Control de versiones con Subversion

Revision 4849

Ben Collins-Sussman
Brian W. Fitzpatrick
C. Michael Pilato

Capítulo 1. Introducción

El control de versiones es el arte del manejo de los cambios en la información. Ha sido durante mucho tiempo una herramienta crítica para los programadores, quienes normalmente empleaban su tiempo haciendo pequeños cambios en el software y después deshaciendo esos cambios al día siguiente. Pero la utilidad del software de control de versiones se extiende más allá de los límites del mundo del desarrollo de software. Allá donde pueda encontrarse a gente usando ordenadores para manejar información que cambia a menudo, hay un hueco para el control de versiones. Y aquí es donde entra en juego Subversion.

Este capítulo contiene una introducción general a Subversion — qué es; qué hace; cómo conseguirlo.

¿Qué es Subversion?

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

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

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

Historia de Subversion

A principios del 2000, CollabNet, Inc. (`http://www.collab.net`) comenzó a buscar desarrolladores para escribir un sustituto para CVS. CollabNet ofrece un conjunto de herramientas de software colaborativo llamado SourceCast, del cual un componente es el control de versiones. Aunque SourceCast usaba CVS como su sistema de control de versiones inicial, las limitaciones de CVS se hicieron evidentes desde el principio, y CollabNet sabía que tendría que encontrar algo mejor. Desafortunadamente, CVS se había convertido en el estándar *de facto* en el mundo del código abierto porque *no había* nada mejor, al menos no bajo una licencia libre. Así CollabNet decidió escribir un nuevo sistema de control de versiones desde cero, manteniendo las ideas básicas de CVS, pero sin sus fallos y defectos.

En febrero del 2000, contactaron con Karl Fogel, autor de *Open Source Development with CVS* (Coriolis, 1999), y le preguntaron si le gustaría trabajar en este nuevo proyecto. Casualmente, por aquel entonces Karl ya se encontraba discutiendo sobre el diseño de un nuevo sistema de control de versiones con su amigo Jim Blandy. En 1995, los dos habían fundado Cyclic Software, compañía que hacía contratos de soporte de CVS, y aunque después vendieron el negocio, seguían usando CVS todos los días en sus trabajos. La frustración de ambos con CVS había conducido a Jim a pensar cuidadosamente acerca de mejores vías para administrar datos versionados , y no sólo tenía ya el nombre de “Subversion”, sino también el diseño básico del repositorio de Subversion. Cuando CollabNet llamó, Karl aceptó inmediatamente trabajar en el proyecto, y Jim consiguió que su empresa, RedHat Software, básicamente lo donara al proyecto por un período de tiempo indefinido. Collabnet contrató a Karl y a Ben Collins-Sussman, y el trabajo detallado de diseño comenzó en mayo. Con la ayuda de algunos ajustes bien colocados de Brian Behlendorf y Jason Robbins de CollabNet, y Greg Stein (por aquel entonces un activo desarrollador independiente del proceso de especificación de WebDAV/DeltaV), Subversion atrajo rápidamente a una comunidad activa de desarrolladores. Ésto vino a demostrar que era mucha la gente que había tenido las mismas frustrantes experiencias con CVS, y que había recibido con agrado la oportunidad de hacer algo

Capítulo 2. Conceptos básicos

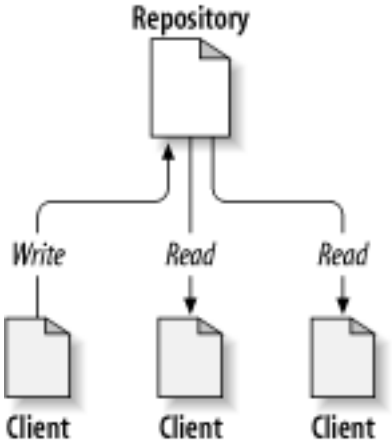
Este capítulo es una introducción breve e informal a Subversion. Si es nuevo en el tema del control de versiones, este capítulo es definitivamente para usted. Empezaremos tratando los conceptos generales en el control de versiones, seguiremos con las ideas específicas detrás de Subversion, y mostraremos algunos ejemplos simples de Subversion en acción.

Aunque los ejemplos de este capítulo muestran a gente compartiendo colecciones de archivos de código fuente, tenga en mente que Subversion puede manejar cualquier tipo de colección de archivos—no está limitado a asistir a programadores de ordenadores.

El repositorio

Subversion es un sistema centralizado para compartir información. La parte principal de Subversion es el repositorio, el cual es un almacén central de datos. El repositorio guarda información en forma de *árbol de archivos*—una típica jerarquía de archivos y directorios. Cualquier número de *clientes* puede conectarse al repositorio y luego leer o escribir en esos archivos. Al escribir datos, un cliente pone a disposición de otros la información; al leer datos, el cliente recibe información de otros. La figura Figura 2.1, “Un sistema cliente/servidor típico” ilustra ésto.

Figura 2.1. Un sistema cliente/servidor típico



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

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

Modelos de versionado

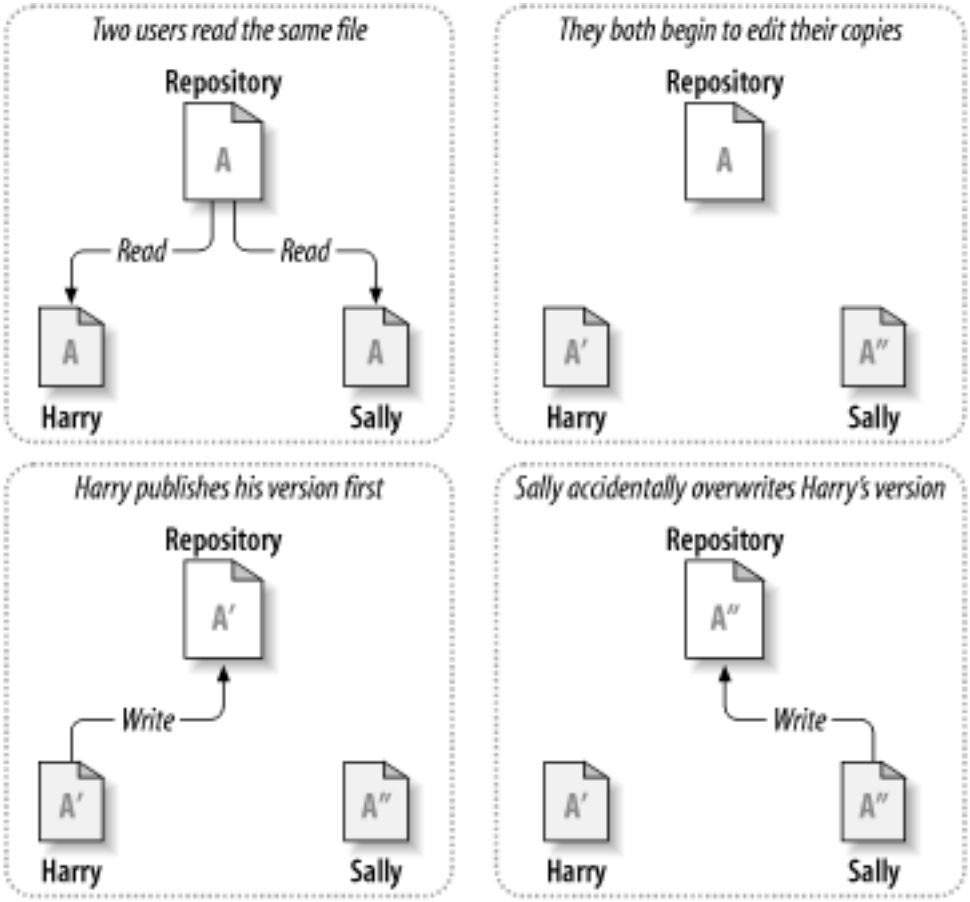
La misión principal de un sistema de control de versiones es permitir la edición colaborativa y la compartición de los datos. Sin embargo, existen diferentes sistemas que utilizan diferentes estrategias para alcanzar este objetivo.

El problema de compartir archivos

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

Considere el escenario mostrado en Figura 2.2, “El problema a evitar”. Suponga que tenemos dos colaboradores, Juan y Carmen. Cada uno de ellos decide editar el mismo archivo del repositorio al mismo tiempo. Si Juan guarda sus cambios en el repositorio en primer lugar, es posible que (unos momentos más tarde) Carmen los sobreescriba accidentalmente con su propia versión del archivo. Si bien es cierto que la versión de Juan no se ha perdido para siempre (porque el sistema recuerda cada cambio), cualquier cambio que Juan haya hecho *no* estará presente en la versión más reciente de Carmen porque, para empezar, ella nunca vio los cambios de Juan. El trabajo de Juan sigue efectivamente perdido—o al menos ausente en la última versión del archivo—y probablemente por accidente. ¡Esta es definitivamente una situación que queremos evitar!

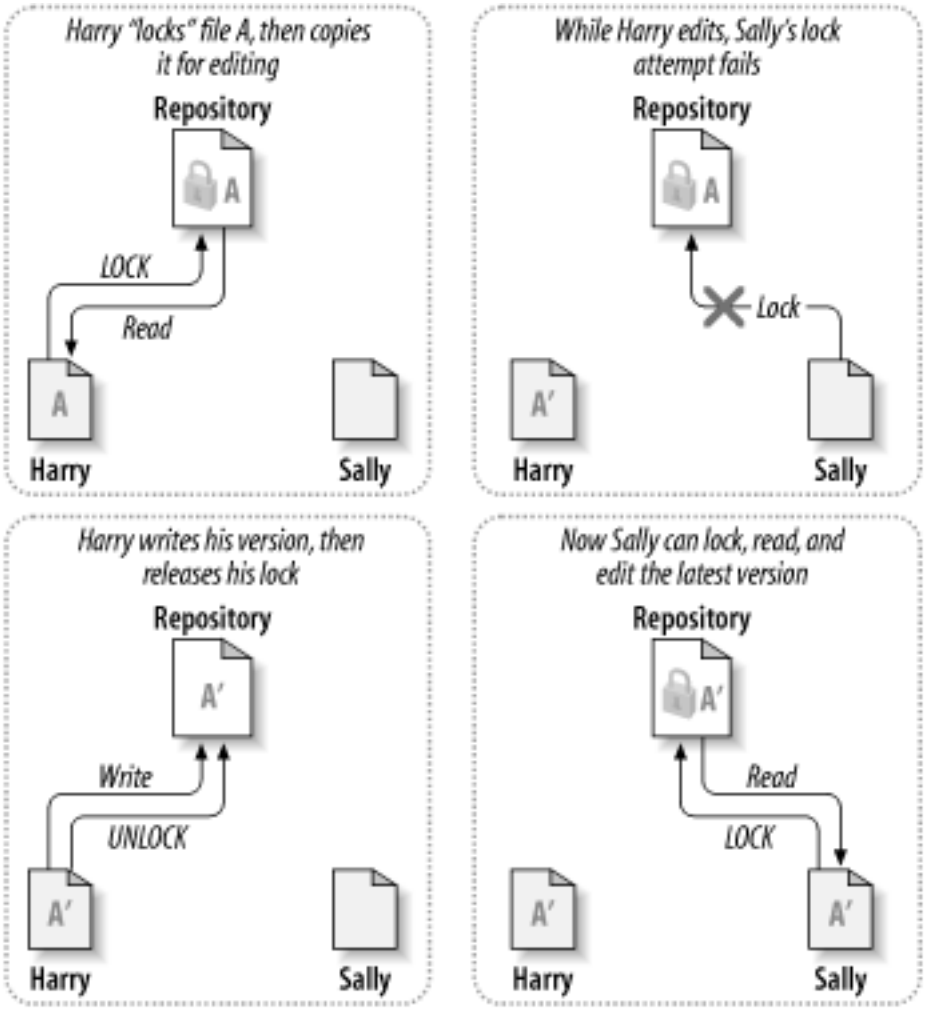
Figura 2.2. El problema a evitar



La solución bloqueo-modificación-desbloqueo

Muchos sistemas de control de versiones utilizan un modelo de *bloqueo-modificación-desbloqueo* para atacar este problema. En un sistema como éste, el repositorio sólo permite a una persona modificar un archivo al mismo tiempo. Juan debe “bloquear” primero el archivo para luego empezar a hacerle cambios. Bloquear un archivo se parece mucho a pedir prestado un libro de la biblioteca; si Juan ha bloqueado el archivo, entonces Carmen no puede hacerle cambios. Por consiguiente, si ella intenta bloquear el archivo, el repositorio rechazará la petición. Todo lo que puede hacer es leer el archivo y esperar a que Juan termine sus cambios y deshaga el bloqueo. Tras desbloquear Juan el archivo, Carmen puede aprovechar su turno bloqueando y editando el archivo. La figura Figura 2.3, “La solución bloqueo-modificación-desbloqueo” demuestra esta sencilla solución.

Figura 2.3. La solución bloqueo-modificación-desbloqueo



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

- *Bloquear puede causar problemas administrativos.* En ocasiones Juan bloqueará un archivo y se olvidará de él. Mientras tanto, como Carmen está aún esperando para editar el archivo, sus manos están atadas. Y luego Juan se va de vacaciones. Ahora Carmen debe conseguir que un administrador deshaga el bloqueo de Juan. La situación termina causando muchas demoras innecesarias y pérdida de tiempo.

- *Bloquear puede causar una serialización innecesaria.* ¿Qué sucede si Juan está editando el inicio de un archivo de texto y Carmen simplemente quiere editar el final del mismo archivo? Estos cambios no se solapan en absoluto. Ambos podrían editar el archivo simultáneamente sin grandes perjuicios, suponiendo que los cambios se combinaran correctamente. No hay necesidad de turnarse en esta situación.

- *Bloquear puede causar una falsa sensación de seguridad.* Imaginemos que Juan bloquea y edita el archivo A, mientras que Carmen bloquea y edita el archivo B al mismo tiempo. Pero suponga que A y B dependen uno del otro y que los cambios hechos a cada uno de ellos son semánticamente incompatibles. Súbitamente A y B ya no funcionan juntos. El sistema de bloqueo se mostró ineficaz a la hora de evitar el problema—sin embargo, y de algún modo, ofreció una falsa sensación de seguridad. Es fácil para Juan y Carmen imaginar que al bloquear archivos, cada uno está empezando una tarea segura y aislada, lo cual les inhibe de

discutir sus cambios incompatibles desde un principio.

La solución copiar-modificar-mezclar

Subversion, CVS y otros sistemas de control de versiones utilizan un modelo del tipo *copiar-modificar-mezclar* como alternativa al bloqueo. En este modelo, el cliente de cada usuario se conecta al repositorio del proyecto y crea una *copia de trabajo* personal—una réplica local de los archivos y directorios del repositorio. Los usuarios pueden entonces trabajar en paralelo, modificando sus copias privadas. Finalmente, todas las copias privadas se combinan (o mezclan) en una nueva versión final. El sistema de control de versiones a menudo ayuda con la mezcla, pero en última instancia es un ser humano el responsable de hacer que ésto suceda correctamente.

He aquí un ejemplo. Digamos que Juan y Carmen crean sendas copias de trabajo del mismo proyecto, extraídas del repositorio. Ambos trabajan concurrentemente y hacen cambios a un mismo archivo A dentro de sus copias. Carmen guarda sus cambios en el repositorio primero. Cuando Juan intenta guardar sus cambios más tarde, el repositorio le informa de que su archivo A está *desactualizado*. En otras palabras, que el archivo A en el repositorio ha sufrido algún cambio desde que lo copió por última vez. Por tanto, Juan le pide a su cliente que *mezcle* cualquier cambio nuevo del repositorio con su copia de trabajo del archivo A. Es probable que los cambios de Carmen no se solapen con los suyos; así que una vez que tiene ambos juegos de cambios integrados, Juan guarda su copia de trabajo de nuevo en el repositorio. Las figuras Figura 2.4, “La solución copiar-modificar-mezclar” y Figura 2.5, “La solución copiar-modificar-mezclar (continuación)” muestran este proceso.

Figura 2.4. La solución copiar-modificar-mezclar

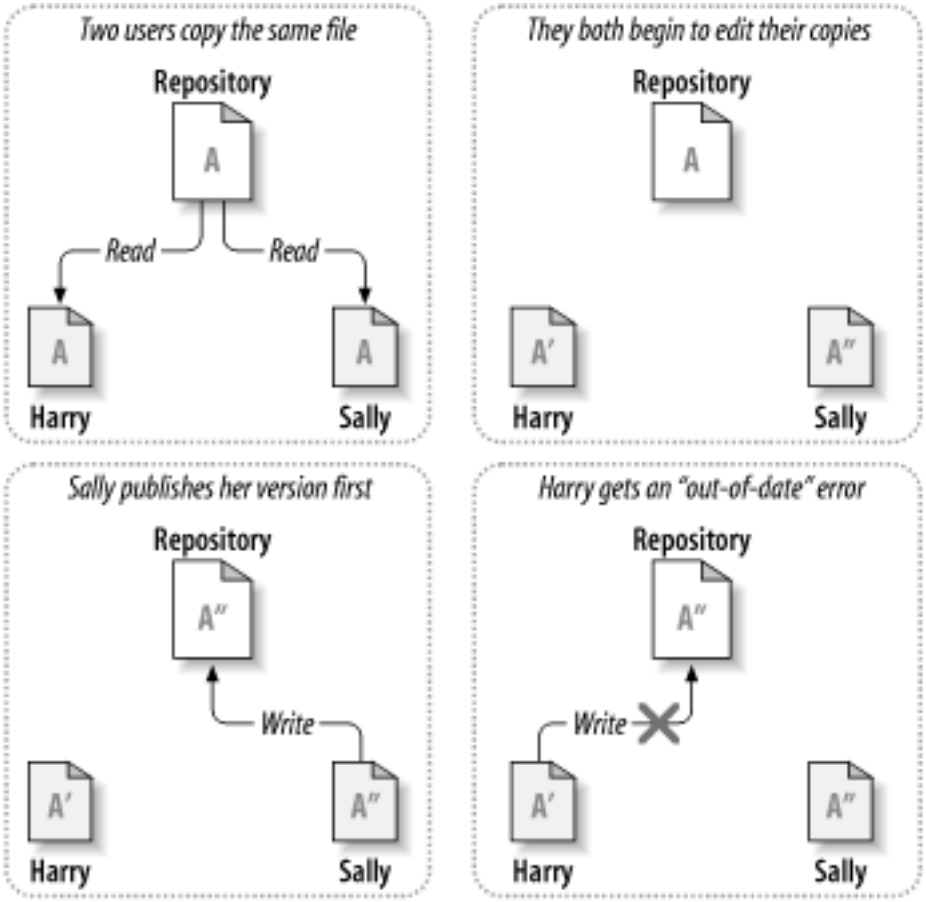
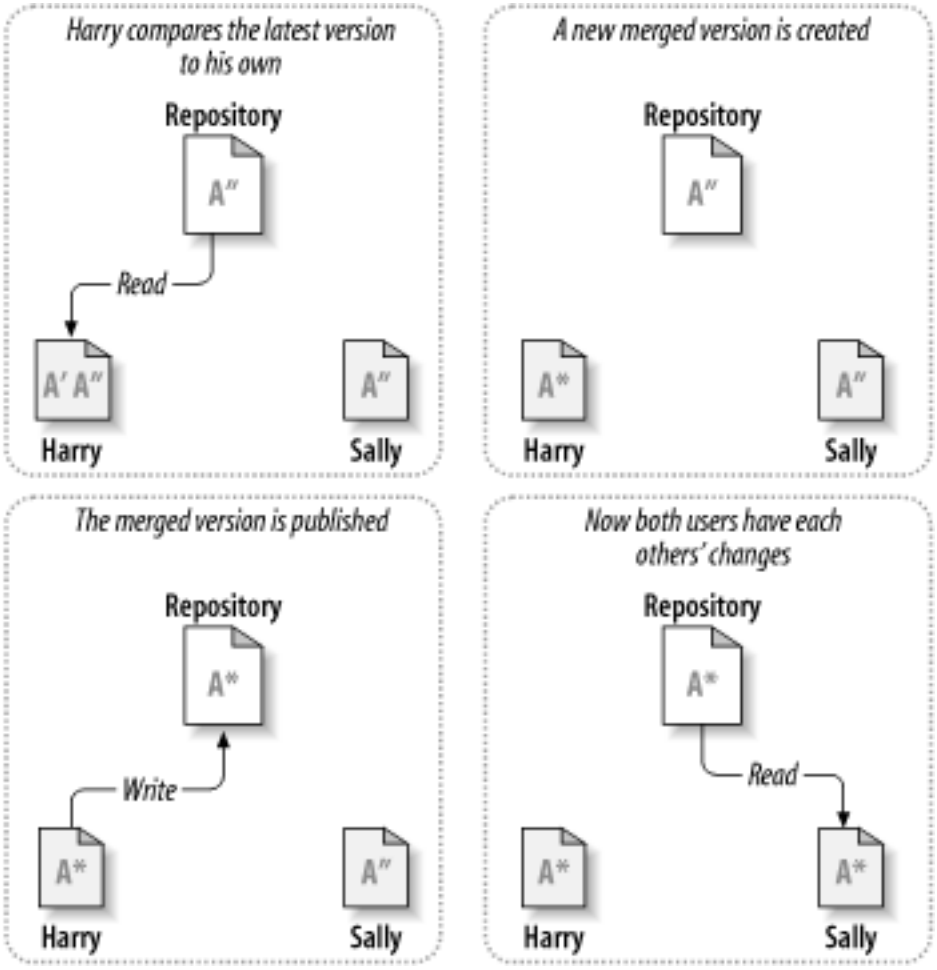


Figura 2.5. La solución copiar-modificar-mezclar (continuación)



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

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

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

Subversion en acción

Es hora de movernos de lo abstracto a lo concreto. En esta sección mostraremos ejemplos reales de Subversion en la práctica.

Copias de trabajo

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

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

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

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

Un repositorio típico de Subversion contiene a menudo los archivos (o el código fuente) de varios proyectos; normalmente, cada proyecto es un subdirectorio en el árbol del sistema de archivos del repositorio. En esta disposición, la copia de trabajo de un usuario se corresponde habitualmente con un subárbol particular del repositorio.

Por ejemplo, suponga que usted tiene un repositorio que contiene dos proyectos de software, `paint` y `calc`. Cada proyecto reside en su propio subdirectorio dentro del directorio raíz, tal como se muestra en Figura 2.6, "El sistema de archivos del repositorio".

Figura 2.6. El sistema de archivos del repositorio

Ahora sus cambios a `button.c` han sido consignados al repositorio; si otro usuario obtiene una copia de trabajo de `/calc`, podrá ver sus cambios en la última versión del archivo.

Suponga que tiene un colaborador, Carmen, quien obtuvo una copia de trabajo de `/calc` al mismo tiempo que usted. Cuando usted envía sus cambios sobre `button.c`, la copia de trabajo de Carmen se deja sin cambios; Subversion solo modifica las copias de trabajo a petición del usuario.

Para tener su proyecto actualizado, Carmen puede pedir a Subversion que proceda a *actualizar* su copia de trabajo, usando para ello el comando **update** de Subversion. Ésto incorporará los cambios hechos por usted en la copia de trabajo de Carmen, así como otros cambios consignados desde que ella hizo el check out.

```
$ pwd
/home/sally/calc

$ ls -A
.svn/ Makefile integer.c button.c

$ svn update
U button.c
```

La salida del comando **svn update** indica que Subversion actualizó el contenido de `button.c`. Observe que Carmen no necesitó especificar qué archivos actualizar; Subversion usa la información del directorio `.svn`, junto con información adicional del repositorio, para decidir qué archivos necesitan una actualización.

Revisiones

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

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

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

La Figura 2.7, “El repositorio” ilustra una manera interesante de ver el repositorio. Imagine un array de números de revisión, comenzando por el 0, que se extiende de izquierda a derecha. Cada número de revisión tiene un árbol de ficheros colgando debajo de él, y cada árbol es una “instantánea” del aspecto del repositorio tras cada envío.

Figura 2.7. El repositorio

Descarga inicial

La mayor parte del tiempo, usted empezará a usar un repositorio de Subversion haciendo un *checkout* de su proyecto . Descargar un repositorio crea una copia de éste en en su máquina local. Esta copia contiene el HEAD (última revisión) del repositorio de Subversion que usted especifica en la línea de comandos:

```
$ svn checkout http://svn.collab.net/repos/svn/trunk
A  trunk/subversion.dsw
A  trunk/svn_check.dsp
A  trunk/COMMITTERS
A  trunk/configure.in
A  trunk/IDEAS
…
Checked out revision 2499.
```

Esquema del repositorio

Si se está preguntando qué es `trunk` en la URL anterior, es parte de la manera que le recomendamos estructurar su repositorio Subversion, acerca del cual hablaremos mucho más en Capítulo 4, *Crear ramas y fusionarlas*.

Aunque el ejemplo de arriba descarga el directorio trunk, usted puede descargar fácilmente cualquier subdirectorio más profundo de un repositorio especificando el subdirectorio en la URL de descarga:

```
$ svn checkout http://svn.collab.net/repos/svn/trunk/doc/book/tools
A  tools/readme-dblite.html
A  tools/fo-stylesheet.xsl
A  tools/svnbook.el
A  tools/dtd
A  tools/dtd/dblite.dtd
…
Checked out revision 2499.
```

Como Subversion usa un modelo “copie-modifique-fusione” en vez de “bloquear-modificar-desbloquear” (vea Capítulo 2, *Conceptos básicos*), usted ya puede empezar a realizar cambios a los ficheros y directorios en su copia de trabajo local. Su copia local es justo como cualquier otra colección de ficheros y directorios en su sistema. Usted puede editarlos y cambiarlos, moverlos, usted puede incluso borrar la copia local entera y olvidarse de ella.

Mientras que su copia de trabajo local es “justo como cualquier otra colección de ficheros y directorios en su sistema”, usted necesita hacer saber a Subversion si va a reacomodar cualquier cosa dentro de su copia local. Si desea copiar o mover un elemento en una copia local, debe usar **svn copy** o **svn move** en vez de los comandos para copiar y mover proporcionados por su sistema operativo. Hablaremos más acerca de ellos al avanzar en este capítulo.

A menos que esté listo para enviar al repositorio un fichero o directorio nuevo, o cambios a unos existentes, no hay necesidad de indicarle al servidor de Subversion que usted haya hecho algo más.

¿Qué pasa con el directorio `.svn`?

Cada directorio en una copia de trabajo local contiene un área administrativa, un subdirectorio llamado `.svn`. Generalmente, los comandos de listado de directorios no muestran este subdirectorio, pero este es sin embargo un directorio importante. Sea lo que fuere lo que haga ¡no borre o cambie nada en el área administrativa! Subversion depende de ella para administrar su copia de trabajo local.

Mientras que usted puede descargar una copia de trabajo local con la URL del repositorio como único argumento, también puede especificar un directorio después de su URL del repositorio. Esto pone su copia de trabajo local dentro del nuevo directorio que usted nombra. Por ejemplo:

```
$ svn checkout http://svn.collab.net/repos/svn/trunk subv
A  subv/subversion.dsw
A  subv/svn_check.dsp
A  subv/COMMITTERS
A  subv/configure.in
A  subv/IDEAS
…
Checked out revision 2499.
```

Esto pondrá su copia de trabajo local en un directorio llamado `subv` en vez de en un directorio llamado `trunk` como hicimos previamente.

Ciclo básico de trabajo

Subversion tiene numerosas características, opciones, campanas y silbidos, pero bajo uso cotidiano lo más probable es que use solamente algunas de ellas. En esta sección veremos las cosas más comunes que usted puede encontrarse haciendo con Subversion en el transcurso de un día de trabajo.

El ciclo de trabajo típico se parece a esto:

- Actualizar su copia de trabajo local
  - **svn update**
- Hacer cambios
  - **svn add**
  - **svn delete**
  - **svn copy**
  - **svn move**
- Examinar sus cambios
  - **svn status**
  - **svn diff**
  - **svn revert**
- Fusionar los cambios de otros en su copia de trabajo
  - **svn merge**
  - **svn resolved**
- Enviar sus cambios
  - **svn commit**

Actualizar su copia de trabajo local

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

```
$ svn update
U  foo.c
U  bar.c
Updated to revision 2.
```

En este caso, alguien envió modificaciones a `foo.c` y `bar.c` desde la última vez que usted actualizó, y Subversion ha actualizado su copia de trabajo local para incluir estos cambios.

Vamos a examinar la salida de **svn update** un poco más. Cuando el servidor envía cambios a su copia de trabajo local, un código de letras es mostrado al lado de cada elemento para hacerle saber qué acciones realizó Subversion para actualizar su copia de trabajo local:

`U foo`
: El fichero `foo` fue actualizado[1] (recibidos cambios del servidor).

`A foo`
: El fichero o directorio `foo` fue Añadido a su copia de trabajo local.

`D foo`
: El fichero o directorio `foo` fue borrado [2] de su copia de trabajo local.

`R foo`
: El fichero o directorio `foo` fue Reemplazado en su copia de trabajo local; esto es, `foo` fue borrado, y un nuevo objeto con el mismo nombre fue añadido. Aunque pueden tener el mismo nombre, el repositorio los considera objetos distintos con historiales distintos.

`G foo`
: El fichero `foo` recibió nuevos cambios del repositorio, pero su copia local del fichero tenía las modificaciones que ud. ya había hecho. O los cambios no se intersectaron , o los cambios eran exactamente iguales que sus modificaciones locales, así que Subversion ha fusionado [3] satisfactoriamente los cambios del repositorio en el fichero sin ningún problema.

`C foo`
: El fichero `foo` recibió cambios Conflicting del servidor. Los cambios del servidor directamente se superpusieron sobre sus propios cambios en el fichero. Aunque no hay necesidad de aterrarse. Esta superposición necesita ser resuelta por un humano (usted); tratamos esta situación más tarde en este capítulo.

Hacer cambios en su copia de trabajo local

Ahora puede conseguir trabajar y hacer cambios en su copia de trabajo local. Generalmente es más conveniente decidir un cambio particular (o un conjunto de cambios) a hacer, por ejemplo escribir una nueva característica, corregir un fallo, etc. Los comandos de Subversion que puede usar aquí son **svn add**, **svn delete**, **svn copy**, y **svn move**. Sin embargo, si usted está simplemente corrigiendo los archivos que están ya en Subversion, puede no necesitar utilizar ninguno de estos comandos hasta que envíe los cambios. Cambios que puede hacer a su copia de trabajo local:

[1] N. del T.: La inicial usada representa la palabra Updated (actualizado) en el idioma inglés
[2] N. del T.: La inicial usada representa la palabra Deleted (borrado) en el idioma inglés
[3] N. del T.: merGed, fusionado, en inglés

Cambios en los ficheros
: Esta es la clase más simple de cambio. No necesita decirle a Subversion que se propone a cambiar un fichero; simplemente haga los cambios. Subversion será capaz de detectar automáticamente qué archivos han sido cambiados.

Cambios en el árbol
: Puede preguntar a Subversion que “marque” ficheros y directorios para el borrado planificado, la adición, la copia, o moverlos. Mientras estos cambios pueden ocurrir inmediatamente en su copia de trabajo local, ninguna adición o borrado sucederá en el repositorio hasta que envíe los cambios.

Para hacer cambios en los ficheros, use su editor de textos, procesador de textos, programa de gráficos , o cualquier herramienta que usaría normalmente. Subversion maneja ficheros binarios tan fácilmente como maneja archivos de texto—y tan eficientemente también.

Aquí hay una descripción de los cuatro subcomandos de Subversion que usted usará más a menudo para hacer cambios del árbol (cubriremos **svn import** y **svn mkdir** después.

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

svn delete foo
: Programa borrar `foo` del repositorio. Si `foo` es un fichero, se borrará inmediatamente de su copia de trabajo local. Si `foo` es un directorio, este no es borrado, pero Subversion lo programa para borrarlo. Cuando envíe sus cambios, `foo` será borrado de su copia de trabajo y del repositorio.[4]

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

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

> **Cambiando el repositorio sin una copia de trabajo**
>
> Anteriormente en este capítulo, dijimos que tiene que enviar cualquier cambio que usted haga en orden para que el repositorio refleje estos cambios. Esto no es enteramente cierto —*hay* algunos usos-casos que inmediatamente envían los cambios del árbol al repositorio. Esto sucede solamente cuando un subcomando está operando directamente sobre una URL, al contrario que sobre una ruta de la copia-local . En particular, usos específicos de **svn mkdir**, **svn copy**, **svn move**, y **svn delete** pueden trabajar con URLs.
>
> Las operaciones de URL se comportan de esta manera porque los comandos que manipulan en una copia de trabajo pueden usar la copia de trabajo como una clase de “área de estancamiento”para preparar sus cambios antes de enviarlos al repositorio. Los comandos que operan sobre URLs no tienen este lujo, así cuando usted opera directamente sobre una URL, cualquiera de las acciones anteriores representa un envío inmediato.

Examine sus cambios

[4] Por supuesto, nada es borrado totalmente del repositorio—solo del `HEAD` del repositorio. Puede conseguir cualquier cosa que borró descargando (o actualizando su copia de trabajo) a una revisión anterior a la que lo borró.

Una vez que haya terminado de hacer cambios, necesita enviarlos al repositorio, pero antes de hacerlo, generalmente es buena idea echar un vistazo a lo que ha cambiado exactamente. Examinando sus cambios antes de que los envíe, puede hacer un mensaje de informe de cambios más exacto. También puede descubrir que ha cambiado inadvertidamente un fichero, y esto le da la posibilidad de invertir esos cambios antes de enviarlos. Además, esta es una buena oportunidad para revisar y escudriñar cambios antes de publicarlos. Usted puede ver exactamente qué cambios ha hecho usando **svn status**, **svn diff**, y **svn revert**. Generalmente usará los primeros dos comandos para descubrir qué ficheros han cambiado en su copia de trabajo local, y después quizá el tercero para invertir algunos (o todos) de esos cambios.

Subversion ha sido optimizado para ayudarle con esta tarea, y es capaz de hacer muchas cosas sin comunicarse con el repositorio. En particular, su copia de trabajo local contiene una copia "prístina" secreta almacenada de cada fichero de versión controlado dentro del área `.svn`. Debido a esto, Subversion puede rápidamente mostrarle cómo sus ficheros de trabajo han cambiado, o incluso permitirle deshacer sus cambios sin contactar con el repositorio.

svn status

Probablemente usará el comando **svn status** más que cualquier otro comando de Subversion.

> **Usuarios de CVS: ¡Lleve a cabo esa actualización!**
>
> Probablemente esté acostumbrado a usar **cvs update** para ver qué cambios le ha hecho a su copia de trabajo local. **svn status** le dará toda la información que necesita sobre qué ha cambiado en su copia de trabajo local—sin acceder al repositorio o incorporando nuevos cambios potenciales publicados por otros usuarios.
>
> En Subversion, **update** hace justo eso—actualiza su copia de trabajo local con cualquier cambio enviado al repositorio desde la última vez que usted ha actualizado su copia de trabajo. Tendrá que romper el hábito de usar el comando **update** para ver qué modificaciones locales ha hecho.

Si ejecuta **svn status** en lo alto de su copia de trabajo sin argumentos, detectará todos los cambios de fichero y árbol que usted ha hecho. Este ejemplo está diseñado para mostrar todos los códigos de estado diferentes que **svn status** puede devolver. (Observe que el texto que sigue a # en el siguiente ejemplo no es impreso realmente por **svn status**.)

```
$ svn status
  L    abc.c                # svn has a lock in its .svn directory for abc.c
M      bar.c                # the content in bar.c has local modifications
 M     baz.c                # baz.c has property but no content modifications
X      3rd_party            # this dir is part of an externals definition
?      foo.o                # svn doesn't manage foo.o
!      some_dir             # svn manages this, but it's either missing or incomplete
~      qux                  # versioned as dir, but is file, or vice versa
I      .screenrc            # this file is ignored
A  +   moved_dir            # added with history of where it came from
M  +   moved_dir/README     # added with history and has local modifications
D      stuff/fish.c         # this file is scheduled for deletion
A      stuff/loot/bloo.h    # this file is scheduled for addition
C      stuff/loot/lump.c    # this file has conflicts from an update
    S  stuff/squawk         # this file or dir has been switched to a branch
…
```

En este formato de salida **svn status** impresa cinco columnas de caracteres, seguidos por varios caracteres de espacio en blanco, seguido por un nombre de fichero o directorio. La primera columna dice el estado de un fichero o directorio y/o su contenido. Los códigos impresos aquí son:

`A file_or_dir`
: El fichero o directorio `file_or_dir` ha sido programado para la adición en el repositorio.

`C file`
: El fichero `file` está en un estado de conflicto. Esto es, los cambios recibidos del servidor durante una actualización se solapan con cambios locales que usted tiene en su copia de trabajo. Debe resolver este conflicto antes de enviar sus cambios al repositorio.

`D file_or_dir`
: El fichero o directorio `file_or_dir` ha sido programado para la supresión del repositorio.

`M file`
: El contenido del fichero `file` ha sido modificado.

`X dir`
: El directorio `dir` está sin versionar, pero está relacionado con una definición externa de Subversion. Para descubrir más acerca de definiciones externas, vea “Repositorios externos”.

`? file_or_dir`
: El fichero o directorio `file_or_dir` no está bajo control de versiones. Puede silenciar la marca de pregunta pasando la opción `--quiet` (`-q`) a **svn status**, o o poniendo la característica `svn:ignore` en el directorio padre. Para más información sobre ficheros ignorados, vea “`svn:ignore`”.

`! file_or_dir`
: El fichero o directorio `file_or_dir` está bajo el control de versiones pero falta o está de alguna manera incompleto. El objeto puede faltar si se ha borrado usando un comando ajeno a Subversion. En el caso de un directorio, puede estar incompleto si ha interrumpido una descarga o una actualización. Un rápido **svn update** repondrá el fichero o el directorio desde el repositorio, o **svn revert file** restaurará un archivo que falta.

`~ file_or_dir`
: El fichero o directorio `file_or_dir` está en el repositorio como un tipo de objeto, pero actualmente está en su copia de trabajo como otro tipo. Por ejemplo, Subversion pudo tener un fichero en el repositorio, pero usted borró el fichero y creó un directorio en su lugar, sin usar los comandos **svn delete** o **svn add**.

`I file_or_dir`
: Subversion está “ignorando” el fichero o directorio `file_or_dir`, probablemente porque usted se lo dijo. Para más información sobre ficheros ignorados, vea “`svn:ignore`”. Observe que este símbolo solo aparece si le pasa la opción `--no-ignore` a **svn status**.

La segunda columna dice el estado de las propiedades de un fichero o un directorio (vea “Propiedades” para más información sobre propiedades). Si aparece una `M` en la segunda columna, entonces las propiedades han sido modificadas, si no un espacio en blanco será impreso.

La tercera columna solo mostrará un espacio en blanco o una `L` la cual significa que Subversion ha bloqueado el objeto en el área de trabajo `.svn`. Usted verá una `L` si ejecuta **svn status** en un directorio donde un **svn commit** esté en progreso—quizás cuando esté editando el informe de cambios. Si Subversion no se está ejecutando, entonces probablemente Subversion fue interrumpido y el bloqueo necesita ser eliminado ejecutando **svn cleanup** (más sobre eso más adelante en este capítulo).

La cuarta columna solo mostrará un espacio blanco o un `+` el cual significa que el fichero o directorio está programado para ser añadido o modificado con historial adicional adjunto. Esto ocurre típicamente cuando usted **svn move** o **svn copy** un fichero o directorio. Si usted ve `A  +`, esto significa que el objeto está programado para la adición-con-historial. Este puede ser un fichero, o la raíz de un directorio copiado. `+` significa que el objeto es parte de un subárbol programado para la adición-con-historial, p.e.algún padre fue copiado,and it's just coming along for the ride. `M  +` significa que el objeto es parte de un subárbol programado para la adición-con-historial, *y* este tiene modificaciones locales. Cuando envíe los cambios, primero el padre será añadido-con-historial (copiado), lo que significa que este fichero existirá automáticamente en la copia. Entonces las modificaciones locales serán enviadas al repositorio.

La quinta columna solo mostrará un espacio en blanco o una `S`. Esto significa que el fichero o directorio ha sido movido de la ruta del resto de la copia de trabajo (usando **svn switch**) a una rama.

Si usted pasa una ruta específica a **svn status**, este le dará información acerca de ese objeto solamente:

```
$ svn status stuff/fish.c
D      stuff/fish.c
```

svn status también tiene una opción `--verbose` (`-v`), el cuál le mostrará el estado de *todos* los objetos en su copia de trabajo, incluso si este no ha sido cambiado:

```
$ svn status --verbose
M               44        23    sally     README
                44        30    sally     INSTALL
M               44        20    harry     bar.c
                44        18    ira       stuff
                44        35    harry     stuff/trout.c
D               44        19    ira       stuff/fish.c
                44        21    sally     stuff/things
A                0         ?     ?        stuff/things/bloo.h
                44        36    harry     stuff/things/gloo.c
```

Esta es la “forma larga” de salida de **svn status**. La primera columna permanece igual, pero la segunda columna muestra la revisión de trabajo del fichero. La tercera y cuarta columna muestra la revisión en la cuál el objeto cambió por última vez, y quién lo cambió.

Ninguna de las invocaciones anteriores a **svn status** contactaban con el repositorio, trabajan solo localmente comparando los metadatos en el directorio `.svn` con la copia de trabajo local. Finalmente está la opción `--show-updates` (`-u`), la cual contacta con el repositorio y añade información acerca de las cosas que están fuera-de-fecha:

```
$ svn status --show-updates --verbose
M      *        44        23    sally     README
M               44        20    harry     bar.c
       *        44        35    harry     stuff/trout.c
D               44        19    ira       stuff/fish.c
A                0         ?     ?        stuff/things/bloo.h
Status against revision:   46
```

Observe los dos asteriscos: si usted ejecutara **svn update** en este punto, usted podría recibir cambios para `README` y `trout.c`. Esto le dice cierta información muy útilmdash;necesitará actualizar y coger los cambios del servidor para `README` antes de enviar sus cambios, o el repositorio rechazará su envío por estar fuera-de-fecha. (Más de este tema más adelante.)

svn diff

Otra manera de examinar sus cambios es con el comando **svn diff**. Puede descubrir *exactamente* cómo ha modificado cosas ejecutando **svn diff** sin argumentos, el cual imprime los cambios de los ficheros en formato unificado del diff:[5]

```
$ svn diff
Index: bar.c
===================================================================
--- bar.c (revision 3)
+++ bar.c (working copy)
@@ -1,7 +1,12 @@
+#include <sys/types.h>
+#include <sys/stat.h>
```

[5]Subversion usa su motor interno de diff, el cual produce un formato unificado del diff, por defecto. Si quiere la salida del diff en un formato diferente, especifique un programa diff externo usando `--diff-cmd` y pasando cualquier parámetro que quiera usando la opción `--extensions`. Por ejemplo, para ver diferencias locales en el fichero `foo.c` en contexto con el formato de salida mientras ignora cambios en espacios en blanco puede ejecutar **svn diff --diff-cmd /usr/bin/diff --extensions '-bc' foo.c**.

```
+#include <unistd.h>
+
+#include <stdio.h>

 int main(void) {
-  printf("Sixty-four slices of American Cheese...\n");
+  printf("Sixty-five slices of American Cheese...\n");
 return 0;
 }

Index: README
===================================================================
--- README (revision 3)
+++ README (working copy)
@@ -193,3 +193,4 @@
+Note to self:  pick up laundry.

Index: stuff/fish.c
===================================================================
--- stuff/fish.c (revision 1)
+++ stuff/fish.c (working copy)
-Welcome to the file known as 'fish'.
-Information on fish will be here soon.

Index: stuff/things/bloo.h
===================================================================
--- stuff/things/bloo.h (revision 8)
+++ stuff/things/bloo.h (working copy)
+Here is a new file to describe
+things about bloo.
```

El comando **svn diff** produce esta salida comparando sus ficheros de copia de trabajo contra la copia “prístina” almacenada en el área `.svn`. Los ficheros programados para la adición se visualizan como texto-añadido, y los ficheros programados para la eliminación son visualizados como texto eliminado.

La salida es visualizada en *formato unificado del diff*. Esto es, las lineas quitadas son empezadas con un `-` y las lineas añadidas son empezadas con un `+`. **svn diff** también imprime el nombre del fichero e información útil para el programa **patch**, así que puede generar “parches” redireccionando la salida del diff a un fichero:

```
$ svn diff > patchfile
```

Usted puede, por ejemplo, mandar por email el fichero de parche a otro desarrollador para la revisión o testeo antes de enviarlo.

svn revert

Ahora suponga que usted ve la salida del diff anterior, y se da cuenta que sus cambios a `README` son un error; quizás accidentalmente tecleó ese texto en el fichero equivocado en su editor

Esta es una oportunidad perfecta para usar **svn revert**.

```
$ svn revert README
Reverted 'README'
```

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

```
$ svn status foo
?      foo

$ svn add foo
A         foo

$ svn revert foo
Reverted 'foo'

$ svn status foo
?      foo
```

svn revert *ITEM* tiene exactamente el mismo efecto que suprimiendo *ITEM* de su copia de trabajo local y después ejecutando **svn update -r BASE** *ITEM*. Sin embargo, si está revirtiendo un fichero, **svn revert** tiene una diferencia muy considerable—no tiene que comunicarse con el repositorio para reponer su fichero.

O quizás usted quito equivocadamente un fichero del control de versión:

```
$ svn status README
       README

$ svn delete README
D          README

$ svn revert README
Reverted 'README'

$ svn status README
       README
```

¡Mira mamá! ¡Sin red!

Estos tres comandos (**svn status**, **svn diff**, y **svn revert**) pueden ser usados sin ningún acceso de red. Esto lo hace sencillo para administrar sus cambios-en-progreso cuando usted está en alguna parte sin una conexión de red, tal como viajando en un avión, montando en un tren o hackeando en la playa.

Subversion hace esto manteniendo almacenes privados de versiones prístinas de cada fichero versionado dentro del área administrativa `.svn`. Esto permite a Subversion reportar—y revertir—modificaciones locales a esos ficheros *sin acceso de red*. Este almacén (llamado el "texto-base") también permite a Subversion mandar las modificaciones locales del usuario durante un envío al servidor como un *delta* comprimido (o "diferencial") contra la versión prístina. Tener este almacén es un beneficio tremendo—incluso si usted tiene una conexión de red rápida, es mucho más rápido mandar solamente los cambios del fichero al servidor antes que el fichero entero. A primera vista, esto puede no parecer tan importante, pero imagine la repercusión si usted intenta enviar un cambio de una línea a un fichero de 400MB ¡ y tiene que enviar el fichero entero al servidor!

Resolver conflictos (fusionando los cambios de otros)

Ya hemos visto cómo **svn status -u** puede predecir conflictos. Suponga que ejecuta **svn update** y ocurren algunas cosas interesantes

```
$ svn update
```

```
$ ls sandwich.*
sandwich.txt
```

Observe que cuando usted invierte un fichero conflictivo, no tiene que ejecutar **svn resolved**.

Ahora usted está listo para enviar sus cambios. Observe que **svn resolved**, al contrario de la mayoría de los otros comandos que nos hemos ocupado en este capitulo, requiere un argumento. En cualquier caso, debe tener cuidado y solo ejecutar **svn resolved** cuando esté seguro que ha arreglado el conflicto en su fichero—una vez los ficheros temporales son borrados, Subversion le dejará enviar el fichero incluso si todavía tiene marcas de conflicto.

Enviar sus cambios

¡Finalmente! Su edición está terminada, ha fusionado todos los cambios del servidor, y está listo para enviar sus cambios al repositorio.

El comando **svn commit** envía todos sus cambios al repositorio. Cuando usted envía un cambio, necesita proveer un *mensaje de registro*, describiendo su cambio. Su mensaje de registro será adjuntado a la nueva revisión que ha creado. Si su mensaje de registro es breve, puede querer proveerlo en la línea de comando usando la opción `--message` (o `-m`):

```
$ svn commit --message "Corrected number of cheese slices."
Sending        sandwich.txt
Transmitting file data .
Committed revision 3.
```

Sin embargo, si ha estado componiendo su mensaje de registro mientras trabaja, puede querer decirle a Subversion que coja el mensaje de un fichero pasando el nombre de fichero con la opción `--file` :

```
$ svn commit --file logmsg
Sending        sandwich
Transmitting file data .
Committed revision 4.
```

Si usted falla en especificar cualquiera de las opciones `--message` o `--file`, entonces Subversion lanzará automáticamente su editor favorito (según lo definido en la variable de entorno `$EDITOR`) para redactar un mensaje de registro.

Si está en su editor escribiendo un mensaje de registro y decide que quiere cancelar su envío, usted puede quitar su editor sin guardar los cambios. Si ya ha guardado su mensaje de registro, simplemente borre el texto y salve otra vez.

```
$ svn commit
Waiting for Emacs...Done

Log message unchanged or not specified
a)bort, c)ontinue, e)dit
a
$
```

El repositorio no sabe ni cuida si sus cambios tienen algún sentido en su totalidad; solo comprueba para asegurarse que nadie haya cambiado cualquiera de los mismos ficheros que usted mientras usted no miraba. Si alguien *ha* hecho esto, el envío entero fallará con un mensaje informándole que uno o más de sus ficheros está fuera-de-fecha:

```
$ svn commit --message "Add another rule"
```

```
Sending        rules.txt
svn: Commit failed (details follow):
svn: Out of date: 'rules.txt' in transaction 'g'
```

En este punto, necesita ejecutar **svn update**, ocupándose con cualquier fusión o conflicto que resulte y procure enviarlo otra vez.

Eso cubre el ciclo básico de trabajo para usar Subversion. Hay muchas otras características en Subversion que usted puede usar para administrar su repositorio y copia de trabajo, pero puede pasar fácilmente usando solo los comandos que hemos visto hasta ahora en este capítulo.

Examinando el historial

Como hemos mencionado anteriormente, el repositorio es como una máquina del tiempo. Este mantiene un expediente de cada cambio enviado, y le permite explorar este historial examinando versiones anteriores de ficheros y directorios así como los metadatos que los acompañan. Con un único comando de Subversion, puede descargar el repositorio (o restaurar una copia de trabajo existente) exactamente como era en cualquier fecha o número de revisión en el pasado. Sin embargo, a veces solo desea *mirar* al pasado en vez de *ir* al pasado.

Hay varios comandos que pueden proporcionarle datos históricos del repositorio:

svn log
: Le muestra amplia información: mensajes de registro unidos a las revisiones, y que ruta de fichero cambió en cada revisión.

svn diff
: Le muestra los detalles específicos de cómo cambió un fichero en un cierto plazo.

svn cat
: Este se utiliza para recuperar cualquier fichero tal como existió en un un número de revisión particular y lo muestra en su pantalla.

svn list
: Muestra los ficheros en un directorio para cualquier revisión dada.

svn log

Para descubrir información sobre la historia de un fichero o directorio, use el comando **svn log**. **svn log** le proporcionará un registro de quién hizo cambios a un fichero o directorio, en qué revisión cambió, la hora y fecha de esa revisión, y, si fue proporcionado, el mensaje de registro que acompañaba al envío.

```
$ svn log
------------------------------------------------------------------------
r3 | sally | Mon, 15 Jul 2002 18:03:46 -0500 | 1 line

Added include lines and corrected # of cheese slices.
------------------------------------------------------------------------
r2 | harry | Mon, 15 Jul 2002 17:47:57 -0500 | 1 line

Added main() methods.
------------------------------------------------------------------------
r1 | sally | Mon, 15 Jul 2002 17:40:08 -0500 | 1 line

Initial import
------------------------------------------------------------------------
```

Observe que los mensajes de registro son impresos en *orden cronológico inverso* por defecto. Si desea ver un rango diferente de revisiones en un orden particular, o solo una única revisión, pase la opción `--revision` (`-r`):

```
$ svn log --revision 5:19    # shows logs 5 through 19 in chronological order
$ svn log -r 19:5            # shows logs 5 through 19 in reverse order
$ svn log -r 8               # shows log for revision 8
```

También puede examinar el historial de registro de un único fichero o directorio. Por ejemplo:

```
$ svn log foo.c
…
$ svn log http://foo.com/svn/trunk/code/foo.c
…
```

Ésto mostrará los mensajes de registro *solo* para esas revisiones en las cuales el fichero de trabajo (o URL) cambió.

Si desea aún más información sobre un fichero o directorio, **svn log** también toma una opción `--verbose` (`-v`). Porque Subversion le permite mover y copiar ficheros y directorios, es importante poder seguir cambios de la ruta del fichero en el sistema de ficheros, así en modo detallado, **svn log** incluirá una lista de rutas de fichero cambiadas en una revisión en su salida:

```
$ svn log -r 8 -v
------------------------------------------------------------------------
r8 | sally | 2002-07-14 08:15:29 -0500 | 1 line
Changed paths:
M /trunk/code/foo.c
M /trunk/code/bar.h
A /trunk/code/doc/README

Frozzled the sub-space winch.

------------------------------------------------------------------------
```

¿Por qué svn log me da una respuesta vacía?

Después de trabajar con Subversion un poco, la mayoría de los usuarios tendrán algo como esto:

```
$ svn log -r 2
------------------------------------------------------------------------
$
```

A primer vistazo, esto parece como un error. Pero recuerde que mientras las revisiones son repository-wide, **svn log** funciona sobre una ruta en el repositorio. Si no provee ninguna ruta de fichero, Subversion usa el directorio de trabajo actual como destino por defecto. En consecuencia, si usted está trabajando en un subdirectorio de su copia de trabajo y procurando registrar una revisión en la cual ni ese directorio ni cualquiera de sus hijos fue cambiado, Subversion le dará un registro vacío. Si desea ver qué cambió en esa revisión, intente indicando **svn log** directamente en lo más alto del URL de su repositorio, como en **svn log -r 2 http://svn.collab.net/repos/svn**.

svn diff

Ya hemos visto **svn diff** antes—éste muestra las diferencias de fichero en un formato unificado del diff; fue utilizado para mostrar las modificaciones locales hechas a nuestra copia de trabajo antes de enviarlas al repositorio.

De hecho, resulta que hay *tres* usos distintos para **svn diff**:

- Examinar cambios locales
- Comparar su copia de trabajo con la del repositorio
- Comparar repositorio con repositorio

Examinando cambios locales

Como hemos visto, invocando **svn diff** sin argumentos comparará sus ficheros de trabajo con las copias "prístinas" almacenadas en el área `.svn`:

```
$ svn diff
Index: rules.txt
===================================================================
--- rules.txt (revision 3)
+++ rules.txt (working copy)
@@ -1,4 +1,5 @@
 Be kind to others
 Freedom = Responsibility
 Everything in moderation
-Chew with your mouth open
+Chew with your mouth closed
+Listen when others are speaking
$
```

Comparando copia de trabajo con repositorio

Si se pasa un único `--revision` (`-r`) número, entonces su copia de trabajo es comparada con la revisión especificada del repositorio.

```
$ svn diff --revision 3 rules.txt
Index: rules.txt
===================================================================
--- rules.txt (revision 3)
+++ rules.txt (working copy)
@@ -1,4 +1,5 @@
 Be kind to others
 Freedom = Responsibility
 Everything in moderation
-Chew with your mouth open
+Chew with your mouth closed
+Listen when others are speaking
$
```

Comparando repositorio con repositorio

Si dos números de revisión, separados por una coma, son pasados vía `--revision` (`-r`), entonces las dos revisiones son comparadas directamente.

```
$ svn diff --revision 2:3 rules.txt
Index: rules.txt
===================================================================
--- rules.txt (revision 2)
+++ rules.txt (revision 3)
@@ -1,4 +1,4 @@
 Be kind to others
-Freedom = Chocolate Ice Cream
+Freedom = Responsibility
 Everything in moderation
 Chew with your mouth closed
$
```

No solo puede usar **svn diff** para comparar ficheros en su copia de trabajo con el repositorio, sino que si suministra una URL como argumento, usted puede examinar las diferencias entre elementos en el repositorio incluso sin tener una copia de trabajo. Esto es especialmente útil si desea inspeccionar cambios en un fichero cuando no tiene una copia de trabajo en su máquina local:

```
$ svn diff --revision 4:5 http://svn.red-bean.com/repos/example/trunk/text/rules.txt
…
$
```

svn cat

Si desea examinar una versión anterior de un fichero y no necesariamente las diferencias entre dos ficheros, puede usar **svn cat**:

```
$ svn cat --revision 2 rules.txt
Be kind to others
Freedom = Chocolate Ice Cream
Everything in moderation
Chew with your mouth closed
$
```

También puede redireccionar la salida directamente a un fichero:

```
$ svn cat --revision 2 rules.txt > rules.txt.v2
$
```

Probablemente se esté preguntando por qué no usamos **svn update --revision** para actualizar el fichero a la revisión más antigua. Hay algunas razones por las que preferimos usar **svn cat**.

Primero, usted puede querer ver las diferencias entre dos revisiones de un fichero usando un programa diff externo (quizás uno gráfico, o quizás su fichero está en un formato que la salida de un diff unificado es absurdo). En este caso, necesitará coger una copia de la revisión antigua, redireccionarla a un fichero, y pasar este y el fichero de su copia de trabajo a su programa diff externo.

A veces es más fácil mirar una versión más antigua de un fichero en su totalidad en comparación con las diferencias entre esta y otra revisión.

svn list

El comando **svn list** le muestra qué ficheros están en un directorio de un repositorio sin realmente descargar los ficheros a su máquina local:

```
$ svn list http://svn.collab.net/repos/svn
README
branches/
clients/
tags/
trunk/
```

Si desea un listado más detallado, pase la opción `--verbose` (`-v`) para obtener una salida como esta.

```
$ svn list --verbose http://svn.collab.net/repos/svn
   2755 harry          1331 Jul 28 02:07 README
   2773 sally               Jul 29 15:07 branches/
   2769 sally               Jul 29 12:07 clients/
   2698 harry               Jul 24 18:07 tags/
   2785 sally               Jul 29 19:07 trunk/
```

Las columnas le dicen la revisión en la cual el fichero o directorio fue modificado por última vez, el usuario qué lo modificó, el tamaño si este es un fichero, la fecha de la última modificación, y el nombre del objeto.

Una palabra final en el historial

Además de todos los comandos anteriores, usted puede usar **svn update** y **svn checkout** con la opción `--revision` para tomar una copia de trabajo entera “anterior en el tiempo” [8]:

```
$ svn checkout --revision 1729 # Checks out a new working copy at r1729
…
$ svn update --revision 1729 # Updates an existing working copy to r1729
…
```

Otros comandos útiles

Mientras que no son usados con tanta frecuencia como los comandos discutidos previamente en este capítulo, usted necesitará de vez en cuando estos comandos.

svn cleanup

Cuando Subversion modifica su copia de trabajo (o cualquier información en el interior de `.svn`), intenta hacerlo tan seguro como sea posible. Antes de cambiar cualquier cosa, escribe sus intenciones en un fichero de registro, ejecuta los comandos en el fichero de registro, entonces borra el fichero de registro (esto es similar en diseño a un sistema de ficheros transaccional). Si una operación de Subversion es interrumpida (si el proceso es matado, o si la máquina se cuelga, por ejemplo), los ficheros de registro permanecen en disco. Ejecutando de nuevo los ficheros de registro, Subversion puede completar la operación anteriormente empezada, y su copia de trabajo puede volver a estar en un estado consistente.

Y esto es exactamente lo que hace **svn cleanup**: busca en su copia de trabajo y ejecuta cualquier registro de sobra, eliminando bloqueos en el proceso. Si Subversion alguna vez le dice que alguna parte de su copia de trabajo está “bloqueada”, entonces éste es el comando que debería ejecutar. También, **svn status** mostrará una `L` al lado de los objetos bloqueados:

```
$ svn status
```

[8] ¿Ve? Le dijimos que Subversion era una máquina del tiempo.

```
  L     somedir
M       somedir/foo.c

$ svn cleanup
$ svn status
M       somedir/foo.c
```

svn import

El comando **svn import** es una manera rápida de copiar un árbol de ficheros sin versionar en el repositorio, creando directorios intermedios como sea necesario.

```
$ svnadmin create /usr/local/svn/newrepos
$ svn import mytree file:///usr/local/svn/newrepos/some/project
Adding         mytree/foo.c
Adding         mytree/bar.c
Adding         mytree/subdir
Adding         mytree/subdir/quux.h

Committed revision 1.
```

El ejemplo anterior copia el contenido del directorio `mytree` debajo del directorio `some/project` en el repositorio:

```
$ svn ls file:///usr/local/svn/newrepos/some/project
bar.c
foo.c
subdir/
```

Observe que después de que la importación esté acabada, el árbol original *no* se convierte en una copia de trabajo. Para empezar a trabajar, usted todavía necesita hacer **svn checkout** en una copia de trabajo fresca del árbol.

Sumario

Ahora hemos cubierto la mayoría de los comandos del cliente de Subversion. Las excepciones notables son ésas que se ocupan de la ramificación y de la combinación (vea Capítulo 4, *Crear ramas y fusionarlas*) y de las propiedades (vea "Propiedades"). Sin embargo, usted puede querer tomarse un momento para echar un ojo a Capítulo 9, *Referencia completa de Subversion* para tener una idea de todos los muchos comandos diferentes que tiene Subversion—y cómo puede usarlos para hacer su trabajo más fácil.