Control de Versiones con Subversion

Introducción

MADS Group - Departamento de Computación

Carlos Abalde (carlos@lfcia.org)
David Cabrero (cabrero@udc.es)
Laura Castro (laura@dc.fi.udc.es)

26 de febrero de 2007

Control de Versiones con Subversion

Contenido

Sistemas de Control de Versiones

Uso elemental de Subversion

Características avanzadas

Bibliografía

Contenido

1. Sistemas de Control de Versiones
2. Uso elemental de Subversion
3. Características avanzadas
4. Bibliografía

Motivación

Lo único constante es el cambio.

Preguntas:

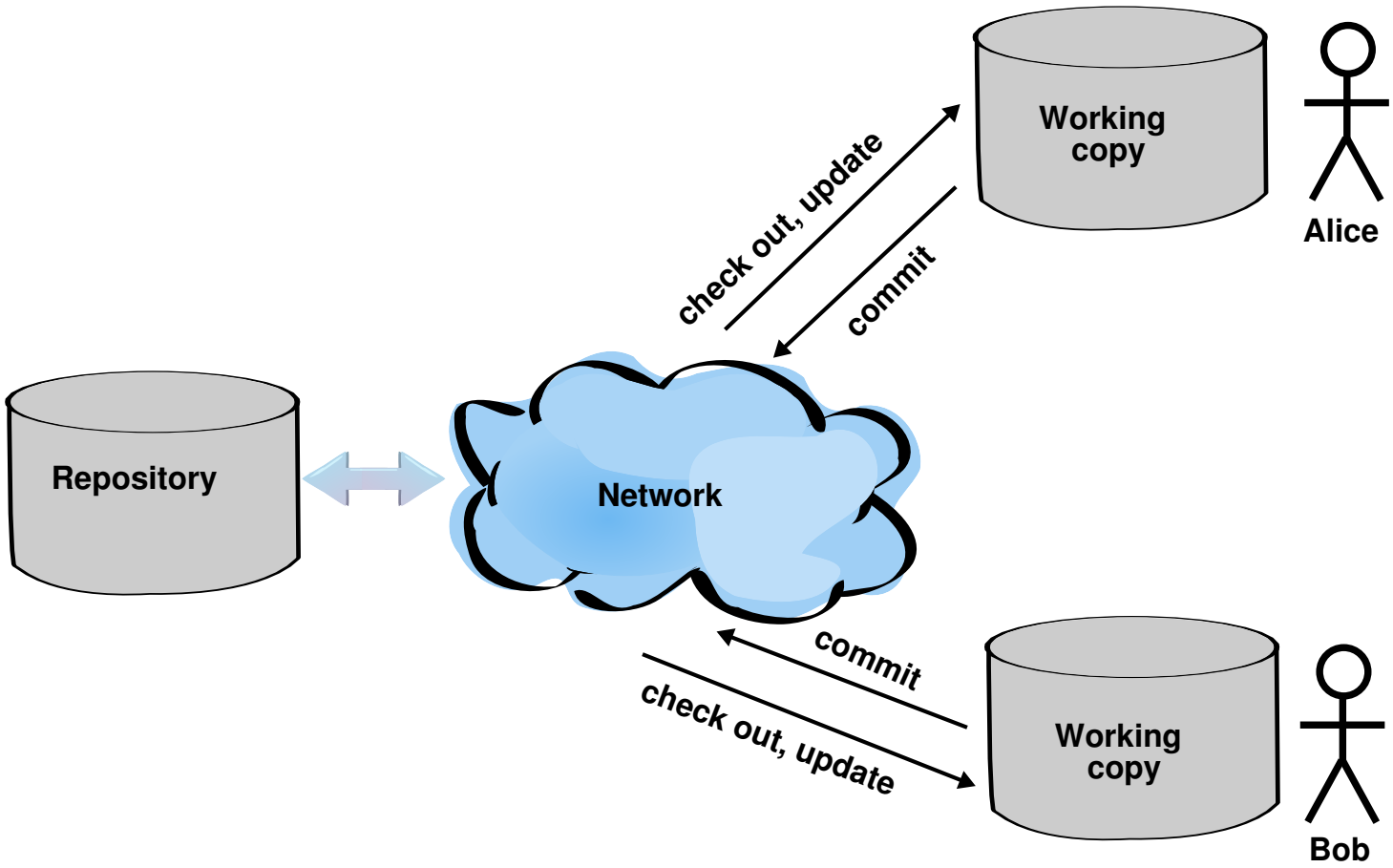
- ¿ Cómo controlo las versiones ?
- ¿ Qué cambios hay entre versiones ?
- ¿ Quién hizo tal cambio ?
- ¿ Por qué se hizo el cambio X ?
- ¿ Cómo vuelvo a una versión estable ?
- ¿ Cómo controlo los cambios simultáneos de varias personas ?

Respuesta (Sistema de Control de Versiones).

¿ Qué es un control de versiones ?

- Un sistema para la gestión y almacenamiento de todas las versiones de los diferentes componentes de un proyecto de desarrollo.
- Características:
  - Existe un repositorio (centralizado o distribuido) común donde se guardan todas las versiones
  - Los desarrolladores trabajan en su copia local e incorporan las modificaciones al repositorio
  - El sistem registra las modificaciones en archivos y sus comentarios asociados
  - El sistema gestiona las distintas versiones almacenadas en el repositorio
  - El sistema permite el desarrollo de varias versiones simultáneamente

Despliegue típico



Contenido
Sistemas de Control de Versiones
Uso elemental de Subversion
Características avanzadas
Bibliografía

Numeración, etiquetas y ramas

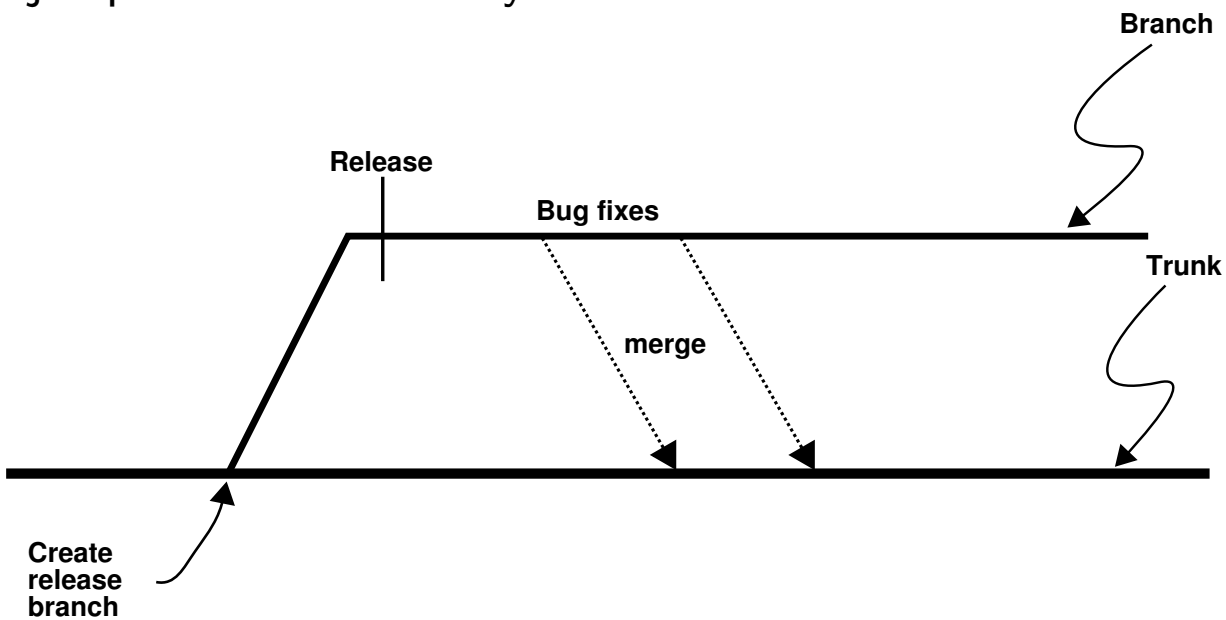
- Numeración de versiones
  - Depende del sistema (por fichero, por commit)
- Etiquetas
  - Nombres en vez de números
- Ramas
  - Varias líneas de desarrollo simultáneas
  - Ejemplo: Rama estable y rama de desarrollo

Numeración, etiquetas y ramas

- Numeración de versiones
  - Depende del sistema (por fichero, por commit)
- Etiquetas
  - Nombres en vez de números
- Ramas
  - Varias líneas de desarrollo simultáneas
  - Ejemplo: Rama estable y rama de desarrollo

Numeración, etiquetas y ramas

- Numeración de versiones
  - Depende del sistema (por fichero, por commit)
- Etiquetas
  - Nombres en vez de números
- Ramas
  - Varias líneas de desarrollo simultáneas
  - Ejemplo: Rama estable y rama de desarrollo



Locking

- ¿ Qué ocurre cuando dos desarrolladores modifican el mismo fichero ?
- Dos aproximaciones:
  - Strict locking.
  - Optimistic locking.

CVS

- *Concurrent Version System*
- Creado a partir del proyecto de Dick Grune (1986), siendo aun hoy un desarrollo activo
- Uso muy extendido
  - Por ejemplo, Sourceforge
- Tiene algunas limitaciones ligadas a su diseño inicial que empujan a su reemplazo por sistemas alternativos
- Alternativas a CVS solucionan algunos de sus problemas,
  - *Commits* atómicos
  - Identificador de archivo distinto de *path* de archivo
  - Permiten trabajo desconectado
  - Soporte de *branchs* mejorado
  - ...

Alternativas a CVS

- Arch
  - `http://arch.fifthvision.net`
  - En teoría, soporta un gran conjunto de funcionalidades
  - Escrito en *shell scripts*
- GIT
  - `http://www.kernel.org/git/`
  - Usado en el desarrollo del *kernel* de Linux
  - Distribuido, no centralizado. Substituto de bitkeeper
- Perforce
  - `http://www.perforce.com`
  - Comercial.
- Subversion
  - `http://subversion.tigris.org`
  - "A compelling replacement for CVS"
  - Creado a semejanza de CVS, eliminado sus carencias

Fundamentos (I)

- Diferencias entre ficheros

```
$ cat hola1.c
#include <stdio.h>

int main(int argc, char* argv[]) {
        printf("Hola mundo");
}
$ cat hola2.c
#include <stdio.h>

int main(int argc, char* argv[]) {
        printf("Hola mundo!\n");
}
$ diff hola1.c hola2.c
4c4
<       printf("Hola mundo");
---
>       printf("Hola mundo!\n");
```

Fundamentos (II)

- Parches

```
$ diff hola1.c hola2.c > parche
$ patch hola1.c < parche
$ cat hola1.c
#include <stdio.h>

int main(int argc, char* argv[]) {
        printf("Hola mundo!\n");
}
```

Modus operandi

1. Para un proyecto nuevo, se crea un repositorio.
2. Cada desarrollador:
   1. Obtiene una copia del repositorio (copia local).
   2. Trabaja sobre su copia local.
   3. Eventualmente:
      - Manda sus cambios al respositorio.
      - Actualiza su copia local con los cambios de otros desarrolladores.

Comandos básicos

- Desde el punto de vista de los usuarios
  - `svn`
- Desde el punto de vista del administrador
  - `svnadmin`
- Otros comandos
  - `svnlook`
  - `svnserve`
  - `svnversion`
- Ayuda
  - `svn help`
  - `svn help copy`
  - `svnadmin help`

Creación de un repositorio

- Creamos un repositorio `test` en el directorio *path_to_repository*/SVN/
  `svnadmin create` *path_to_repository*`/SVN/test`
- Atención a *path_to_repository*`/SVN/test/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."*

- URL (local): `file:///`*path_to_repository*`/SVN/test`
- La estructura de directorios es libre, sin embargo
  - El uso natural es crear un repositorio para cada proyecto. A diferencia de en CVS, la numeración de versiones es global para cada repositorio, y por ello la existencia de repositorios con más de un proyecto no es razonable

Inserción de datos en el repositorio

- Podemos comenzar a añadir ficheros al repositorio desde cero, pero
- Si ya disponemos de una estrucutra inicial, podemo usarla para popular el repositorio
  - `svn import PATH URL`
- En nuestro caso
  - `svn import` *`path_inital_files`* `file:///`*`path_to_repository`*`/SVN/test`
- Variable de entorno `EDITOR`

Descarga de una copia de trabajo

- Para comenzar a trabajar con el repositorio es necesario descargar nuestra copia de trabajo personal
  - `svn checkout URL PATH`
  - HEAD (última versión) en el repositorio
- Directorio `.svn`
- Manipulación básica de archivos
  - `svn copy SRC DEST`
  - `svn move SRC DEST`
  - `svn add PATH`
  - `svn rm TARGET`
  - `svn mkdir PATH`

Edición de archivos

- Sobre la copia de trabajo los archivos pueden modificarse con normalidad
- Para comprobar los cambios hechos respecto a la copia de trabajo de partida
  - `svn diff [TARGET]`
- Para descartar los cambios hechos y volver a la versión inicial de trabajo
  - `svn revert TARGET`
- Para recordar que cambios que se han hecho sobre la copia de trabajo
  - `svn status`
  - **M**odified, **A**dded, **D**eleted...
- Actualización del repositorio
  - `svn commit`
  - Documentar el cambio! (opción `-m`)
  - Debe funcionar! Estamos trabajando en grupo!

Logs, versiones, etc

- Para mostrar el *log* de cambios
  - `svn log [TARGET]`
- Para descargar copias de trabajo de versiones específicas
  - `svn -r 2 checkout file:///home/carlos/SVN/dsi0405_p1 dsi`
  - Algunos nombres de versiones
    - `HEAD`: ultima versión en el repositorio
    - `BASE`: versión inicial en la copia de trabajo
    - ...
- Para comparar versiones específicas
  - `svn -r 2:4 diff README.txt`
- Clientes gráficos
  - eSvn, rapidsvn, websvn...

Propiedades (metadatos)

Contenido
Sistemas de Control de Versiones
Uso elemental de Subversion
Características avanzadas
Bibliografía

- Subversion almacena metadatos (properties) en el repositorio.
- Se tratan igual que los datos (commits, updates, conflictos, ...)
- Instrucciones de svn `propset`, `propedit`, `proplist`, `propget`.

```
svn propset checked-by "M. Mason" Number.txt
```

- Propiedades especiales: `svn:.*`
- Ejemplo: `svn:eol-style`
  Valores: `native`, `CRLF`, `LF`, `CR`

Trabajo en red

- El uso normal de Subversion involucra a un conjunto de desarrolladores distribuidos y trabajando de forma concurrente
- Aparte de los repositorios locales, es necesario proveer de modos de acceso en red
  - `svnserve`
    - Ligero, simple, limitado...
    - `svn://`
  - Apache + WebDAV
    - Versátil, control de acceso, SSL...
    - `http://`

Branchs

Contenido
Sistemas de Control de Versiones
Uso elemental de Subversion
Características avanzadas
Bibliografía

- Un *branch* es una línea de desarrollo que coexiste con otras líneas de desarrollo y que en el pasado comparten una historia común
- Ejemplo: una vez alcanzada la versión 1 de un programa es deseable dividir el repositorio de forma que una línea de desarrollo de dedique a pequeñas correcciones sobre la versión 1 (mantenimiento), mientras que otra línea de desarrollo avance en el desarrollo de la versión 2 del programa
- `svn copy dsi dsi0405`

Mezclas y resolución de conflictos

Contenido
Sistemas de Control de Versiones
Uso elemental de Subversion
Características avanzadas
Bibliografía

- Escenarios de mezcla donde pueden surgir conflictos
  - Desde la descarga de la copia de trabajo (*checkout*) hasta la confirmación de los cambios en la misma (*commit*) es posible que otros usuarios hayan modificado y confirmado cambios sobre archivos también modificados por nosotros
  - En ocasiones es deseable poder integrar los cambios hechos en una línea de desarrollo fruto de un *branch* en la línea de desarrollo principal
- Ante un conflicto, `svn commit` fallará anulando el *commit*
- Se debe ejecutar `svn update` de forma que
  - Marcadores del conflicto son colocados en el archivo original
  - Para cada archivo en conflicto, se crean tres adicionales en la copia de trabajo
    - Una copia del archivo modificado
    - Una copia del archivo en el repositorio
    - Una copia del archivo en el momento del *checkout*
- El conflicto debe resolverse con la ayuda de los archivos creados y eliminarlos una ver resuelto. A continuación puede hacerse el `commit`

Bibliografía I

[Mas06] Mike Mason.
Pragmatic Version Control Using Subersion.
The Pragmatic Starte Kit. The Pragmatic Programmers LLC, 2006.

[svn05a] Subversion project home.
`http://subversion.tigris.org`, 2005.

[svn05b] Version control with subversion.
`http://svnbook.red-bean.com`, 2005.