CVS

Concurrent Version System

Juan Quintela

quintela@dc.fi.udc.es

Universidade da Coruña

Mandrakesoft

¿Qué es CVS?

- CVS es un sistema de control de versiones de ficheros.

¿Qué es CVS?

- CVS es un sistema de control de versiones de ficheros.
- concurrente: permite varios usuarios simultaneos.

¿Qué es CVS?

- CVS es un sistema de control de versiones de ficheros.
- concurrente: permite varios usuarios simultaneos.
- Se usa cuando es importante ser capaz de acceder a versiones antiguas de esos ficheros.

¿Para qué sirve CVS?

Para responder a las preguntas:

- ¿Quién hizo un cambio?

¿Para qué sirve CVS?

Para responder a las preguntas:

- ¿Quién hizo un cambio?
- ¿Cuándo se hizo?

¿Para qué sirve CVS?

Para responder a las preguntas:

- ¿Quién hizo un cambio?
- ¿Cuándo se hizo?
- ¿Porqué se hizo?

¿Para qué sirve CVS?

Para responder a las preguntas:

- ¿Quién hizo un cambio?
- ¿Cuándo se hizo?
- ¿Porqué se hizo?
- ¿Qué otros cambios se hicieron al mismo tiempo?

Creando un repositorio

- Decidir donde lo vamos a colocar:
  `mkdir /home/master`

Creando un repositorio

- Decidir donde lo vamos a colocar:
  `mkdir /home/master`
- Inicializarlo:
  `cvs -d /home/master init`

Creando un repositorio

- Decidir donde lo vamos a colocar:
  `mkdir /home/master`
- Inicializarlo:
  `cvs -d /home/master init`
- Decirle a CVS donde esta el repositorio:
  `export CVSROOT=/home/master`

Incorporando un projecto existente

- Poner el projecto bajo control de revisiones:
  `cvs import hola HOLA HOLA_1`

Incorporando un projecto existente

- Poner el projecto bajo control de revisiones:
  `cvs import hola HOLA HOLA_1`
- Poner el projecto bajo control de revisiones con un comentario:
  `cvs import -m "Version inicial"hola HOLA HOLA_1`

Incorporando un projecto existente

- Poner el projecto bajo control de revisiones:
  `cvs import hola HOLA HOLA_1`
- Poner el projecto bajo control de revisiones con un comentario:
  `cvs import -m "Version inicial"hola HOLA HOLA_1`

Integrando cambios

- Ver que ficheros han cambiado con respecto al repositorio:
  `cvs release.`

Integrando cambios

- Ver que ficheros han cambiado con respecto al repositorio:
  `cvs release.`
- Integrar cambios que hemos hecho en el repositorio:
  `cvs commit -m "poner n final"ejemplo.c.`

Integrando cambios

- Ver que ficheros han cambiado con respecto al repositorio:
  `cvs release.`
- Integrar cambios que hemos hecho en el repositorio:
  `cvs commit -m "poner n final"ejemplo.c.`
- Integrar cambios del repositorio en nuestra copia:
  `cvs update.`

Update model

CVS se basa en el *update model*

- Primero actualizar nuestra copia al ultimo valor del repositorio:
  `cvs update`

Update model

CVS se basa en el *update model*

- Primero actualizar nuestra copia al ultimo valor del repositorio:
  `cvs update`
- Ver que no hay conflictos y que todo funciona bien:
  `cvs release`

Update model

CVS se basa en el *update model*

- Primero actualizar nuestra copia al ultimo valor del repositorio:
  `cvs update`
- Ver que no hay conflictos y que todo funciona bien:

Ver diferencias

- Ver cambios de nuestra copia con el repositorio: `cvs diff`.

Ver diferencias

- Ver cambios de nuestra copia con el repositorio: `cvs diff`.
- Ver cambios de un fichero `cvs diff nombre_fichero`.

Ver diferencias

- Ver cambios de nuestra copia con el repositorio: `cvs diff`.
- Ver cambios de un fichero `cvs diff nombre_fichero`.
- Ver diferencias con las versiones de ayer: `cvs diff -D'yesterday'`.

Ver diferencias

- Ver cambios de nuestra copia con el repositorio: `cvs diff`.
- Ver cambios de un fichero `cvs diff nombre_fichero`.
- Ver diferencias con las versiones de ayer: `cvs diff -D'yesterday'`.

Etiquetas

O como `marcar` que una versión funciona bien de forma que sea facil recuperarla en el futuro:

```
cvs tag VERSION_0_1
```


Etiquetas (II)

```
quintela$ cvs diff -rVERSION_0_0_1
Index: ejemplo.c
===================================================================
RCS file: /home/master/ejemplo3/ejemplo.c,v
retrieving revision 1.1.1.1
retrieving revision 1.2
diff -u -p -r1.1.1.1 -r1.2
--- ejemplo.c   7 May 2003 11:26:30 -0000       1.1.1.1
```

Anotaciones

O la respuesta a la pregunta: ¿ A quién le echo la culpa ?

```
quintela$ cvs annotate ejemplo.c

Annotations for ejemplo.c
***************
1.1          (quintela 07-May-03):
1.1          (quintela 07-May-03): #include <stdio.h>
1.1          (quintela 07-May-03): #include <stdlib.h>
1.1          (quintela 07-May-03):
1.1          (quintela 07-May-03): int main(void)
1.1          (quintela 07-May-03): {
1.2          (quintela 07-May-03):      printf("Hola mundo\n");
1.1          (quintela 07-May-03):      exit(0);
1.1          (quintela 07-May-03): }
```


Historia de un fichero

```
quintela$ cvs log ejemplo.c
RCS file: /home/master/ejemplo3/ejemplo.c,v
Working file: ejemplo.c
head: 1.2
branch:
locks: strict
access list:
symbolic names:
```

Añadir/Borrar ficheros

- Crear fichero

Añadir/Borrar ficheros

- Crear fichero
- `cvs add new_file`

Añadir/Borrar ficheros

- Crear fichero
- `cvs add new_file`
- `cvs commit new_file`

Borrar fichero

- `rm -f old_file`

Borrar fichero

- `rm -f old_file`
- `cvs del old_file`

Borrar fichero

- `rm -f old_file`
- `cvs del old_file`
- `cvs commit old_file`

Renombrar un fichero

- `mv old_name new_name`

Renombrar un fichero

- `mv old_name new_name`
- `cvs add new_name`

Renombrar un fichero

- `mv old_name new_name`
- `cvs add new_name`
- `cvs del old_name`

Renombrar un fichero

- `mv old_name new_name`
- `cvs add new_name`
- `cvs del old_name`
- `cvs commit -m "mv old_name new_name.ºld_name new_name`

Manejando Conflictos

```
$ cvs update
cvs update: Updating .
RCS file: /u/src/master/httpc/httpc.c,v
retrieving revision 1.8
retrieving revision 1.9
Merging differences between 1.8 and 1.9 into httpc.c
rcsmerge: warning: conflicts during merge
cvs update: conflicts found in httpc.c
C httpc.c
```


Manejando Conflictos (II)

```
  host_info = gethostbyname (hostname);
<<<<<<< httpc.c
  if (! host_info) {
      fprintf (stderr, "%s: host not found: %s\n", progname, hostname);
      exit (1);
    }
=======
  if (! host_info) {
```

Ramas

- Todo lo que hemos visto hasta aqui se relaciona con `HEAD`, la rama principal.

Ramas

- Todo lo que hemos visto hasta aqui se relaciona con `HEAD`, la rama principal.
- A veces nos interesan poder cambiar cosas en una version ya creada, por ejemplo un bugfix, y que no quieres que tambien tomen los cambios de la version actual.

cvsup

- Forma de ver cambios relacionados como uno solo cambio.

cvsup

- Forma de ver cambios relacionados como uno solo cambio.
- Ejemplo: cambiar el numero de argumentos de una funcion.

cvsup

- Forma de ver cambios relacionados como uno solo cambio.
- Ejemplo: cambiar el numero de argumentos de una funcion.
- Al usuario le interesa ver como un todo todos los cambios a ficheros individuales.

BitKeeper, Subversion, Arch, perforce

Solucionan algunos de los problemas de CVS:

- commits atomicos.

BitKeeper, Subversion, Arch, perforce

Solucionan algunos de los problemas de CVS:

- commits atomicos.
- identificador de fichero distinto de path de fichero.

BitKeeper, Subversion, Arch, perforce

Solucionan algunos de los problemas de CVS:

- commits atomicos.
- identificador de fichero distinto de path de fichero.
- permiten trabajo desconectado.

BitKeeper, Subversion, Arch, perforce

Solucionan algunos de los problemas de CVS:

- commits atomicos.
- identificador de fichero distinto de path de fichero.
- permiten trabajo desconectado.
- Tienen ramas que funcionan.

BitKeeper, Subversion, Arch, perforce

Solucionan algunos de los problemas de CVS:

- commits atomicos.
- identificador de fichero distinto de path de fichero.
- permiten trabajo desconectado.
- Tienen ramas que funcionan.

Más información

http://www.cvshome.org