El control de versiones

Guillem Borrell

Historial de revisiones

Revisión 0.1 13-Abr-2006
Versión inicial, sólo con los comandos básicos de subversion. El resto de comandos aparecerán en futuras vers

1. Algunas preguntas esenciales

1.1. ¿Qué es el control de versiones?

Se llama control de versiones a los métodos y herramientes disponibles para controlar todo lo referente a los cambios en el tiempo de un archivo.

Difícilmente un archivo de código o un documento de texto está terminado con la primera escritura; necesita cambios o reescrituras para corregir errores, modificar su contenido... A medida que el documento cambia existen dos opciones, mantener un historial de cambios o dejar que evolucione sin *memoria*. El control de versiones es un método estándar para mantener esta memoria haciendo además que sea útil para el desarrollo futuro.

En documentos sencillos como un ensayo o un pequeño programa la *memoria* no es algo esencial, pero en la escritura de un libro o un programa con centenares de páginas y una docena de manos involucradas no hay otra manera de trabajar. Esta es precisamente la palabra clave, mantener un control de las versiones de todos los archivos de un proyecto es una manera de trabajar completamente estandarizada; todas las prácticas tienen un nombre.

La buena noticia es que todo este formalismo es generosamente recompensado por el uso de sistemas de contol de versiones automáticos como CVS, Subversion, git...

1.2. ¿Cómo se especifican las versiones?

El método más habitual de asignar una versión a un documento es mediante un número o un grupo de números. No existe un modo fijo de numerar una versión, se deja al criterio de cada desarrollador. Sí existen ciertas prácticas habituales en la numeración como el uso de tres cifras o la numeración decimal.

1.2.1. Versiones `X.Y.Z`

Un método bastante habitual de numerar las versiones es utilizando dos o tres cifras decimales para indicar la importancia de los cambios realizados. El cambio de la primera cifra indica cambios más importantes que el de la segunda. El criterio más habitual es seguir las siguientes normas:

- La primera cifra (`X`) indica la versión mayor del documento. Si empieza con un cero significa que el documento aún no está listo o no cumple con los requerimientos mínimos. Cada cambio en esta cifra denota una reescritura o la incompatibilidad con versiones mayores anteriores.
- La segunda cifra (`Y`) indica la versión menor del documento. Denota cambios en el contenido o en la funcionalidad del documento pero no lo suficientemente importantes como para decir que ya no es el mismo. Cuando se estrena una versión mayor se deja la versión menor a cero pero aún así se incluye de modo que la segunda versión mayor sería la `2.0`
- La tercera cifra (`Z`) indica la segunda versión menor. Indica que el documento se ha corregido pero que no se ha añadido ni eliminado nada relevante. Cuando se estrena una versión menor, es decir, cuando la segunda versión menor es igual a cero; suele omitirse.
- Pueden añadirse recursivamente versiones menores, algunos proyectos hacen uso de ellas cuando los criterios de selección de versiones no corresponden con los definidos en los puntos anteriores. El núcleo del sistema operativo Linux utiliza una numeración de cuatro cifras.
- Otra práctica común es definir *versiones especiales* como las versiones *alpha*, *beta* o las *release candidate*. Estas versiones suelen utilizarse antes de llegar a un hito como una nueva versión menor. Por ejeplo, antes de llegar a la segunda versión mayor, la versión `2.0`, puede publicarse una primera versión previa *alpha* numerada como `2.0a` o `1.99`. La segunda versión previa se llamaría *beta* y se numeraría como `2.0b` o (por ejemplo) `1.999`. Si aún así son necesarias más versiones previas, en vez de optar por más letras griegas se numeran versiones *candidatas*; la primera sería la *release candidate 1* numerada como `2.0-rc1`

1.2.2. Versiones numeradas

Un método de numeración compatible con el anterior es numerar las versiones mediante un número decimal. La primera versión es la 1, la segundo es la 2 y así sucesivamente. Esta numeración suele utilizarse en el desarrollo cuando el control de versiones se lleva de forma automática mediante algún programa de control de versiones.

En algunos casos ambas numeraciones se combinan. Por ejemplo se está desarrollando la versión `2.1` a partir de la `2.0.5` y para el software de control de versiones (se toma como ejemplo Subversion) la última versión es la 671. Entonces la versión de desarrlollo podría ser la `2.0.5-svn671`.

1.3. ¿Qué son los repositorios y las copias de trabajo?

Todas las herramientas de control de versiones se basan en la típica comunicación servidor-cliente.

Desde el punto de vista de los documentos se podría decir que el repositorio es el código está en el servidor y la copia de trabajo en el cliente.

La copia de trabajo es entonces una imagen del contenido en el repositorio de código y es lo que se utiliza para el trabajo diario (modificar archivos existentes crear documentos nuevos...). Todos los cambios que se hagan en dicha imagen no son definitivos hasta que se suban al repositorio. El control de versiones se ve influido en gran parte por cómo se realiza esta comunicación entre copia de trabajo-repositorio.

El lector puede plantearse la necesidad de duplicar los datos del proyecto. Esta arquitectura tiene la ventaja de que pueden existir tantas copias de trabajo como sean necesarias, normalmente tantas como participantes tenga el proyecto. Cada participante tiene su copia de trabajo y es el servidor el encargado de que los cambios aportados por los desarrolladores no entren en conflicto.

1.4. ¿Cómo se organizan los documentos dentro de la copia de trabajo?

Los documentos sujetos al control de versiones deben estar organizados de un modo determinado. No sirve utilizar un directorio distinto para cada versión porque se supone que el control de versiones es automático.

La estructura de documentos suele dividirse en tres grandes directorios llamados *trunk*, *branches* y *tags*.

1.4.1. El directorio *trunk*

El directorio *trunk* o tronco es el que marca el desarrollo del proyecto, la versión principal. En proyectos pequeños en los que sólo hay una rama, ésta es el tronco. Es un criterio bastante común que contenido del tronco del repositorio, que no el de la copia de trabajo, sea funcional. Por ello se entiende que en el caso de documentación no haya ninguna sección a medias o en el caso de código que todo compile y/o funcione.

1.4.2. El directorio *branches*

El directorio *branches* o ramas es el que contiene todas las derivaciones del proyecto que contiene el tronco y que no pueden convivir en la rama principal.

Por ejemplo, supóngase que varias personas están escribiendo esta documentación a la vez y que el formato de la fuente del documento es *docbook*. Uno de los autores decide migrar el documento a LaTeX porque contiene muchas más fórmulas de lo que se esperaba en un principio. El proyecto escrito en docbook seguirá viviendo en trunk mientras que para el documento escrito en LaTeX se creará una nueva rama en el directorio branches. Cuando el nuevo proyecto supere en funcionalidades al antiguo puede que abandone branches para pasar a ser trunk.

1.4.3. El directorio *tags*

Tag podría traducirse (aunque no literalmente) por galón o hito. Cuando la versión trunk llega a una versión mayor o menor, es dercir, un estado en el que podría recibir el calificativo de *completa*; pasa al directorio tags. En él no se realizan cambios, es un almacén donde se guardan algunas versiones de valor histórico.

2. Control de versiones con Subversion

Subversion es uno de los sistemas de control de versiones más modernos y utiliza un sistema con repositorio centralizado y fue diseñado como remplazo del sistema más utilizado hasta la fecha, CVS[1]. Subversion es un proyecto de software libre maravillosamente documentado gracias al libro Version Control with Subversion (http://svnbook.red-bean.com/). Esta introducción puede considerarse un resumen de los contenidos de este libro.

El control de versiones se basa en una serie de acciones más o menos estándar de comunicación entre la copia de trabajo y el repositorio. Estas acciones son precisamente las que permite el cliente Subversion. No se entrarán en detalles de cómo administrar un repositorio y cómo se resuelven conflictos entre el repositorio y la copia de trabajo. Es una introducción a todas las prácticas beneficiosas del control de versiones; Subversion no es más que un ejemplo práctico.

2.1. Instalar Subversion

Instalar subversion es sencillo tanto en UNIX como en Windows. Es un programa común en GNU/Linux, no ocupa más de unos pocos megabytes y todas las distribuciones lo incluyen. En Debian bastará con escribir

```
$ aptitude install subversion
```

Método que se extrapola a todas las distribuciones con gestión automática de paquetes. Esta operación sirve para instalar tanto el cliente como el servidor. En Windows servidor y cliente suelen instalarse a parte. La mejor manera de instalar un servidor subversion es optar por la distribución SVN-1ClickSetup (http://svn1clicksetup.tigris.org/). El cliente más recomendable es Tortoise svn (http://tortoisesvn.tigris.org/). Este cliente convierte el mismo explorer en un cliente Subversion, es realmente rápido y cómodo de usar.

Figura 1. Explorer modificado por Tortoise SVN

2.2. Crear un repositorio subversion

Crear un repositorio en UNIX es tan sencillo como lo siguiente:

```
guillem@pc:~$ svnadmin create repositories/test
guillem@pc:~$ cd repositories
guillem@pc:~/repositories$ ls

test

guillem@pc:~/repositories$ cd test
guillem@pc:~/repositories/test$ ls

conf  dav  db  format  hooks  locks  README.txt
```

Importante: Puede surgir la pregunta de por qué subversion ha creado estos directorios en vez de los ya descritos trunk, branches y tags. Esta duda proviene de no haber entendido la diferencia entre el repositorio y la copia de trabajo. Acabamos de crear un repositorio que no contiene ningun archivo sino una base de datos de todos los cambios realizados durante el tiempo. Los desarrolladores de Subversion saben que este error es muy común, por ello han puesto en este directorio el archivo README.txt

```
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.

If the directory "db" contains a Berkeley DB environment, you
may need to tweak the values in "db/DB_CONFIG" to match the
requirements of your site.

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

Lo siguente es subir al repositorio los archivos sin versión del ordenador local. Recuérdese que aún no existe ninguna copia de trabajo. Para ello se usa el comando `import`

```
guillem@pc:~$ svn import misarchivos \
          file:///home/guillem/repositorios/test/misarchivos \
          -m "Initial import of misarchivos"
```

```
Adding         misarchivos/cversiones.xml
Adding         misarchivos/figuras
Adding         misarchivos/figuras/tortoise.png

Committed revision 1.
```

Ahora el repositorio ya contiene una cantidad mínima de archivos suficiente para crear la copia de trabajo.

> **Aviso**
>
> Es muy importante que el repositorio se encuentre en un ordenador con conexión permanente a internet. Aunque no se necesite ninguna conexión para trabajar con la copia de trabajo la mayoría de las acciones de Subversion requieren algún tipo de comunicación con el repositorio

2.3. Crear una copia de trabajo

Despues de crear el repositorio y llenarlo con documentos el siguiente paso es crear la copia de trabajo. En el control de versiónes generar una copia de trabajo a partir *de una rama o la totalidad del árbol del repositorio* recibe el nombre de *check out*.

```
guillem@pc:~$ mkdir workspace
guillem@pc:~$ cd workspace
```

Una vez creado el directorio que almacenará las copias de trabajo hay que escoger el método de comunicación con el repositorio. Los cuatro disponibles son:

`file`

Acceso al repositorio que se encuentra en el mismo ordenador. Aunque a continuación se listan los distintos métodos para acceder a un repositorio es importante que se entienda la sintaxis del comando `checkout`, para ello nada mejor que utilizar la ayuda mediante el comando `svn help`. Para consultar un comando en concreto, por ejemplo el anterior, basta con teclear `svn help checkout`

```
guillem@pc:~/workspace$ svn checkout \
file:///home/guillem/repositorios/test/misarchivos misarchivos
guillem@pc:~/workspace$ ls
```

```
misarchivos
```

`http`

Acceso mediante http. Ideal cuando el repositorio es público. Requiere que el servidor http Apache pueda servir páginas a internet y que sea capaz de comunicarse con subversion. La primera condición suele ser la más restrictiva. Muchos de los servidores subversion de proyectos de software libre utilizan este método para hacer públicos sus repositorios con permiso de lectura.

```
guillem@pc:~/workspace$ svn checkout \
http://svn.guillem.com/misarchivos misarchivos
```

`https`

Acceso mediante http con encriptación SSL

`svn`

Acceso mediante el propio servidor de archivos de Subversion llamado svnserve.

`svn+ssh`

Acceso mediante ssh. Es la mejor opción en el caso de repositorios privados de uso personal. El check out se haría con:

```
guillem@pc:~/workspace$ svn checkout \
svn+ssh:///home/guillem/repositorios/test/misarchivos misarchivos
```

2.4. Sesiones de control de versiones.

Ya se dispone de un repositorio y de una copia de trabajo. Sólo falta conocer las prácticas más comunes del control de versiones, es decir, cómo utilizar una copia de trabajo Subversion. El método será plantear sesiones ejemplo, con un principio y un final claros.

2.4.1. Una sesión simple

La sesión de control de versiones más simple posible consta de tres pasos, actualizar la copia de trabajo, modificar y crear archivos y subir los cambios al repositorio.

2.4.1.1. Actualizar la copia de trabajo

Es siempre el primer paso antes de realizar cualquier cambio a un documento. El hecho de omitir la sincronización con el repositorio genera la mayoría de los errores de funcionamiento de la copia de trabajo. Para ello sirve el comando

```
guillem@pc:~/workspace/misarchivos$ svn update
```

Esto sincroniza recursivamente todos los archivos y directorios a partir de la rama `misdocumentos`. La acción de este comando depende del directorio donde se ejecute ya que actúa sobre los subdirectorios posteriores y no sobre los previos en el árbol. Por ejemplo, si se desea sincronizar sólo el subdirectorio `figuras` y no hacer lo propio con el resto del contenido de `misdocumentos` se ejecutará dentro del directorio correspondiente:

```
guillem@pc:~/workspace/misarchivos/figuras$ svn update
```

2.4.1.2. Acciones con archivos

El trabajo normal con un ordenador se basa en modificar, crear y eliminar documentos, el hecho que estén bajo el control de versiones no modifica ni debe modificar estas acciones básicas.

`svn add`

Añade un documento *que ya se encuentra en algún directorio de trabajo* al sistema de control de versiones.

Aviso

No todos los documentos que se encuentran en los directorios de la copia de trabajo están dentro del sistema de control de versiones, deben añadirse explícitamente. Es así como debe ser para evitar que el repositorio se hinche debido a los archivos temporales.

`svn ls`

Lista los archivos en el directorio actual que están dentro del sistema de control de versiones. Es el método más común para saber qué documentos se encuentran o no en el repositorio.

`svn rm`

Borra un documento del directorio actual tanto de la copia de trabajo como del repositorio.

2.4.1.3. Subir los cambios al repositorio

Siempre que se cumple algún objetivo es muy recomendable subir los cambios al repositorio. Para ello

```
svn commit
```

Acto seguido se abrirá un editor para que, si es necesario, se anoten los comentarios correspondientes en los logs del control de versiones.

2.4.2. Otros comandos

Obviamente los comandos de trabajo con los archivos no se limitan a `add`, `ls`, y `rm`, Subversion cuenta con muchísimos mas. Para tener una idea de la potencia de subversion basta con consultar la ayuda

```
guillem@pc:~/workspace$ svn help


usage: svn "subcommand" [options] [args]
Subversion command-line client, version 1.2.0.
Type 'svn help "subcommand"' for help on a specific subcommand.

Most subcommands take file and/or directory arguments, recursing
on the directories.  If no arguments are supplied to such a
command, it recurses on the current directory (inclusive) by default.

Available subcommands:
add
blame (praise, annotate, ann)
cat
checkout (co)
cleanup
commit (ci)
copy (cp)
delete (del, remove, rm)
diff (di)
export
help (?, h)
import
info
list (ls)
lock
log
```

```
merge
mkdir
move (mv, rename, ren)
propdel (pdel, pd)
propedit (pedit, pe)
propget (pget, pg)
proplist (plist, pl)
propset (pset, ps)
resolved
revert
status (stat, st)
switch (sw)
unlock
update (up)

Subversion is a tool for version control.  For additional
information, see http://subversion.tigris.org/
```

3. Una reflexión sobre la abstracción

Todos los métodos de trabajo existentes buscan la abstracción, es decir, que el resultado no dependa de los detalles del proceso de creación así como las fórmulas matemáticas no dependen del valor de sus argumentos.

Es curioso como la abstracción conseguida por los lenguajes de programación se va al traste por culpa de las herramientas. C es en teoría un lenguaje portable, todos los sistems operativos cuentan con un compilador de C. En cambio las herramientas necesarias para programar; un editor, el entorno de desarrollo... tienen en algunos casos particularidades tan acusadas que suprimen la portabilidad del código. Es el caso de algunos IDEs que utilizan su propio sistema de configuración de proyecto.

Subversion, como otros sistemas de control de revisiones, viene a recuperar el espíritu UNIX en el que el sistema operativo y sus aplicaciones estándar son un gran entorno de desarrollo. Lo que antes se hacía con comandos en la consola como diff, cat, patch, grep... ahora se puede hacer con subversion. No debe verse Subversion como una mera herramienta de backup. Su utilidad va mucho más allá, junto con un buen editor y los compiladores o intérpretes correspondientes representa un entorno de desarrollo.

¿Cómo influye entonces Subversion en la abstracción? Un valor importante en una herramienta es que sea portable incluso entre sistemas operativos. Un editor será mejor si existen versiones para Linux, Windows, MacOS X, Solaris... Emacs y VIM son dos grandes ejemplos de portabilidad. Como subversion es completamente portable representa, junto con un editor también portable *un entorno de desarrollo integrado que permite trabajar del mismo modo independientemente del sistema operativo*.

Esto puede parecer una nimia ventaja pero no siempre uno puede llevarse su sistema operativo consigo y podrá configurarlo según sus apetencias. Gracias a Subversion *Explorer y notepad pueden utilizarse del mismo modo que bash y Emacs*. Increíble pero cierto.

Notas

1. CVS=Concurrent Versions System