/Desktop/proyecto esta ahora en subversion. totalmente versionadito y guardadito. para eso sirve el comando "svn import".

y ahora podes tirar a la basura la carpeta /Users/xxxxxx/Desktop/proyecto. esto es algo que puede parecer raro, pero tu carpeta /Users/xxxxxx/Desktop/proyecto no esta versionada, sino que fue usada para alimentar el repositorio. lo cual no es lo mismo. para seguir laburando en tu proyecto de manera versionada, tenes que tener una version versionada (sic), generalmente llamada "working copy" del proyecto:

```
cd ~/Desktop
svn checkout file:///Users/xxxxxx/Repositories/Proyecto/trunk proyecto
```

con este comando, le estamos diciendo a svn que nos de una copia versionada del proyecto, sacandolo de la carpeta "trunk" del repositorio file:///Users/xxxxxx/Repositories/Proyecto, en una carpeta local que se llame "proyecto". y listo! ahora tenes una version lista para trabajar en tu escritorio.

bueno, ahora si te digo, que lo jodido, ya paso. las operaciones que quedan son las mas simples. lo que hicimos hasta ahora es algo que se hace generalmente una sola vez cada tanto, y ahora viene el trabajo diario.

modificate un par de lineas de readme.txt; cualquier cosa, lo que se te ocurra. poco importa, con cualquier editor. tambien create un nuevo archivo adentro de la carpeta proyecto, que se llame piruleta.txt, y pone un poco de texto adentro.

con esto estamos simulando cambios diarios dentro del proyecto. es el trabajo cotidiano. se modifican cosas, se agregan otras. por ahora no borres nada, ya te mostrare como se hace.

ahora vamos a la linea de comandos a ver que ha pasado en el proyecto despues de un dia de trabajo:

```
cd ~/Desktop/proyecto
svn status
```

y vemos esto:

```
?      piruleta.txt
M      readme.txt
```

como veras, ? significa que hay un archivo nuevo, que svn desconoce, y otro que fue modificado. los demas archivos no aparecen en esta lista. cuando haces svn status en una copia local versionada, ves los cambios hechos hasta ese momento. muy interesante. ahora vamos a "agregar" piruleta.txt al repositorio:

```
svn add piruleta.txt
```

y de paso, vamos a borrar "forradas.txt", que despues de todo son solo forradas:

```
svn delete forradas.txt
```

y si pedimos el status,

```
svn status
```

vemos que

```
A      piruleta.txt
D      forradas.txt
M      readme.txt
```

donde A = add (agregar); D = deleted; M = modified; mas facil, imposible.

bueno, ya laburaste mucho por hoy, asi que vamos a subir los cambios al repositorio. si "checkout" es para obtener una copia de trabajo versionada, "commit" es para mandar los cambios al repositorio:

```
cd ~/Desktop/proyecto
svn commit -m "Hecho algunos cambios..."
```

en la jerga se dice "commite unos archivos hace un rato" para indicar que fueron puestos en el repositorio. como veras, siempre aparece el comando -m para indicar un texto que cuente lo que paso.

si ahora haces

```
svn status
```

veras que svn no dice nada; quiere decir que tu version es la ultima. para ver las cosas que se hicieron, podes hacer un

```
svn log readme.txt
```

y eso te muestra lo siguiente:

```
------------------------------------------------------------------------
r5 | xxxxxx | 2006-05-20 16:37:03 +0200 (Sat, 20 May 2006) | 1 line

Hecho algunos cambios...
------------------------------------------------------------------------
```

```
r4 | xxxxxx | 2006-05-20 16:25:10 +0200 (Sat, 20 May 2006) | 1 line

Import inicial
------------------------------------------------------------------------
```

obviamente las fechas cambian, depende de cuando vos hiciste las cosas. ahi ves la necesidad de

- tener un repositorio por proyecto (para que no se mezclen los mensajes de log) y
- poner un texto descriptivo cuando se realiza una modificacion del repositorio

pero bueno, saber que hubo un cambio en readme.txt entre la version actual (5) y la anterior (4) no te ayuda mucho; que habra cambiado?

```
svn diff -r 4 readme.txt
```

y eso muestra:

```
--- readme.txt  (revision 4)
+++ readme.txt  (working copy)
@@ -1,13 +1,8 @@
The files in this folder contain the definition of the MySQL database
used to store the information of the application. Its structure
-is as simple as possible, allowing minimum flexibility.
-NOTES:
-1) InnoDB tables are used to support FOREIGN KEY relationships between tables
-
-
+esta parte la agregue durante la ultima revision
2) "timestamp" type columns do not hold values of type NULL, even the table
definition says so; following MySQL documentation, "timestamp" columns
that receive a NULL store the current date and time automatically.
```

(obviamente tu texto sera distinto) pero lo importante son los signos “+” y “-” que indican las lineas de texto que se agregaron y se sacaron entre la version actual (la “working copy”) y la revision 4 (la “r” es de revision).

como veras, esto es simplemente tocar la punta del iceberg. svn es un mundo aparte, pero permite cosas realmente esotericas.

bueno, creo que por hoy es bastante; obviamente hemos visto solamente la mecanica para un solo usuario; svn es usado en varios proyectos conocidos en el mundo open source (mira la lista aca, es realmente impresionante), y ahora tambien sourceforge lo ofrece para los proyectos que alli se alojan (antes era solamente cvs).

imaginate que alguien, por ejemplo yo, modifica el repositorio durante la noche; como haces para ponerte al dia vos? bueno, a la mañana siguiente, o cuando te aviso por e-mail, haces simplemente un

```
svn update
```

con este simple comando, svn va a buscar la ultima version del repositorio, y te pone tu copia local al dia. si hubieses hecho modificaciones sin commitarlas, pero que fueron modificadas en el repositorio (una situacion mas que comun), svn se encarga de "mezclar" convenientemente las modificaciones remotas con las tuyas, y si no puede hacerlo, te pregunta. como veras, es tremendo. ese proceso de mezcla se denomina merge en la jerga de los sistemas de gestion de versiones.

finalmente, existe un proyecto open source, hecho en php, que se llama websvn que permite ver los contenidos de un repositorio con un browser. yo lo tengo instalado en casa para ver rapidamente mis repositorios, muy practico.

te dejo con un

```
svn help
svn help mkdir
svn help commit
svn help checkout
svn help update
svn help add
svn help import
svn help diff
```

para que veas las distintas opciones que tiene svn… para cada uno de los comandos que aprendiste hoy.

que te parece? subversion hoy dia esta integrado a visual studio, eclipse y a xcode, de manera muy natural, para poder "mirar" repositorios de manera simple y asi obtener copias de trabajo, enviar modificaciones, etc. muchos proyectos estan siendo "migrados" de cvs, sourcesafe y otros sistemas a subversion, ya que es tecnicamente muy superior, gratis, open source y activamente mantenido. subversion fue escrito en C, usando el "apache portable runtime" como framework para acelerar el desarrollo.

bueno, bajate SmartSVN y tendras un cliente subversion escrito en java que esta muy bueno; configuralo y paseate por tu repositorio (necesitas java 1.4.1 para que ande…)

si no podes usar smartsvn, es normal, tenes que largar primero esto en la linea de comandos

```
svnserve -d
```

y despues configura el repositorio asi:

aqui arriba estas pidiendo conectarte mediante el protocolo "svn" al repositorio, por eso necesitas el daemon svnserve (que es el que escucha el puerto 3690, el puerto de subversion) para asi poder usar el protocolo svn.

creo que con esto tendras para divertirte un rato……. :)

una vez que le tomes la mano a esto, con linea de comando o no, veras que lo necesitas absolutamente. para cualquier tipo de proyecto, y no solo para software.