Uso práctico de CVS

Carlos Hernando

Capítulo de Estudiantes ACM

chernando@acm.asoc.fi.upm.es

Actualmente vivimos en la filosofía *OpenSource* y hacer que nuestro código sea libre y abierto nos permite enviarlo a lugares en los que nunca antes podríamos haber imaginado. Sin embargo a la hora de mantener software de esta forma surgen problemas para poder poner en contacto y coordinar a los desarrolladores que colaboran en el proyecto. Por suerte contamos con una herramienta que hará las cosas más fáciles, CVS.

1. Sobre este documento

1.1. Licencia

1.2. Descargar el documento

Las últimas versiones de este documento se publican en diferentes formatos en: HTML (http://acm.asoc.fi.upm.es/~chernando/doc/cvs/), DVI (http://acm.asoc.fi.upm.es/~chernando/doc/cvs/usocvs.dvi), PDF (http://acm.asoc.fi.upm.es/~chernando/doc/cvs/usocvs.pdf), PS (http://acm.asoc.fi.upm.es/~chernando/doc/cvs/usocvs.ps) y RTF (http://acm.asoc.fi.upm.es/~chernando/doc/cvs/usocvs.rtf).

1.3. Estado del documento

En desarrollo. Se agradece cualquier sugerencia / correción / aporte a mi e-mail (mailto:chernando@acm.asoc.fi.upm.es)

2. Introducción a CVS

CVS, Concurrent Versions System, es una aplicación cliente/servidor que se encarga de mantener un *repositorio* de software centralizado que es actualizado y distribuido desde y hacia las copias locales de los desarrolladores. Es una utilidad muy sencilla de usar ;-)

2.1. ¿Cómo funciona?

El funcionamiento es muy simple: el desarrollador se conecta con el servidor CVS y le pide la última versión disponible del proyecto, en este paso el desarrollador puede ver que cambios se han realizado respecto a su versión local y los conflictos que pudiera ocasionar el código que el desarrollador ha realizado en su copia local con el código que ya está disponible en el servidor. En caso de que el código no sea problemático se modifican los ficheros locales respentando los cambios del desarrollador.

El servidor CVS se encarga de manejar el histórico de lo que ocurre, mantener un registro de los cambios realizados a cada fichero y de servirlos según las necesidades del desarrollador. Además gestiona diversas utilidades para controlar en que ficheros se esta trabajando, notificar a los autores de los ficheros de los cambios... y muchas más cosas pues es posible añadir scripts al funcionamiento a su configuración.

El cliente CVS del desarrollador se encarga de obtener las últimas versiones disponibles (o las que necesite), de confrontarlas con las copias locales y de crear una copia local de los ficheros del proyecto que sean editables por el desarrollador. Y por supuesto de añadir el código del desarrollador al proyecto.

2.2. ¿Qué estructura tiene el servidor?

El servidor de CVS se encarga de mantener los ficheros que conforman nuestro proyecto de tal forma que se mantienen los cambios realizados hasta el momento, esto nos permite consultar versiones anteriores de un mismo fichero de una forma rápida y sencilla.

En la raíz del servidor se encuentran los *módulos* que están disponibles, normalmente cada módulo corresponde a un directorio en el raíz. Los directorios que cuelgan de un directorio del raíz se consideran *submódulos*, es decir, la estructura es así:

```
/Raíz_CVS
|
|---/Módulo_1
|   |
|   |---/Submódulo_1 del Módulo_1
|   |---/Submódulo_2 del Módulo_1
|
|---/Módulo_2
|---/CVSROOT[1]
```

Los ficheros que el servidor almacena son del tipo `fichero.ext,v`. En estos ficheros se encuentra nuestro fichero original, los cambios realizados en las versiones anteriores y más información. Al manejarse en un formato especial debemos tener cuidado al enviar ficheros especiales como binarios y formatos de texto estricto ya que es posible que alguna combinación pueda ser interpretada por el servidor, veremos más adelante la manera de marcar estos ficheros especiales.

2.3. ¿Qué estructura tiene el cliente?

La copia local que realiza el cliente de CVS solamente incluye el fichero, que es lo que el desarrollador necesita pues es el código con el que trabaja. Una peculiaridad de la copia local es que todos los directorios tienen un subdirectorio `CVS`, en él se guardan tres archivos: `Entries, Repository, Root`. En estos tres archivos se guarda información relacionada al repositorio de dónde se obtuvo el proyecto. Es decir, cada directorio se puede manejar de forma completamente independiente y sin necesidad de especificar el repositorio de origen.

```
/Directorio_Local
|
|---/Módulo_1
    |
    |---/Submódulo_1 del Módulo_1
    |   |---/CVS
    |
    |---/Submódulo_2 del Módulo_1
    |   |---/CVS
    |
    |---/CVS
```

2.4. ¿Para qué se usa CVS?

CVS se puede utilizar en casi todo proyecto que implique una colaboración entre desarrolladores (aunque solamente haya *uno*) simplemente darle un vistazo a SourceForge (http://sourceforge.net/) para descubrir la gran cantidad de proyectos que lo están usando.

Sin embargo no se utiliza solamente para desarrollo de software y manejo de código fuente. CVS está siendo utilizado para mantener sitios web, como GNOME Hispano (http://www.es.gnome.org/), o para mantener documentación, como Debian Documentation Proyect (http://www.debian.org/doc/ddp) y para grandes proyectos como los ports que forman parte de FreeBSD (http://www.freebsd.org/). Incluso se puede utilizar para mantener copias de trabajo en la máquina de casa, en la del trabajo, en el notebook, éct...

2.5. Instalación de CVS

Solamente es necesario instalar el paquete cvs: **apt-get install cvs**, Debian rulez! ;-). En otras distribuciones existen los correspondientes paquetes RPM o los tgz cuya instalación es muy sencilla. El cliente y el servidor vienen en el mismo paquete por lo que nos preguntará en que directorio se situará el raíz del servidor CVS, recomendado en /var/cvs, /var/lib/cvs o en /home/cvs.

Para iniciar un repositorio en nuestra maquina será necesario ejecutar: **cvs -d $CVSROOT init**, siendo `$CVSROOT` la variable que contiene el directorio de nuestro respositorio. Después de ejecutarlo se creará el directorio CVSROOT en la raíz del CVS, *no confundir el directorio raíz del CVS, $CVSROOT, con el directorio CVSROOT que hay en la raíz* (Esto último puede hacerse automáticamente en la instalación del paquete cvs).

2.6. El comando cvs

Antes de entrar en materia un par de notas sobre el uso del commando **cvs**:

cvs [opcion_globales] operación [opcion_operación | fichero | módulo]

En opciones globales normalmente utilizaremos **-q** para limitar la salida, **-Q** para que la salida se reduza a lo mínimo, **-d $CVSROOT** para indicar la localización del repositorio y por último **-z N** siendo N el nivel de compresion al que queremos someter a los datos que se van a intercambiar, normalmente de 3 a 5.

Las operaciones y sus opciones las veremos a lo largo de este documento. Las más comunes son: **update, checkout, commit**.

A lo largo de esta documentación en algunos ejemplos después de la operación viene la ruta hasta un fichero o no hay nada. Si al comando **cvs** se indica una ruta *solamente* actuará en ese directorio o fichero, por el contrario si se le ejecuta sin especificar ninguna ruta actuará sobre el directorio de trabajo actual y de forma recursiva en sus subdirectorios.

3. Curioseando un repositorio de CVS

La mejor forma de empezar a colaborar en un proyecto es tener su código. En este apartado vamos a conectarnos a un servidor CVS, descargaremos uno de sus módulos y lo mantendremos actualizado según pase el tiempo.

3.1. Formas de identificarse en un servidor CVS

Normalmente el servidor de CVS está escuchando en el puerto 2401 `pserver` y es la configuración por defecto de los clientes, por ejemplo: **cvs -d :pserver:anoncvs@servidor.dom:/home/cvs update**.

Esta identificación se realiza por texto plano, nos pedirá una contraseña asociada al usuario anoncvs. Este tipo de identificacion recuerda la contraseña del usuario por lo que solo será necesario logearse una sola vez (en el fichero $HOME/.cvspass). Este método es inseguro para los desarrolladores del proyecto si bien para permitir el acceso al código a todo el mundo es una buena solución.

3.1.1. Usando SSH para conectarnos con el servidor

Por motivos de seguridad se recomienda a los desarrolladores (y a todo aquel que tenga cuenta en el servidor) el uso de SSH para establecer comunicaciones con el servidor CVS, este tipo de identificación obliga a introducir

la contraseña siempre que se realice una operación con el servidor de CVS.

Para poder realizar la conexión con SSH será necesario fijar el valor de la variable `CVS_RSH` a `SSH` y cambiar la método de conexión `pserver` a `ext`. Es interesante fijar las siguientes variables en el profile para economizar las pulsaciones (y la vida útil del teclado ;-))

```
charlie$ CVS_RSH=ssh
charlie$ CVSROOT=:ext:$USER@servidor.dom:/home/cvs
charlie$ export CVS_RSH CVSROOT
```

A partir de este momento el cliente de cvs tomará por defecto conectar utilizando ssh y al repositorio disponible en servidor.dom identificándome como $USER.

3.1.2. Desde un repositorio de nuestra propia máquina

Si el repositorio al que queremos acceder reside en el mismo sistema en el que trabajamos no será necesario especificar el método de conexión ni al sistema al que se accede, solamente es necesario especificar la ruta hasta el repositorio.

Para acceder al repositorio dentro de mi sistema utilizo **export CVSROOT=/var/lib/cvs** o la opción **-d /var/lib/cvs** y la operación que quiera realizar.

3.2. Descargar un proyecto del repositorio

Para descargar un proyecto del repositorio utilizaremos el comando **checkout** o su abreviatura **co** y el nombre del módulo a descargar.

Supongamos que queremos colaborar en el proyecto VIM-ES (http://helvete.escomposlinux.org/vimes/), que se encarga de traducir la documentación existente del editor VI, para ello vamos a descargar su proyecto `vimes`.

```
charlie$ pwd
/home/charlie/cvs

charlie$ cvs -z3 -d \
  :pserver:anoncvs:anoncvs@helvete.escomposlinux.org:/var/cvs/vim \
  checkout vimes
cvs server: Updating vimes
U vimes/DIRECTORIOS
U vimes/PROGRESO
cvs server: Updating vimes/beer
cvs server: Updating vimes/doc
U vimes/doc/autocmd.txt
U vimes/doc/change.txt
U vimes/doc/cmdline.txt
U vimes/doc/debugger.txt
```

```
# ...

charlie$ ls
vimes
```

El programa nos informa de los pasos que está siguiendo, las `U` significa que la copia local ha sido creada. El resultado es la creación de un directorio llamado como el módulo que hemos descargado y que contiene todos los archivos y subdirectorios que cuelgan del mismo. Prestar atención a la creación del directorio `CVS` en cada directorio del módulo.

3.3. Actualizando nuestra copia local

Para actualizar nuestra copia local respecto a los cambios del repositorio central se utiliza el comando **update**. El resultado de este comando nos informará de que archivos hayamos modificado localmente con una `M`, de los ficheros que se actualizan con `U`[2], los ficheros que tienen conflictos con `C` y un `?` en caso de que tengamos ficheros localmente que no existen en el repositorio.

Después de unas semanas curioseando el proyecto vimes queremos comprobrar que tenemos una copia local actualizada para ello haremos lo siguiente:

```
charlie$ pwd
/home/charlie/cvs/vimes
# Como ya estoy en el directorio vimes no será necesario espeficar
# ni el repositorio ni el módulo pues están en el directorio CVS

charlie$ cvs -z3 update
cvs server: Updating .
U ./PROGRESO
cvs server: Updating beer
cvs server: Updating doc
cvs server: Updating doc-6.0
# ...
```

El resultado, hemos actualizado el fichero `PROGRESO` (y algunos archivos más) y hemos revisado el estado de todo.

Puede ocurrir que en algún momento cvs nos informe de la existencia de un fichero que no volverá a ser pertinente en el proyecto, es decir, ha sido eliminado. El cliente nos mostrará el mensaje y lo borrará.

Con los directorios no ocurre lo mismo. Si un directorio se crea en un proyecto depués de nuestro primer checkout no se nos notificará su existencia (tendremos que estar informados de antemano), por lo que tendremos que volver a ejecutar un **cvs checkout módulo** si queremos trabajar con el nuevo directorio creado en el proyecto. Si un directorio después de una actualización se queda vacio no se borra automáticamente. Para borrar los directorios vacios añadiremos la opción **-P**, quedando así: **cvs update -P**.

4. Trabajando con CVS

Es momento de colaborar activamente en el proyecto, para ello necesitaremos tener cuenta de desarrollador en el servidor CVS ya que la cuenta `anoncvs` no suele tener privilegios para mandar código al repositorio.

4.1. Subiendo nuestras modificaciones

Una vez que hemos realizado las modificaciones adecuadas a nuestros ficheros locales procederemos a actualizar el repositorio y añadiremos un pequeño comentario sobre el motivo del cambio. Para eso utilizaremos el comando **commit** con el modificador **-m "Comentario sobre el cambio realizado"**.

Como no tengo cuenta en el proyecto vimes voy a cambiar a otro proyecto local ;-) . En este caso he modificado unas cosas en el fichero `shell.sgml` y voy a actualizar el repositorio. Para ello primero actualizo mi copia con **update** (lo que me permite saber si alguien más ha modificado el mismo archivo y puede surgir conflicto) y después lo subo al repositorio.

```
charlie$ pwd
/home/charlie/cvs

charlie$ cvs update doc-general
# ...
M doc-general/shell.sgml
# Me notifica que poseo una versión local modificada respecto al
#  repositorio

# Voy a subir mis modificaciones
charlie$ cvs commit -m "Actualizada seccion 2" doc-general/shell.sgml
Checking in shell.sgml;
/home/cvs/doc-general/shell.sgml,v  <--  shell.sgml
new revision: 1.4; previous revision: 1.3
done
# La actualización ha sido correcta
```

4.1.1. Problemas al actualizar archivos

Cuando dos desarrolladores están trabajando sobre el mismo archivo es posible que que en algún momento las modificaciones que realiza cada uno de ellos coincida. Por lo que cuando el segundo actualice el archivo CVS le devuelva una linea de error mostrando lo que tiene la versión local y lo que tiene la versión del CVS.

Después de trabajar en unos cambios en el fichero `menu.php` quiero subir mi versión al servidor, lo que no sé es que otro desarrollador ya ha realizado cambios en la misma sección del fichero. Al intentar actualizar el fichero cvs nos devolverá un error diciéndonos que es no es capaz de fusionar la versión local con la que hay en el repositorio. Este es el caso:

```
charlie$ cvs update
cvs update: Updating .
RCS file: /var/cvs/web/menu.php,v
```

```
retrieving revision 1.5
retrieving revision 1.6
Merging differences between 1.5 and 1.6 into menu.php
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in menu.php
C menu.php

charlie$
```

Si ahora edito el fichero `menu.php` veo que cvs ha realizado unos cambios en el fichero que son los que están dando problemas:

```
<<<<<<< menu.php
<a href="foro.cgi">Foro</a>
=======
<a href="libro.cgi">Libro de visitas</a>
>>>>>>> 1.6
```

Al realizar la actualización se ha remarcado que problema hay en el fichero, en este caso la versión disponible en servidor es la 1.6 pero la copia sobre la que realice los cambios era la 1.5. Se puede ver que el otro desarrollador y yo hemos modificado la misma línea del fichero (el otro desarrollador fue más rápido y actualizó antes) y CVS no sabe elegir entre las dos.

Simplemente se editan las líneas que dan problemas y se eliminan las marcas. Una vez el fichero sea correcto lo subimos al repositorio.

```
charlie$ cvs commit -m "Enlace al libro de visitas arreglado" menu.php
Checking in menu.php;
/var/cvs/web/menu.php,v  <-- menu.php
new revision: 1.7; previous revision: 1.6
done
```

4.1.2. Recuperando una versión anterior

En algún momento un desarrollador subirá al repositorio unas modificaciones que no sean muy acertadas o que podrían causar problemas con los otros archivos y tal vez nos interesase volver a una versión anterior del proyecto y hacer como si nunca se hubiese modificado.

Desarrollando `menu.php` otro desarrollador ha introducido una nueva función, sin embargo esa función ni es necesaria ni es segura por lo que es conveniente recuperar la versión anterior y olvidar esos cambios:

```
charlie$ cvs update -j 1.8 -j 1.7 menu.php
RCS file: /var/cvs/web/menu.php,v
```

```
retrieving revision 1.8
retrieving revision 1.7
Merging differences between 1.8 and 1.7 into menu.php

charlie$ cvs update
cvs update: Updating .
M menu.php

charlie$ cvs commit -m "Regreso a la version 1.7" menu.php
Checking in menu.php;
/var/cvs/web/menu.php,v  <--  menu.php
new revision: 1.9; previous revision: 1.8
done
```

4.2. Añadir archivos al repositorio

Durante el desarrollo de un proyecto es casi seguro que necesitaremos incluir más ficheros de los que estaban previstos al comienzo del mismo. Para incluir un fichero nuevo solamente será necesario utilizar el comando **add** y el fichero a añadir al repositorio. Para añadir un nuevo directorio se sigue el mismo método.

En este caso voy añadir un nuevo directorio y el archivo que cuelga de él. La única diferencia es que el directorio se añade automáticamente y para el fichero es necesario realizar **cvs commit -m "Comentario"**.

```
# Añadir un nuevo directorio: kde-gnome
charlie$ cvs add kde-gnome/
Directory /home/cvs/lpractico/kde-gnome added to the repository
# El directorio ya forma parte del repositorio.

# Añado el fichero kde.tex en el directorio kde-gnome
charlie$ cd kde-gnome
charlie$ cvs add kde.tex
cvs add: scheduling ’kde.tex’ for addtion
cvs add: use ’cvs commit’ to add this file permanently
# El fichero kde.tex ha sido programado para ser añadido al repositorio

# Confirmo la adición del fichero kde.tex
charlie$ cvs ci -m "Adicion: Texto de la comparativa de KDE-GNOME"
cvs commit: Examining .
RCS file: /home/cvs/lpractico/kde-gnome/kde.tex,v
done
Checking in kde.tex;
/home/cvs/lpractico/kde-gnome/kde.tex,v  <--  kde.tex
initial revision: 1.1
done
# La adicción del nuevo fichero ha sido correcta.
```

4.2.1. Ficheros especiales

En la introducción se mencionaba que CVS maneja los ficheros en un formato especial y que en algún caso era posible que nuestro fichero pudiera ser malinterpretado por el servidor al guardarlo (con lo que seguramente perderíamos el fichero). Este caso es muy común cuando se maneja binarios y algunos ficheros de texto (aunque esto es menos normal).

Trabajando en la documentación he decidido añadir una imágen de una captura del escritorio en formato eps. Para evitar problemas voy a añadirla marcando como binario el fichero:

```
charlie$ cvs add -k 'b' captura.eps
cvs add: scheduling file 'captura.eps' for addition
cvs add: use 'cvs commit' to add this file permanently
charlie$ cvs commit -m "Nueva captura de escritorio" captura.eps
RCS file: /home/cvs/lpractico/consola/captura.eps,v
done
Checking in captura.eps;
/home/cvs/lpractico/consola/captura.eps,v  <--  captura.eps
initial revision: 1.1
done
```

La opción **-k 'b'** marca el fichero como binario, es decir el servidor CVS lo guardará exactamente igual sin añadirle nada. Sin embargo no se nos informa de que el fichero ha sido marcado como binario.

4.3. Eliminar ficheros del repositorio

Los pasos a seguir para eliminar ficheros del repositorio son idénticos al caso de añadir ficheros con una sola peculiaridad, primero deberemos borrar localmentel fichero que queremos eliminar:

```
charlie$ rm variance.sgml

charlie$ cvs remove variance.sgml
cvs remove: scheduling 'variance.sgml' for removal
cvs remove: use 'cvs commit' to remove this file permanently

charlie$ cvs commit -m "Fichero obsoleto" variance.sgml
cvs commit: Examining .
Removing variance.sgml;
/home/cvs/doc-general/variance.sgml,v  <--  variance.sgml
new revision: delete; previous revision: 1.1.1.1
done
```

En el caso de los directorios tendremos que ser más cuidadosos, primero deberemos comprobar que no haya ficheros en él:

```
charlie$ cvs remove parte2/
cvs remove: Removing parte2/
cvs remove: file ‘parte2//index.html’ still in working directory
cvs remove: 1 file exists; remove it first

charlie$ rm parte2/index.html

charlie$ cvs remove parte2/
cvs remove: Removing parte2/
cvs remove: scheduling ‘parte2//index.html’ for removal
cvs remove: use ’cvs commit’ to remove this file permanently

charlie$ cvs commit -m "Parte asimilada ;)" parte2
cvs commit: Examining parte2
Removing parte2/index.html;
/home/cvs/lpractico/consola/parte2/index.html,v  <--  index.html
new revision: delete; previous revision: 1.1
done
```

Primero nos dice que todavía hay archivos en el directorio que queremos eliminar. Una vez borrados los ficheros que hay el directorio procedemos a su eliminación: el directorio `parte2` es eliminado inmediatamente mientras que el fichero `index.html` queda pendiente de confirmación con la operación **commit**.

4.3.1. CVS y los ficheros eliminados

CVS se encarga de mantener un histórico y todas las versiones de los arhivos que hay en los proyectos. Si esto es cierto no se pueden borrar los ficheros pues aunque no sean necesarios en las versiones actuales del proyecto si que lo fueron en etapas anteriores.

En realidad CVS no borra ningún fichero de repositorio simplemente lo mueve a un directorio llamado `Attic` y lo marca como “muerto” estando disponible en caso de tener que “revivirlo”.

```
charlie$ pwd
/home/cvs/doc-general/Attic

charlie$ ls
variance.sgml,v
```

5. Más posibilidades con CVS

CVS tiene muchas posibilidades para seguir el recorrido que ha seguido un proyecto ya que mantiene un fichero de registro con todo lo acontecido.

5.1. Marcar con etiquetas

Uno de los inconvenientes de trabajar con CVS es que siempre se manejan los ficheros con sus versiones y mantener una lista de versiones relaciona con todos los archivos puede resultar muy molesto en proyectos no muy grandes. Para evitar esto existe el comando **tag etiqueta** que nos permite enlazar diferentes versiones con un nombre simbólico.

Una semana después de empezar un proyecto y de desarrollo marco todo el contenido actual como `version_2001-12-31`:

```
charlie$ cvs tag version_2001-12-31
cvs tag: Tagging .
T index.sgml
T Makefile
T bibliografia.sgml
cvs tag: Tagging capitulo01
T capitulo01/shell.sgml
[...]
```

A partir de ahora puedo hacer referencia a este estado de versiones como `version_2001-12-31`, lo que me permitirá en un futuro recuperar todos los archivos tal y cómo estaban en el momento en el que los he etiquetado.

5.2. La historia de un fichero

Según se va desarrollando un proyecto es interesante saber que trayectoria ha seguido un fichero y aquí es donde entra la calidad de los *comentarios* introducidos por los desarrolladores. Para ello se utiliza el commando **log**.

En este ejemplo voy a consultar la historia del fichero `menu.php`:

```
charlie$ cvs log menu.php

RCS file: /var/cvs/web/menu.php,v
Working file: menu.php
head: 1.12
branch:
locks: strict
access list:
symbolic names:
  Diciembre_2001: 1.1.1.1
  chernando: 1.1.1
keyword substitution: kv
total revisions: 13;    selected revisions: 13
description:
----------------------------
revision 1.12
date: 2001/12/30 15:54:18;  author: chernando;  state: Exp;  lines: +1 -1
De siglinux a quienes y por tanto nuevo siggulfi con enlace
----------------------------
```

```
revision 1.11
date: 2001/12/30 12:01:56;  author: chernando;  state: Exp;  lines: +7 -7
Limando detalles 2
----------------------------
revision 1.10
date: 2001/12/30 11:57:15;  author: chernando;  state: Exp;  lines: +7 -5
Limando detalles 1
----------------------------
revision 1.9
date: 2001/12/30 11:40:43;  author: chernando;  state: Exp;  lines: +8 -9
Arreglado un enlace erroneo
[...]
```

5.3. Incluyendo la versión dentro del propio fichero

CVS maneja internamente los ficheros que se suben al repositorio, por ello es necesario controlar a los fichero binarios y algunos formatos de texto ya que es posible que CVS los interprete. En este caso vamos a ver cuales son algunas de las expresiones que se expanden automáticamente.

Un ejemplo muy sencillo es añadir `$Id$`, que será automáticamente por información relacionada a la última modificación, el autor de la misma y la versión. Este ejemplo es del XML de SIGLinux - GULFI (http://acm.asoc.fi.upm.es/siglinux/):

```
chernando@acm:~/cvs/gulfi-web/servicios$ head cvs.xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE webpage PUBLIC "-//laespiral.org//DTD LE-document 1.1//EN"
   "LE-document-1.1.dtd" [<!ENTITY menu SYSTEM "menu.xml"> ] >

<!-- $Id: cvs.xml,v 1.2 2002/04/03 18:16:11 chernando Exp $ -->
<webpage lang="es" xreflabel="CVS" folder="servicios" id="cvs">
  <title>CVS</title>
  <sect1>
    <title>CVS</title>
    <para>
# ...
```

En este caso, cada vez que sea actualiza el fichero la línea `<!-- $Id: cvs.xml,v 1.2 2002/04/03 18:16:11 chernando Exp $ -->` también se actualiza.

Existen otros patrones de sustitución como son: `$Author$, $Date$, $Header$, $Name$, $Log$,...`. Para más información consultar el manual de CVS.

Hay que tener en cuenta la precaución de dejar `$Id` dentro de un comentario dentro del lenguaje que estemos utilzando.

6. Administrando un servidor CVS

Una vez instalado el servidor CVS es momento de jugar con él y comenzar un proyecto completamente nuevo en nuestro propio servidor.

6.1. Creando un nuevo módulo

Para comenzar un proyecto es necesario, o al menos recomendable, partir de algo que tenga contenido. Una vez que tengamos un poco la estructura de nuestro módulo es el momento de llevarlo al servidor CVS con el comando **import -m "Comentario" nombre_módulo etiqueta_origen etiqueta_versión** y empezar a trabajar tirando del CVS.

```
charlie$ pwd
/home/charlie/documentacion-cvs

charlie$ cvs -d /home/cvs import -m "Documentacion sobre CVS" \
         cvs-doc chernando diciembre-2001
N cvs-doc/cvs.sgml
N cvs-doc/Makefile

No conflicts created by this import

charlie$
```

En este caso he creado el módulo `cvs-doc` y he creado dos ficheros: `cvs.sgml y Makefile` (las etiquetas de origen y versión no son importantes, su uso solamente interesa cuando estamos desarrollando para terceros). A partir de ahora se puede acceder al módulo normalmente.

```
charlie$ pwd
/home/charlie/cvs

charlie$ cvs -d /home/cvs checkout cvs-doc
cvs checkout: Updating cvs-doc
U cvs-doc/Makefile
U cvs-doc/cvs.sgml
```

6.2. Liberando una versión

A lo largo del desarrollo surgen versiones que son estables y pueden ser distribuidas, para evitar a los posibles usuarios el uso de CVS se realiza mediante el comando **export -t etiqueta -d ruta_destino módulo** una copia exacta de la versión etiquetada sin los directorios `CVS` que es distribuible si la empaquetamos con **tar** y **gzip** o **bzip2**.

Mi documentación sobre CVS es suficientemente madura como para poder distribuirla para ello realizo lo siguiente:

```
charlie$ cvs -d /home/cvs/ export -t v0.5 -d /home/charlie/cvsdoc-0.5 cvs-doc
cvs export: Updating /home/charlie/cvsdoc-0.5
U /home/charlie/cvsdoc-0.5/Makefile
U /home/charlie/cvsdoc-0.5/cvs.sgml
```

6.3. Manejando ramas

Contenido para las próximas vacaciones ;)

Notas

1. El directorio CVSROOT contiene la configuración y ficheros de información del servidor CVS.
2. La actualización implica descargar los cambios a la versión local, pero no actualizar los archivos locales al repositorio central