CODA COMO

Iñaki Fernández Villanueva (autor y maquetador Linuxdoc-SGML), *iniaki@mail.com*
Javier Ruiz González, *javierrg77@yahoo.es*
Josu Abajo Marón, *jabajo00@yahoo.es*
v1.0, Mayo 2000

El objetivo de este documento es mostrar las características básicas del Sistema de Ficheros Distribuido Coda. También trata su instalación y configuración en un PC con Linux (Debian y Red Hat). No es una traducción de *The Coda Howto* (*http://coda.cs.cmu.edu/doc/html/coda-howto.html*), sino de un COMO redactado por sus autores que también incluye traducciones de otros documentos citados en la bibliografía.

Contents

1 Copyright y propiedad intelectual

2 Renuncia de responsabilidad

Este documento intenta ser una introducción a la instalación y configuración de Coda así como de su funcionamiento. Los autores de este documento NO SE HACEN RESPONSABLES DE NINGÚN DAÑO PRODUCIDO POR ACCIONES CON BASE EN ESTE DOCUMENTO, el cual puede contener erratas y/o fallos.

3 Dónde obtener este COMO

Puede obtener la última versión de este documento en el sitio web del proyecto Insflug *http://www.insflug.org*

4 Introducción

Un sistema de ficheros distribuido almacena ficheros en uno o más ordenadores sincronizados entre sí llamados servidores, y los hace accesibles a otros ordenadores llamados clientes, para quienes el acceso a estos ficheros es transparente. La principal ventaja es la compartición de ficheros y su gestión centralizada desde los servidores (como por ejemplo el control de acceso y la gestión de copias de seguridad). Esta compartición de ficheros es especialmente útil para grupos de trabajo que comparten documentos, aunque también es posible compartir software, como por ejemplo, un procesador de textos.

Un buen sistema de ficheros distribuido debe tener en cuenta cosas tan importantes como la latencia de la red, los posibles cuellos de botella y sobresaturación del servidor, la seguridad de datos comprometidos y los posibles fallos de red y servidores. Evidentemente todo esto toma especial importancia en el caso de que el sistema funcione bajo una *WAN*.

El Sistema de Ficheros Distribuido Coda es el sucesor de *Andrew File System* (**AFS**) y es un desarrollo de la Universidad de Carnegie-Mellon como ejemplo de entorno de trabajo distribuido. Coda destaca sobre AFS por permitir la Computación Móvil (trabajar en modo desconectado), soportar mejor la tolerancia a fallos del sistema (por ejemplo caída de los servidores o fallos de la red) y por disponer de técnicas de replicación de los servidores. Al ser gratuito, su código fuente está disponible (la licencia de Coda se puede encontrar en *ftp://ftp.coda.cs.cmu.edu/pub/coda/LICENSE*) y está diseñado para trabajar tanto en *LAN* como en *WAN*.

Coda se implementó originalmente en **Mach 2.6** y ha sido portado recientemente a **Linux**, **NetBSD** y **FreeBSD**. También se está portando a Windows95/NT, pero el estado de desarrollo es menor.

La Computación Móvil [MAR99] permite por ejemplo que un usuario pueda trabajar con su portátil enganchado a la red Coda, llevárselo a su casa para trabajar con él, y cuando vuelva a conectarse a la red los datos se reintegrarán automáticamente, sin que el usuario se percate de ello (entendiendo por reintegrar el proceso en el que tras una reconexión se ponen al día en los servidores los cambios realizados por el cliente durante la desconexión).

4.1 El cliente Coda

[BRA98] Bajo el directorio `/coda` el cliente monta un sistema de ficheros de tipo «Coda», desde donde se accederán a todos los ficheros del Sistema Coda. Un cliente se conecta a todo el sistema Coda y no a un servidor individual como ocurre en NFS, donde existe un único directorio o punto de montaje por servidor. La ventaja de un sólo punto de montaje reside en que todos los clientes pueden ser configurados de forma idéntica, y en que los usuarios siempre verán el mismo árbol de ficheros. Con NFS los clientes necesitan actualizar la lista de servidores y la ruta de directorios que exportan en `/etc/fstab`, mientras que en Coda los clientes sólo deben acceder al directorio `/coda` para ver los cambios incluso cuando se añaden nuevos servidores. En las dos siguientes figuras se aprecia la diferencia funcional entre un sistema NFS y un sistema Coda [MAR99]:

```
Sistema de Ficheros Distribuido, entorno Cliente-Servidor NFS:


                  RED
+---------+     +-----+
|         |     |     |
|Servidor |<----|-----|------->Cliente 1
|  NFS 1  |<----|-----|--+
|         |     |     |  |
+---------+     |     |  +---->Cliente 2
                |     |  |
```

```
                              |     |  |
              +---------+     |     |  |
              |         |     |     |  |
              |Servidor |<----|-----|--+
              |  NFS 2  |<----|-----|------->Cliente 3
              |         |     |     |
              +---------+     +-----+

Sistema de Ficheros Distribuido, entorno Coda:

              Sistema Coda
            +-------------+       RED
            | +---------+ |     +-----+
            | |         | |     |     |
            | |Servidor | |<----|-----|------->Cliente 1
            | | Coda 1  | |     |     |
            | |         | |     |     |
            | +----+----+ |<----|-----|------->Cliente 2
            |      |      |     |     |
            |      |      |     |     |
            | +----+----+ |     |     |
            | |         | |     |     |
            | |Servidor | |     |     |
            | | Coda 2  | |<----|-----|------->Cliente 3
            | |         | |     |     |
            | +---------+ |     +-----+
            +-------------+
```

La implementación de Coda en Linux está constituida por un conjunto de demonios que se ejecutan en los servidores, normalmente llamados *Vice*, y un demonio (***Venus***) más un módulo del *Núcleo* en la parte del cliente. La comunicación se establece entre los demonios, siendo el módulo del *núcleo* la interfaz entre el Sistema Coda y el **VFS** (*Virtual File System*) de Linux.

Cuando un usuario intenta leer un fichero del Sistema Coda (por ejemplo con `cat /coda/users/user15/doc.txt`) el programa cat realizará unas llamadas al sistema relacionadas con el fichero, que se comunicarán con el VFS del *núcleo*. El VFS comprueba entonces que la petición se refiere a un fichero Coda, y encamina la petición al módulo del sistema de ficheros Coda en el *Núcleo* (*Coda FS*). Este módulo mantiene una caché de peticiones recientes resueltas, y si la respuesta no se encuentra en esta caché, se encamina de nuevo la petición al gestor de caché Coda Venus (a través del dispositivo de caracter `/dev/cfs0`). Venus comprobará si el fichero `user15/doc.txt` se encuentra en una segunda caché, almacenada en disco, y en caso contrario contactará con los servidores (`Vice`) para obtenerlo. Una vez conseguido el fichero, Venus responderá al *núcleo*, quien devolverá la respuesta al programa `cat`.

Cuando el *núcleo* solicita a Venus la apertura de un fichero por primera vez, éste obtiene el fichero completo de los servidores utilizando las llamadas a procedimientos remotos (RPC) para comunicarse con los ellos. Después almacena el fichero en el área de caché (en el directorio `/usr/coda/venus.cache/`), desde donde será controlado por el sistema de ficheros `ext2` de Linux. Si un fichero ya se encuentra en la caché del cliente, Venus permitirá trabajar con él sin contactar con los servidores, de modo que si el fichero se abre una segunda vez, no se obtendrá del almacén remoto (trabajo en modo desconectado) sino de la caché.

Así pues, Venus sólo almacena aquella información que necesita el cliente, como ficheros o directorios (los directorios en Linux son ficheros) y sus atributos (propietario, permisos y tamaño). Si el fichero ha sido modificado y cerrado, entonces Venus actualiza los servidores enviándoles el nuevo fichero. Cualquier otra operación que modifique el sistema de ficheros (como la creación o eliminación de directorios y enlaces -*links*-) se propagará también a los servidores.

Coda ofrece distintos niveles de seguridad mediante **Kerberos**: no cifrar; cifrar sólo los paquetes de protocolo interno; cifrar además las cabeceras de los mensajes; y cifrado completo.

4.2 Las operaciones desconectadas

Si el cliente está desconectado e intenta actualizar un fichero que tiene en la caché, Venus se da cuenta que los servidores no están disponibles, se declara en modo desconectado y registra los cambios realizados en el **CML** (*Client Modification Log* o Registro de Modificación del Cliente) sobre el sistema de ficheros para actualizarlos en los servidores durante la siguiente conexión. Este proceso es transparente para el usuario, quien no se percata que ha cambiado a modo desconectado. Asimismo el CML está optimizado (por ejemplo, si un fichero es creado y luego borrado, se eliminan estos pasos del CML).

En ocasiones un cliente intenta acceder a un fichero que no tiene en su caché. Si está conectado lo consigue de los servidores, pero si no lo está, no hay nada que hacer y se devuelve un error al programa que haya hecho la petición. Para evitarlo existen los **ficheros HOARD**, que son un conjunto de ficheros relativamente importantes establecidos por el usuario que se mantienen en la caché. El usuario define la base de datos de ficheros HOARD, y puede solicitar a los servidores las últimas actualizaciones antes de desconectarse. Esta base de datos la puede construir automáticamente el sistema haciendo un seguimiento de los accesos que hace el usuario. Los ficheros Hoard permiten, por ejemplo, que un cliente forzar la carga del caché local antes de entrar en modo desconectado, y tener la garantía de que todo lo que necesita estará en su portátil tras la desconexión.

Puede ocurrir que dos o más clientes hayan actualizado el mismo fichero cuando estaban en modo desconectado. Cuando los clientes se conecten se producirá un **conflicto LOCAL/GLOBAL** entre cliente y servidor y se debe decidir por una de las actualizaciones. Para «reparar» este conflicto, el usuario dispone de la orden `repair`. La reparación la puede realizar a veces automáticamente «reparadores» específicos de la aplicación (por ejemplo, si dos usuarios modifican registros distintos de una misma base de datos, la propia base de datos los actualizaría sin que existiera un posible conflicto).

4.3 Volúmenes, servidores y replicaciones.

Los servidores Coda no almacenan los ficheros en un sistema de ficheros tradicional. En lugar de disco, partición o directorio, se utiliza el concepto de ***volumen***. Físicamente [MAR99] representa un grupo de ficheros mapeados en memoria por el demonio servidor `codasrv`, que contienen la información almacenada en dicho volumen. Los volúmenes proporcionan mayor flexibilidad al administrador, y su tamaño medio aproximado es de 10 MB, llegando a existir cientos de volúmenes por servidor.

RVM (*Recoverable Virtual Memory* o Memoria Virtual Persistente) registra la información de volúmenes y directorios, listas de control de accesos y atributos de los ficheros en particiones «crudas»[1] . En caso de una caída del host repara el sistema accediendo a la información de estas particiones, consiguiendo velocidad y consistencia. Hay dos particiones crudas: *Log Partition* y *Data Partition*.

Existe un volumen raíz que se monta bajo `/coda`, desde donde se montan el resto de los volúmenes. Obviamente este volumen es propiedad del administrador Coda. Un volumen tiene un nombre y un identificador *Id*, y se monta en cualquier subdirectorio de `/coda` (por ejemplo el volumen de un usuario `users.user15` se puede montar bajo `/coda/users/user15`). Cada fichero se identifica con un identificador `Fid` único en un sistema Coda y está compuesto por tres enteros de 32 bits:

VolumeId:

identifica el volumen en el que reside el fichero.

[1] *raw* en inglés, aquellas que tienen escrituras síncronas

VnodeId:

número de inodo del fichero.

Uniquifier:

identificador necesario para la resolución de conflictos.

Un volumen replicado es aquél que está almacenado en un grupo de servidores que pertenecen al mismo **VSG** (*Volume Storage Group*), de modo que cualquier operación sobre los ficheros de ese volumen afectará a todo el VSG al que pertenece (lo cual no supone mucho coste, ya que Coda implementa difusión[2]).El objetivo de esto es la alta disponibilidad del volumen. Asimismo existe el subgrupo **AVSG** (*Available VSG*), que son aquellos servidores accesibles y pertenecientes a un mismo VSG (puede ocurrir por ejemplo que uno de los servidores del VSG se averíe, con lo cual deja de ser accesible y deja de pertenecer al AVSG). Otros tipos de volúmenes son los locales (no replicados) y volúmenes *backup*. Los volúmenes *backup* permiten realizar copias de seguridad del Sistema de Ficheros Coda; sin embargo no se tratará en este documento.

La replicación de servidores puede provocar **conflictos globales** cuando el número de servidores que forman parte de un mismo AVSG es inferior al VSG (por ejemplo si las máquinas de un VSG son separados de los demás por una caída de la red). En este caso las actualizaciones de los ficheros no pueden propagarse a todos los miembros del VSG y es necesario resolver el conflicto con la orden `repair` (muchas veces sólo hay que decirle a Coda que lo repare como cuando hay que sustituir un disco duro y el sistema se encarga de actualizarlo).

4.4 Aplicaciones de Coda

En Internet, las réplicas de **servidores FTP** podrían ser clientes Coda que se actualizarían cada vez que los servidores Coda sufrieran cualquier modificación. Lo mismo pasaría con la réplica de **servidores WWW**, los cuales también pueden ser clientes Coda (los cuales pueden ser optimizados guardando en su caché local todos los datos a replicar). En ambos casos NFS es inadecuado ya que está diseñado para un entorno *LAN*, y hasta la aparición de Coda eran necesarias herramientas de sincronización entre servidores como `rsync`, que periódicamente contrastan las diferencias entre nodos y actualizan las diferencias. La computación móvil de Coda también puede ser aprovechada para aquellos clientes de proveedores de Internet que actualizan su página Web tras diseñarla en modo desconectado.

En las redes locales un usuario podría, por ejemplo, conectarse al sistema Coda cargando en su caché local los datos que vaya a utilizar ese día (promoviendo el acceso a servicios locales frente a remotos, lo cual incrementa el rendimiento disminuyendo los costes de comunicación), desconectarse[3] y, cuando acabe el día, volver a conectarse para reintegrar los cambios efectuados.

4.5 Desventajas

[MAR99] Debido a sus características Coda tiene una serie de desventajas (algunas ya mencionadas):

- Las operaciones de bloqueo de ficheros no están implementadas debido a que no es posible un algoritmo de bloqueo que tenga en cuenta un funcionamiento en modo desconectado.
- Existe un problema de sincronización intrínseco al modo desconectado: cuando al reconectar un cliente, un fichero ha cambiado tanto en el cliente como en el servidor, ¿cúal es la versión que se debe sincronizar con el resto del sistema?. Existen diversos algoritmos, pero frecuentemente se requiere la mano del operador.

[2] *multicast* en inglés

[3] Sin desconectarse también se consigue aumentar el rendimiento, ya que siempre es más eficiente trabajar sobre una caché local que sobre un fichero remoto (tal y como ocurre en NFS)

- La implementación de cuotas es limitada y sólo existe para los directorios (no existen cuotas para usuarios). Para solucionarlo se puede asignar un volumen por usuario, pero cambiar la cuota a un usuario es complicado porque los volúmenes Coda no son redimensionables.
- Coda no es estable y actualmente no se soportan bien volúmenes de más de 100 usuarios, ni mezcla de servidores Coda que no estén replicados (cada servidor Coda sirviendo un volumen independiente).
- Todas las operaciones de administración deben hacerse desde un cliente Coda sin que se pueda trabajar directamente con los volúmenes. Esto dificulta enormemente las tareas de mantenimiento y administración.
- Una máquina no puede ser a la vez cliente y servidor Coda.

5 Instalación y configuración

5.1 Dónde obtener los binarios de Coda

Todos los servidores deben tener la misma versión Coda para evitar problemas. La versión de los clientes puede ser anterior a la de los servidores pero mayor que una dada (dependiendo de la versión del servidor, aunque es conveniente que todas las versiones coincidan).

También es aconsejable instalar Coda a partir de los paquetes binarios para evitar compilar el código fuente. Existen binarios para las dos distribuciones Linux más utilizadas, Debian *http://www.debian.org* y Red Hat *http://www.redhat.com*. El código fuente se puede obtener junto a los binarios de Red Hat en *ftp://ftp.coda.cs.cmu.edu/pub/coda*, y los binarios de Debian de *ftp://ftp.debian.org/debian/project/experimental*. Estos paquetes binarios tienen unas dependencias o requisitos y deben ser compatibles con la versión de Linux en la que queremos instalar Coda. Por ejemplo en Debian se puede conocer las dependencias de un paquete binario con

```
dpkg --info nombrePaquete.deb
```

, y si nuestro sistema Linux lo cumple lo podremos instalar sin problemas.

En este informe se ha utilizado la versión 5.2.0-1 de Coda bajo Debian 2.1 *slink*. Existe en binario la versión 5.3.1-1 para Debian 2.2, pero hemos optado por la primera para evitar actualizar Linux. Aún así suponemos que entre ambas versiones no existen cambios importantes en la instalación y administración.

También hemos trabajado con la versión Coda de Red Hat, aunque recomendamos la de Debian por ser más fácil su instalación y administración. Por ejemplo, en Debian el programa de instalación pasa automáticamente a su configuración y el servidor Coda se lanza con el script `/etc/init.d/coda-server`, mientras que en Red Hat son varios los scripts.

5.2 Sincronización de los servidores

Los servidores deben estar sincronizados en fecha y hora. Para lograrlo hemos optado por `XNTP`, basado en sincronización externa `UTC` (Tiempo Universal Coordinado). En el caso de dos servidores Coda hay que añadir las siguientes líneas en sus ficheros de configuración `ntp.conf` del programa `xntp` (en Debian este fichero se encuentra en el directorio `/etc/`):

Servidor Coda 1:

```
server máquinaServidorXntp1
server máquinaServidorXntp2
peer  servidorCoda2
```

Servidor Coda 2:

```
server máquinaServidorXntp1
server máquinaServidorXntp2
peer servidorCoda1
```

Las dos primeras líneas de cada configuración especifican con qué servidor o servidores xntp se sincronizan. Si por redundancia se sincronizan con más de un servidor xntp como en el ejemplo, estos servidores deben tener el mismo nivel xntp garantizando la sincronización entre sí. La tercera línea

```
peer servidorCodax
```

asegura la sincronización entre los dos servidores Coda en el caso de que se pierda la comunicación con los servidores xntp (es importante mantener bien sincronizados los relojes locales del sistema distribuido).

5.3 Instalación de los Servidores

De todos los servidores Coda sólo uno puede ser el servidor `SCM`, desde donde se administra el sistema de volúmenes y las cuentas de usuario.

Se explicarán los procesos de instalación y configuración de las versiones 5.2.0-1 para Red Hat y Debian. Nótese que para la instalación y configuración de los servidores es necesario ser el superusuario del sistema Linux.

5.3.1 Servidor SCM en Red Hat

Se procederá a instalar el paquete, para lo que introduciremos por la línea de órdenes:

```
# rpm -i coda-debug-server-5.2.0-1.i386.rpm
```

Una vez instalado el servidor, iniciaremos su configuración, la cual difiere entre el servidor maestro SCM y el resto de servidores.

```
# vice-setup
```

A continuación se detalla el proceso de configuración con `vice-setup`. Introducimos una cadena de 8 caracteres (debe ser exactamente de 8 caracteres para evitar problemas causados por un posible *bug* de Coda). Un ejemplo puede ser `elephant`:

```
Setting up config files (under /vice).
Directories under /vice are set up.

Setting up tokens for authentication.
The following tokens must be identical on all servers.
Enter a random token for auth2 authentication : elephant
```

Introducimos, al igual que antes, una cadena de exactamente 8 caracteres. Debe ser distinta a la anterior (otro *bug*), por ejemplo, `elephann`:

```
The following tokens must be identical on all servers.
Enter a random token for volutil authentication : elephann
```

A partir de aquí empiezan las diferencias entre la configuración del servidor maestro y del resto de servidores. Contestar «y» si se trata del maestro y «n» en el caso de un servidor normal. Continuaremos como si hubiéramos contestado «y» a esta pregunta , y posteriormente comentaremos las diferencias que habría si se tratase de un servidor no SCM:

```
tokens done!

Setting up the file list for update
filelist for update ready.

Populating /vice/vol...
lockfiles created.
/etc/services ready for Coda
Is this the master server, aka the SCM machine? (y/n) y
```

Introduciremos un identificador para el servidor, por ejemplo el '1'. Esta pregunta se hará solamente al configurar el SCM, por lo que el resto de servidores habrá que añadirlos directamente en el fichero `/vice/db/servers` del SCM:

```
Now installing files specific to the SCM...

Setting up servers file.
Enter an id for this server (a number > 0 < 255) : 1
```

Aquí se establece el VSG (*Volume Storage Group*) del servidor que se está configurando, asignándosele un identificador (el valor por defecto es el `E0000100`). Toda la información correspondiente a los grupos de servidores se guarda en el fichero `/vice/db/VSGDB`. Si se quiere añadir un nuevo servidor a un grupo de servidores, o se quiere incluir un nuevo grupo, habrá que hacerlo editando directamente este archivo del SCM:

```
done!
Initializing the VSGDB to contain the SCM as E0000100
/vice/db/VSGDB set up
```

Se pide el nombre del *rootvolume*, que es el volumen que se montará como raíz en los clientes. Un posible valor es `coda:root`.

```
Setting up ROOTVOLUME file
Enter the name of the rootvolume (< 32 chars) : coda:root
```

En este paso se debe introducir un identificador de usuario que será el administrador del sistema. Este identificador deberá ser obligatoriamente `500`.

```
Setting up users and groups for Coda

You need to give me a uid (not 0) and username (not root)
for a Coda System:Administrator member on this server,
(sort of a Coda super user)

Enter the uid of this user:500
```

Ahora hay que darle un nombre a la cuenta del administrador Coda (el nombre con que trabajaremos será `admin`).

```
Enter the username of this user: admin
```

Se ha creado el nuevo usuario (nombre de usuario `admin` e identificador `500`), que tendrá su contraseña inicializada a `changeme`. Si se quiere cambiar habrá que utilizar o bien `cpasswd` o la utilidad de administración de usuarios `au`.

```
An initial administrative user admin (id 500)
with Coda password changeme now exists.
```

Aquí se pregunta si se quieren configurar las particiones RVM del servidor. Durante la instalación de cada servidor Coda es aconsejable dedicar 2 particiones tipo ext2 por razones de eficiencia (en otro caso se tratará como ficheros). Ambos ficheros forman la RVM usada para lograr una persistencia de la memoria virtual del host en caso de una caída:

```
You need a small log disk partition, preferrably on a disk by itself.
You need a metadata partition of approx 4% of you filespace.

For trial purposes you may give oridnary files instead of raw
partitions. Keep all size small if you do this.
Production servers will want partitions for speed.

-------------------------------------------------------
WARNING: you are going to play with your partitions now.
verify all answers you give.
-------------------------------------------------------

WARNING: these choices are not easy to change once you are up and
running.
Are you ready to set up RVM? [yes/no] yes
```

Hay que indicar cual es la partición o fichero que se va a utilizar como *log* (por ejemplo `/dev/hda4` para una partición o `/codap/logpartition` en el caso de un fichero):

```
What is your log partition? /dev/hda4
```

En la partición de *log* (*log partition*) se registran las transacciones de volúmenes coda que quizás no hayan sido aún actualizadas en la partición de datos coda. No debe sobrepasar los 30MB (el sistema coda no ha sido puesto a prueba con tamaños mayores a éste) y es aconsejable tener una partición de 2MB (lo merjo es ceñirse a lo que se indica en la instalación). Nunca se debe tener una partición menor que el tamaño indicado en la instalación (por ejemplo si la partición es de 11.6MB y en la instalación se indica que tiene 12MB, puede fallar la instalación):

```
The log size must be smaller than you log partition.
We recommend not more than 30M log size, and 2M is a good choice.
What is your log size? (enter as e.g. '12M') 2M
```

Elección de la partición de datos RVM (*data partition*). Indicar una partición (`/dev/hdxx`) o el nombre de un archivo:

```
What is your data partition (or file)? /dev/hda5
```

Se indicará el tamaño de la partición de datos. Recomendamos encarecidamente que se utilice uno de los valores dados por el *script*, ya que únicamente así se hará una configuración válida de las particiones (si se opta por poner otro tamaño de partición se deberá inicializar la partición de datos mediante el programa `rdsinit`. Este programa es difícil de manejar, siendo aconsejable poner el tamaño de la partición que viene por defecto en la instalación). Si se pone una partición menor que 22MB puede fallar la instalación. El script es muy sensible a las mayúsculas y minúsculas, por lo que es importante poner `22M` y no `22m` al indicar el tamaño de la partición:

```
The data size must be approx 4% of you server file space.
We have templates for servers of approx: 500M, 1G, 2.2G, 3.3G,8G
(you can store less, but not more on such servers).
The corresponding data sizes are  22M, 44M, 90M, 130M, 315M.
(use 330M only on very fast machines)
Pick one of the defaults, otherwise I will bail out

What is the size of you data partition (or file) [22M,44M, 90M,130M, 315M]:
22M
```

Aquí se pide confirmación para inicializar las particiones. Si se ha configurado bien saldrá lo siguiente:

```
-------------------------------------------------------
WARNING: DATA and LOG partitions are about to be wiped.
-------------------------------------------------------

  --- log area: /dev/hda4, size 2M.
  --- data area: /dev/hda5, size 22M.

Proceed, and wipe out old data? [y/n]  y
```

Se pregunta por el nombre del directorio en el que se guardarán los datos de los volúmenes coda. Pulse «intro» para que el directorio por defecto sea `/vicepa`:

```
LOG file has been initialized!

Rdsinit will initialize data and log.
This takes a while.

Your server directories will hold the files (not directories).
You can currently only have one directory per disk partition.

Where shall we store your data [/vicepa]?
```

Pregunta por el número aproximado de entradas de fichero que se van a tener:

```
Shall I set up a vicetab entry for approx 256000 files for y (y/n)  y
```

Una vez termine la ejecución de `vice-setup` habrá que levantar los servicios Coda ejecutando los correspondientes demonios:

```
Read vicetab(5) and makeftree(8) for set up info.
Server directory is set up!

Congratulations: your configuration is ready...and now
```

```
to get going do the following:
 - start the auth2 server as: auth2
 - start rpc2portmap as: rpc2portmap
 - start updateclnt as:  updateclnt -h ha0 -q coda_udpsrv
 - start updatesrv as: updatesrv -p /vice/db
 - start the fileserver: startserver &
 - wait until the server is up: tail -f /vice/srv/SrvLog
 - create your root volume: createvol_rep coda:root E0000100 /vicepa
 - setup a client: venus-setup ha0 20000
 - start venus: venus
 - enjoy Coda.
```

Los demonios están disponibles en el directorio `/etc/rc.d/init.d/`, y son los siguientes:

```
update.init
auth2.init,
codasrv.init
```

Para lanzarlos bastará con pasarles como parámetro la cláusula «start»:

- `/etc/rc.d/init.d/update.init start`, ejecuta `rpc2portmap`, `updateclnt` y `updatesrv`. Este script no ejecuta bien el demonio `updatesrv`, hay que hacer una pequeña modificación añadiendo "> /dev/null" al final de la línea en la que se le ejecuta.
- `/etc/rc.d/init.d/auth2.init start`, ejecuta `auth2`.
- `/etc/rc.d/init.d/codasrv.init start`, ejecuta `startserver`, quien a su vez ejecuta el `codasrv`.

Se puede comprobar que el servidor ha empezado a funcionar mirando el siguiente *log*:

```
$ tail -f /vice/srv/SrvLog &
```

El siguiente paso es instalar el resto de los servidores no-SCM y configurarlos.

5.3.2 Servidor no SCM en Red Hat

La configuración de un servidor no SCM es prácticamente idéntica a la del servidor maestro, aunque se omiten una serie de pasos (los de la creación de usuarios y especificación del volumen root).

Es importante indicar que por razones de seguridad todos los servidores de un mismo VSG deberán tener los mismos *tokens* de autentificación.

En esta ocasión no se pedirá un número de identificación del servidor (tenemos que introducirlo antes o después en `/vice/db/servers` de la máquina SCM), pero sí se nos preguntará la ruta a la máquina maestra (dirección IP ó nombre de la máquina).

Tenemos que configurar, de igual manera, los RVM's (particiones log y de datos). Tras terminar con la configuración lanzaremos los demonios del servidor no-SCM:

- `/etc/rc.d/init.d/update.init`
- `/etc/rc.d/init.d/auth2.init`
- `/etc/rc.d/init.d/codasrv.init`

5.3.3 Debian

La instalación en Debian es bastante similar a la de Red Hat, por lo que solamente contaremos las diferencias entre una y otra. Comenzaremos por instalar el paquete binario, para ello teclearemos:

```
# dpkg -i coda-server_5.2.0-1.i386.deb
```

La propia instalación del paquete lanza el programa de configuración `vice-setup`, que es idéntico al de la distribución Red Hat (la única diferencia es que éste lanza los demonios directamente).

Una vez hecha la configuración (siguiendo los mismos pasos que hemos descrito en el apartado anterior para Red Hat) estarán lanzados todos los demonios a excepción de `startserver` y `codasrv` (los cuales habrá que lanzar a mano con `startserver &`).

Para que estos dos demonios se ejecuten automáticamente tenemos que crear el siguiente fichero `/vice/srv/STARTFROMBOOT`. Una forma de hacerlo es con:

```
# echo >/vice/srv/STARTFROMBOOT
```

o

```
# touch /vice/srv/STARTFROMBOOT
```

En Debian existe un único script encargado de lanzar o matar todos los demonios de un servidor Coda (nótese que la ruta del script cambia con respecto a la ruta de Red Hat):

`/etc/init.d/coda-server stop`
: para parar el servidor Coda.

`/etc/init.d/coda-server start`
: para iniciarlo.

Tras todo esto ya sólo nos queda configurar los servidores desde el SCM para que Coda empiece a funcionar correctamente.

5.4 Configuración de los servidores Coda

Los volúmenes son unidades de información que permiten gestionar conjuntamente los datos que contienen. Es posible que un volumen pertenezca sólo a un servidor (**volumen no replicado**) y que sólo unos pocos usuarios puedan leer y escribir sobre él. También es posible que un volumen pertenezca a más de un servidor (**volumen replicado**) y que todos los usuarios coda puedan leer y escribir sobre él.

5.4.1 Creación de volúmenes

El primer paso para configurar los servidores es crear el volumen *root* desde el SCM (el único que puede crear y borrar volúmenes) con una de las siguientes órdenes:

```
# createvol_rep coda:root E0000100 /vicepa
```

```
# createvol coda:root sipt30 /vicepa
```

En ambos casos el volumen *root* se llama `coda:root`, pero con la diferencia que en el primero creamos el volumen replicado y en el segundo no (donde `sipt30` es el servidor Coda que contiene el volumen *root*).

En el ejemplo del volumen root replicado `E0000100` es el identificador del VSG al que pertenece el servidor SCM (por defecto el SCM pertenece a este VSG) y el resto de servidores Coda donde queremos replicar el volumen. Para añadir nuevos servidores a este VSG (al que inicialmente sólo pertenece el SCM) el administrador debe modificar los siguientes ficheros del SCM:

- `/vice/db/VSGDB`, para indicar a qué servidores corresponde el identificador de grupo E0000100. Por ejemplo:

  ```
  E0000100 sipt30 sipt31
  ```

- `/vice/db/servers`, para indicar cuáles son los identificadores de cada servidor (el del SCM ya ha sido escogido durante el proceso de instalación). En el siguiente ejemplo el identificador del servidor coda y SCM `sipt30` es `1` y el del servidor `sipt31` es `2`

  ```
  sipt30  1
  sipt31  2
  ```

Dichos ficheros habrá que modificarlos en el SCM cada vez que se quiera añadir o eliminar un nodo del grupo de servidores.

Cuando se conecten los servidores No-SCM, se actualizarán por la red sus ficheros del directorio `/vice` (los ficheros de configuración del servidor) incluyendo los dos anteriores.

Al igual que el *root volume*, el servidor SCM puede crear otros volúmenes necesarios con `createvol_rep` (volúmenes replicados) o con `createvol` (volúmenes locales al servidor sin replicar). De este modo podemos encontrarnos con la siguiente configuración del fichero `/vice/db/VSGDB`:

```
E0000100 ha0 ha1
E0000200 ha1
```

donde los servidores `ha0` y `ha1` pertenecen al grupo `E0000100` y sólo `ha1` pertenece al grupo `E0000200` (aunque el identificador pertenezca a un volumen replicado).

5.4.2 Eliminación de volúmenes

Para eliminar un volumen se utiliza la orden `purgevol` (para volúmenes no replicados) o `purgevol_rep` (para volumenes replicados), que sólo puede ser ejecutada desde el servidor SCM:

```
# purgevol_rep NombreVolumen
```

5.4.3 Ficheros de configuración del servidor

La información almacenada en los servidores Coda está organizada en varios directorios, que están descritos a continuación:

- `/vice/auth2` Este directorio contiene información relacionada con el proceso de autenticación, incluyendo sus ficheros *log*.

- `/vice/bin` Contiene los binarios del sistema de ficheros Coda para los servidores y el SCM.
- `/vice/db` Contiene las bases de datos con información importante para los servidores y los ficheros *log* de las actualizaciones.
- /vice/srv Contiene información relacionada con los demonios del servidor, incluyendo sus ficheros *log*.
- `/vice/vol` Contiene información de los volúmenes Coda.
- `/vice/vol/remote` Existe sólo en el SCM y contiene información de todos los servidores remotos.
- En `/vicepa` (si no se ha cambiado por otro directorio durante la instalación) se guardan los datos de los volúmenes Coda que tiene el servidor. La información se guarda en forma de ficheros que se distribuyen a lo largo de un árbol de directorios codificado numéricamente. Consideramos interesante reseñar que en nuestro caso los ficheros comenzaban a guardarse ordenadamente en el directorio `/vicepa/0/0/`, aunque desconocemos el sistema de codificación.

5.5 Instalación de los clientes Coda

El cliente debe instalar dos cosas: un módulo del núcleo para reconocer el sistema de ficheros Coda, y el propio programa cliente Coda. Se recuerda que un cliente y un servidor Coda no funcionan bajo una misma máquina, por lo que debemos evitar que ocurra.

5.5.1 Módulo del núcleo Coda

Para que el cliente tenga acceso al sistema de ficheros distribuido Coda es necesario que el Núcleo lo reconozca. Esto se puede conseguir de varias formas:

1. Compilando uno de los *núcleo*s 2.2.x que ya disponen de soporte Coda y habilitándolo dentro del Núcleo (parte del proceso de compilación de un *núcleo* en Linux).
2. Compilando uno de los *núcleos* 2.2.x para que soporte Coda, pero esta vez obteniendo un módulo `coda.o` que podremos cargar y descargar del *Núcleo* cuando queramos.
3. Si nuestro *Núcleo* tiene una versión anterior al 2.2.x, podemos obtener el módulo `coda.o` mencionado antes compilando el código fuente de Coda (normalmente se encuentra en el fichero `linux-coda-(versión).tgz`)

Si nos hemos decidido por trabajar con el módulo coda (opciones 1 (1) y 2 (2)), dicho módulo no se cargará en memoria una vez arrancado Linux. Para cargarlo en memoria existen varios métodos, por ejemplo:

- Mediante la línea de órdenes del *root* con:

  ```
  # insmod coda
  ```

- Cargando obligatoriamente el módulo cuando se arranque Linux (introduciendo en el fichero `/etc/modules` la línea `coda`).
- Cargando automáticamente el módulo sólo cuando sea necesario (introduciendo en el fichero `/etc/modules` la línea `auto`).

Nota: el módulo `coda.o` debe encontrarse en `/lib/modules/versionNúcleo/fs`, donde «versionNúcleo» es la versión del *núcleo* de Linux (para consultar la versión del *núcleo* desde la línea de órdenes probar con `uname -r`).

5.5.2 Instalación de los binarios

La instalación del cliente a partir de un paquete binario Linux se realiza de distinta forma dependiendo de la distribución a la que pertenece:

Red Hat El primer paso es instalar el paquete binario:

```
# rpm -i coda-debug-client-5.2.0-1.i386.rpm
```

Venus es el gestor de la caché del cliente. Para configurarlo tenemos el *script* `venus-setup`:

```
# /usr/bin/venus-setup <lista_de_hosts_separados_por_comas> <tamaño_de_caché_en_kb>
```

con lo que decimos a Venus cuál es su lista de servidores Coda a los que debe conectarse. También inicializa un directorio para utilizarlo como caché de disco del cliente Coda, con el tamaño indicado en el *script* `venus-setup` (para empezar se recomienda un a pequeña caché de 20 MB, aunque funciona bien hasta 300 MB). Asimismo `venus-setup` creará el dispositivo `/dev/cfs0` para comunicarse con el *Núcleo* y dejará todos los ficheros del cliente Coda en el directorio `/usr/coda`.

También sería recomendable probar nuestro cliente Coda con el servidor Coda *testserver.coda.cs.cmu.edu*, aunque deberá asegurarse que no tiene ningún *firewall* que le impida comunicarse con él. Una caché de 20000 es aconsejable para probar este servidor.

Tras la instalación del paquete binario se puede lanzar Venus en *background* con la orden:

```
# venus &
```

y se puede parar con

```
# kill -9 venus
```

o con

```
vutil shutdown
umount /coda
```

Aunque una manera más limpia de lanzar y parar Venus es desde su *script* de inicio `/etc/rc.d/init.d/venus`.

Nota: Antes de volver a lanzar Venus el directorio `/coda` debe ser desmontado. Si esto diera problemas asegúrese de matar todos los procesos que cuelgan de Coda, por ejemplo cuando tenemos ficheros de Coda abiertos por una aplicación o porque estamos dentro del directorio `/coda`. Las utilidades `lsof` y `fuser` pueden ayudarnos para solucionar estas cosas.

Debian Instalamos el binario del cliente Coda:

```
# dpkg -i coda-client_5.2.0-1_i386.deb
```

El proceso es similar al de Red Hat, aunque en Debian `venus-setup` se ejecuta en la propia instalación. Aún así siempre se puede utilizar `venus-setup` y `venus &` para una posterior configuración.

`/etc/init.d/coda-client` es el script que lanza y para el demonio `Venus`.

5.5.3 Desinstalación

A continuación se describe el proceso de desinstalación de un servidor Coda, útil cuando nos hemos equivocado en el proceso de instalación o configuración y queremos volver a empezar.

Servidor Red Hat Comenzaremos por parar a todos los demonios utilizando los lanzadores de los que disponemos en `etc/rc.d/init.d/`:

```
# /etc/rc.d/init.d/auth2.init stop
# /etc/rc.d/init.d/update.init stop
# /etc/rc.d/init.d/codasrv.init stop
```

Tras esto verificaremos que ninguno de los siguientes procesos esté cargado en memoria con `ps uax | less`:

```
auth2
rpc2portmap
update
updateclnt
updatesrv
startserver
codasrv
```

Si alguno de estos procesos está en ejecución lo podemos parar con:

```
kill -9 pid
```

donde `pid` es el identificador del proceso que aparece indicado al ejecutar `ps`.

Ahora ya nos hemos asegurado de que no hay ningún proceso del servidor coda en funcionamiento, por lo que podemos proceder a la desinstalación del paquete.

```
# rpm -e coda-debug-server-5.2.0-1
```

Por último, sólo nos queda borrar los directorios de Coda:

```
# rm -rf /vicepa
# rm -rf /vice
# rm -f /var/lock/subsys/auth2.init
# rm -f /var/lock/subsys/update.init
# rm -f /var/lock/subsys/codasrv.init
```

Nota: Se pueden producir fallos en la configuración de Coda si se intenta instalar de nuevo sin borrar antes los directorios `/vicepa` y `/vice`.

Servidor Debian Comenzaremos por dar de baja a todos los demonios:

```
# /etc/init.d/coda-server stop
```

El resto del proceso es idéntico al de Red Hat, salvo que el paquete binario de Coda de desinstala con:

```
# dpkg -r coda-debut-server_5.2.0-1
```

o con la herramienta `dselect` de Debian.

Cliente (Red Hat y Debian) El cliente es mucho más sencillo y es suficiente con desinstalar el paquete binario de la distribución (orden `rpm` en Red Hat y `dpkg` en Debian). Asimismo la desinstalación del módulo Coda del *núcleo* dependerá del proceso escogido para su instalación.

6 Administración

6.1 Creación de cuentas de usuario

[HAR98] Una vez instalados y configurados correctamente los servidores Coda debemos crear las **cuentas de los usuarios Coda**. Para ello emplearemos la orden interactiva `pdbtool`. Las órdenes más utilizadas en `pdbtool` son:

`nu nombreusuario`
: crea un nuevo usuario (el sistema le asigna un identificador).

`nui nombreusuario idusuario`
: crea un nuevo usuario con el identificador especificado.

`ng nombregrupo idpropietario`
: crea un nuevo grupo con el propietario especificado.

`ci usuario/nombregrupo nuevoid`
: cambia el identificador de un usuario o grupo existente.

`ag id-grupo usuario/idgrupo`
: añade un usuario o grupo a un grupo.

`n usuario/nombregrupo`
: lista toda la información del usuario o del grupo especificado.

donde los identificadores de usuario son enteros positivos y los identificadores de grupo son enteros negativos. Como ejemplo se creará una cuenta Coda con la herramienta `pdbtool`. Esta operación debemos realizarla sobre el servidor SCM, ya que es el único que puede realizarla.

```
root@scm# pdbtool
pdbtool> nu tux
pdbtool> n tux
USER tux
     *    id: 779
     *    belongs to no groups
     *    cps:  [  779  ]
     *    owns no groups
pdbtool> ng users 779
pdbtool> n users
GROUP users OWNED BY tux
     *    id: -205
     *    owner id:  779
     *    belongs to no groups
     *    cps:  [  -205  ]
     *    has members: [ 779 ]
pdbtool> n System:AnyUser
```

```
GROUP System:AnyUser OWNED BY System
     *    id: -101
     *    owner id:  777
     *    belongs to no groups
     *    cps:  [  -101  ]
     *    has members: [ 777 ]
pdbtool> ag -101 779
pdbtool> ag -205 779
pdbtool> n tux
USER tux
     *    id: 779
     *    belongs to groups:  [ -101   -205]
     *    cps:  [ -101   -205   779 ]
     *    owns: [ -205 ]
pdbtool> q
```

La anterior secuencia ha creado una nueva cuenta de usuario llamada `tux` y se ha hecho que forme parte del también creado grupo `users`. Igualmente se ha introducido esta nueva cuenta en el grupo `System:AnyUser`, el cual contiene a todas las cuentas del Sistema Coda. Para activar la cuenta es necesario asignarle una contraseña desde cualquier servidor, autenticándose antes como el administrador de Coda (durante la instalación del SCM se ha solicitado el nombre y la contraseña de la cuenta de administración, que en nuestro caso son `admin` y `changeme` respectivamente):

```
admin@cualquiermáquina$ au -h scm nu
Your Vice Name: admin
Your Vice Password: ********
New User Name: tux
New User Password: nuevaContraseña
```

A continuación creamos un *home volume* replicado llamado `users:tux` con VSG E0000100, lo montamos, le asignamos todos los permisos de usuario posibles sobre ese volumen (ver siguiente sección 3 (orden cfs), donde en el apartado 3 se explican los permisos de Coda), y lo desmontamos. Nótese que la orden `cfs mkm` crea y monta a la vez el directorio del volumen asociado.

```
root@scm# createvol_rep users:tux E0000100 /vicepa
admin@cualquiermáquina$ cfs mkm /coda/usr/tux users:tux
admin@cualquiermáquina$ cfs sa /coda/usr/tux tux all
admin@cualquiermáquina$ cfs rmm /coda/usr/tux
```

Finalmente el usuario `tux` podrá cambiar su contraseña desde cualquier máquina cliente Coda con:

```
tux@cualquiermáquina$ cpasswd -h scm
```

siendo SCM la máquina SCM del sistema Coda.

6.2 Acceso a las cuentas y órdenes

Si todo va bien el cliente debería ser capaz de montar el sistema de ficheros Coda bajo el directorio `/coda` (donde se monta el volumen *root*). Si existe el fichero `/coda/NOT_REALLY_CODA` entonces aún no se ha montado Coda y debemos comprobar que el demonio `Venus` está lanzado.

Para acceder a una cuenta Coda existe la orden `clog user`, donde *user* es el nombre de usuario o *login*. Desde cualquier máquina con cliente Coda:

```
$ clog tux
username: tux
password: ********
```

A partir de entonces el usuario autenticado puede montar los volúmenes a los que tiene acceso (nuestro usuario `tux` tiene acceso al volumen `users:tux` y lo monta bajo `/coda/usr/tux`):

```
tux@cualquiermáquina$ cfs mkm /coda/usr/tux users:tux
```

El usuario ya puede trabajar con el directorio `/coda/usr/tux` como si se tratara de uno tradicional. Después siempre podrá desmontar el volumen con la orden:

```
cfs rmm /coda/directoriomontaje
```

Nota: la versión Coda 5.2.0 tiene problemas si en `cfs mkm` se indica el *path* de montaje acabado en '/'. Al parecer se ha conseguido arreglar este *bug* en versiones posteriores. Asimismo hemos tenido problemas al intentar editar un fichero Coda con el editor `emacs` (lo hemos «solucionado» trabajando con `vi` en las pruebas).

6.2.1 Comando cfs

[SAT97-1] La orden `cfs` (*Coda File System Interface Program*) permite a los usuarios ejecutar operaciones específicas del Sistema de Ficheros Coda. A menudo se utiliza para ver el espacio de almacenamiento utilizado y para cambiar los permisos de protección de un directorio. A continuación se detallan los opciones de `cfs` más importantes:

1. `cfs mkmount <directorio> <nombre-volumen> [-rw]` Crea un directorio de montaje especificado en `<directorio>` y monta un volumen especificado en `<nombre-volumen>`. Si se utiliza el *flag -rw* el volumen se monta con permisos de lectura y escritura, ya que de otro modo los permisos serían de sólo lectura si su volumen padre también lo es. Nótese que en ambos casos el usuario debe tener los privilegios necesarios.

   Abreviatura: mkm

2. `cfs rmmount <dir> [<dir> <dir> ...]` Elimina uno o más directorios de montaje (especificados por <dir>) del sistema de ficheros. En sí mismo el volumen no cambia.

   Abreviatura: rmm

3. `cfs setacl [-clear] [-negative] <dir> <id> <perm> [<id> <perm> ....]` En Coda el concepto tradicional de permisos de usuario, grupo y otros desaparece. En su lugar CODA utiliza las denominadas Listas de Control de Acceso (ACL's), las cuales consisten en una serie de datos que definen qué usuarios o grupos pueden hacer qué cosas con cada elemento de un espacio de direccionamiento Coda. Estos permisos [MAR99] constituyen un modelo mucho más elaborado que los tradicionales permisos de ejecución/lectura/escritura de Unix. Los permisos no son establecidos para cada fichero, sino para todos los ficheros de un directorio (aunque en la documentación de la orden `cfs` se aconseja el uso de la orden Unix `chmod` para cambiar los permisos de los ficheros):

   ```
   r       Read, permiso de lectura.
   l       Lookup, permiso para obtener el status de un fichero.
   i       Insert, permiso de creación de ficheros o directorios.
   d       Delete, permiso de borrado.
   w       Write, permiso de modificación.
   a       Administer, permiso de control de los permiso de acceso.
   ```

Con la orden `cfs setacl` (`cfs sa` abreviado) se configura la ACL del directorio <dir> para cada usuario identificado por <id> con los permisos <perm> *rlidwa* explicados anteriormente (*read, lookup, insert, delete, write* y *administer*). El *flag -clear* borra toda la lista de control de accesos a excepción de lo especificado en la propia orden `cfs`. El *flag -negative* niega los permisos especificados en la orden en lugar de concederlos.

Abreviatura: sa

4. `cfs listacl <dir> [<dir> <dir> ...]` Muestra la lista de control de accesos para los directorios dados.

   Abreviatura: la

5. `cfs whereis <dir> [<dir> <dir> ...]` Lista los servidores donde residen los ficheros especificados.

6. `cfs disconnect` Desconecta el cliente Coda de los servidores Coda. Útil por ejemplo cuando queramos trabajar localmente desde nuestra caché Coda en modo desconectado y aumentar así el rendimiento en los tiempos de acceso.

7. `cfs reconnect` Reconecta el cliente a los servidores Coda, deshaciendo los efectos de `cfs disconnect.`

   Nota:Hasta la versión 5.2.2 de Coda esta orden tenía un *bug* conocido que impedía la reconexión (`cfs disconnect` pone un software de filtrado en los niveles de rpc2 pero `reconnect` falla al borrarlo). Para solucionarlo ejecutar desde el servidor SCM:

   ```
   # filcon clear -c hostcliente
   ```

   consiguiendo el rid del filtro sin arrancar Venus.

8. `cfs writedisconnect [-age <secs> -time <secs> <dir>]`Indica a Venus que va a escribir en modo desconectado en los volúmenes/directorios dados o en todos los volúmenes si no se especifica ninguno, para lo cual se cargará en la caché los ficheros correspondientes de los servidores (no propagará los cambios inmediatamente). El argumento *-age* especifica el tiempo de *caching* en la caché del cliente antes de reintegrar los datos. El argumento *-time* proporciona el número de segundos que debería tardar el envío de un fragmento de reintegración.

   Abreviatura : wd

9. `cfs writereconnect [<dir> <dir> ...]` Conexión estricta de los directorios Coda a los servidores.

   Abreviatura : wr

10. `cfs examineclosure` Examina el cierre de reintegración, mostrando la localización de los ficheros no reintegrados y modificados durante la desconexión.

    Abreviatura: ec

11. `cfs replayclosure` Repite el cierre de reintegración (útil por ejemplo si falta algún fichero con conflictos por reintegrar).

    Abreviatura: rc

12. `cfs listcache [<dir> <dir> ...]`Muestra los contenidos de la caché de los directorios/volúmenes dados (si no se especifican por defecto se muestra toda la caché).

    Abreviatura: lc

13. `cfs listvol <dir> [<dir> <dir> ...]` Muestra el estado actual del volumen en el que el directorio especificado se almacena.

    Abreviatura: lv

14. `cfs lsmount <dir> [<dir> <dir> ...]` Lista los contenidos de un directorio de montaje. Esta orden se puede utilizar para conocer a qué volumen se asocia un directorio de montaje dado.

Para más información consultar las `man` de la orden `cfs`.

6.2.2 Reparación de conflictos

[SAT97-2] Como se ha explicado en la introducción, sucede ocasionalmente que un directorio se vuelve inconsistente debido a un conflicto global, es decir, cuando Coda no puede resolver automáticamente una replicación entre servidores de un mismo VSG (por ejemplo cuando un mismo grupo de servidores VSG se particiona y un mismo fichero se modifica en más de una de las particiones). También es posible una inconsistencia local de algún cliente con respecto al estado global (conflicto local/global que se produce en los fallos de reintegración), normalmente porque un cliente desconectado actualiza un fichero que también ha sido actualizado en los servidores por otro cliente. Cuando ocurre algún conflicto de éstos, el directorio que contiene el conflicto se convertirá en un enlace simbólico que apunta a su `fid`. Por ejemplo, si el directorio `conflicto` es inconsistente aparecerá así:

```
$ ls -l conflicto
lr-r-r- 1 root 27 Mar 23 14:52 conflicto -> @@7f0000b3.00000005.0000011a
```

La mayoría de las aplicaciones devolverán un error de «`Fichero no encontrado`» cuando intenten abrir un fichero inconsistente. Para resolver este conflicto existe la herramienta `repair`.

Conflictos servidor-servidor Tras ejecutar `repair` desde un cliente Coda, es necesario hacer `begin` del objeto inconsistente, tras lo cual el directorio inconsistente tendrá una entrada por cada uno de los volúmenes replicados. Con una observación a estos subdirectorios el usuario podrá elegir qué copia quiere, y con la orden `repair` podrá copiar la versión correcta y eliminar la inconsistencia. En el siguiente ejemplo el fichero `conflicto/ejemplo.txt` está replicado en tres servidores y queremos resolver la inconsistencia entre servidores:

```
$ ls -lL conflicto
lr-r-r- 1 root 27 Dec 20 13:12 conflicto -> @@7f0002ec.000000e3.000005d1

$ repair
The repair tool can be used to manually repair files and directories
that have diverging replicas.  You will first need to do a "beginRepair"
which will expose the replicas of the inconsistent object as its
children.

If you are repairing a directory, you will probably use the "compareDir"
and "doRepair" commands.

For inconsistent files you will only need to use the "doRepair" command.
If you want to REMOVE an inconsistent object, use the "removeInc" command.

Help on individual commands can also be obtained using the "help"
facility.

repair> begin conflicto
a server-server-conflict repair session started
```

```
use the following commands to repair the conflict:
        comparedirs
        removeinc
        dorepair
repair> ^Z
Stopped
$ ls conflicto
gershwin.coda.cs.cmu.edu        schumann.coda.cs.cmu.edu
$ ls conflicto/*
conflicto/gershwin.coda.cs.cmu.edu:
ejemplo.txt

conflicto/schumann.coda.cs.cmu.edu:
ejemplo.txt
$ fg
repair
compare
Pathname of Object in conflict?  [conflicto]
Pathname of repair file produced?  []  /tmp/fix

NAME/NAME CONFLICT EXISTS FOR ejemplo.txt

-rw-r--r-- 1 tux 0 Dec 20 13:10 gershwin.coda.cs.cmu.edu/ejemplo.txt
-rw-r--r-- 1 -101 0 Dec 20 13:11 schumann.coda.cs.cmu.edu/ejemplo.txt


/coda/project/conflicto/gershwin.coda.cs.cmu.edu/ejemplo.txt
        Fid: (0xb0.612) VV:(0 2 0 0 0 0 0 0)(0x8002f23e.30c6e9aa)
/coda/project/conflicto/schumann.coda.cs.cmu.edu/ejemplo.txt
        Fid: (0x9e.5ea) VV:(2 0 0 0 0 0 0 0)(0x8002ce17.30d56fb9)

Should /coda/project/conflicto/gershwin.coda.cs.cmu.edu/ejemplo.txt be
removed?  [no] yes

Should /coda/project/conflicto/schumann.coda.cs.cmu.edu/ejemplo.txt be
removed?  [no]

Do you want to repair the name/name conflicts [yes]
Operations to resolve conflicts are in /tmp/fix
repair> do
Pathname of object in conflict?  [conflicto]
Pathname of fix file?  [/tmp/fix]
OK to repair "conflicto" by fixfile "/tmp/fix"?  [no]  yes
SCHUMANN.CODA.CS.CMU.EDU  succeeded
GERSHWIN.CODA.CS.CMU.EDU  succeeded
repair> quit
$ ls conflicto
ejemplo.txt
$ exit
```

Conflicto local-global El uso de la orden `repair` es similar al caso anterior. Después de empezar la sesión de reparación con `begin` e indicar el *path* del objeto en conflicto, las réplicas local y global serán visibles en `pathObjetoEnConflicto/local` (sólo lectura) y en `pathObjetoEnConflicto/global` (modificable). Con la orden `listlocal` se muestra la lista de todas las modificaciones locales sobre el objeto inconsistente o sobre sus descendientes, siendo necesario reparar de uno en uno y en el orden de la lista los posibles conflictos de estas modificaciones. `checklocal` nos dice si el primer elemento de la lista a tratar tiene algún conflicto o no. El siguiente algoritmo muestra el proceso principal de reparación:

```
listlocal (para visualizar la lista)
para cada elemento de la lista «listlocal» hacer
    checklocal(se refiere al primer elemento de la lista de modificaciones
               locales)
    si el elemento a tratar tiene conflicto:
        resolver conflicto
    sino
        decidir si queremos preservar la copia local en el servidor
    finsi
```

Con `discardlocal` se descarta la copia local preservando la del servidor, y con `preservelocal` la copia que se preserva es la local. Ambas opciones se pueden utilizar tanto si se trata de un conflicto como si no (ambos casos del *if* del algoritmo). Para acelerar el proceso existen las órdenes `preservealllocal` (preserva todos los elementos del objeto en conflicto) y `discardalllocal` (todos los elementos modificados localmente se desechan para preservar el estado global del servidor).

Se pueden utilizar herramientas o editores como `vi` para actualizar convenientemente la réplica global del objeto en conflicto, ya que como se ha dicho antes, la réplica global es la única modificable. La orden `quit` se utiliza para comprometer o abortar la sesión de reparación. Las páginas `man` ofrecen información más detallada acerca de estas órdenes de reparación. En el siguiente ejemplo se ilustra el proceso de reparación de un conflicto local/global, en el cual se supone que durante la desconexión un usuario crea un nuevo directorio `/coda/usr/tux/doc` y actualiza el fichero `/coda/usr/tux/fichero.txt`. Simultáneamente otro usuario con permisos también crea un directorio `/coda/usr/tux/doc` y almacena varios ficheros en él, produciéndose el conflicto local/global del objeto `/coda/usr/tux` durante la reintegración:

```
tux@clientecoda$ ls -l /coda/usr/tux/doc
lr--r--r-- 1 root 27 Dec 20 00:36 doc -> @@7f000279.00000df3.0001f027

tux@clientecoda$ repair
repair> begin
Pathname of object in conflict?  []  /coda/usr/tux
a local-global-conflict repair session started
the conflict is caused by a reintegration failure
use the following commands to repair the conflict:
        checklocal
        listlocal
        preservelocal
        preservealllocal
        discardlocal
        discardalllocal
        setglobalview
        setmixedview
        setlocalview
a list of local mutations is available in the .cml file in the coda
spool directory

repair> !ls -l /coda/usr/tux/
total 4
drwxr-xr-x  3 tux         2048 Dec 20 00:51 global
drwxr-xr-x  3 tux         2048 Dec 20 00:51 local
```

```
repair> listlocal
local mutations are:

Mkdir   /coda/usr/tux/local/doc
Store   /coda/usr/tux/local/fichero.txt (length = 19603)

repair> checklocal
local mutation: mkdir /coda/usr/tux/local/doc
conflict: target /coda/usr/tux/global/doc exist on servers

repair> discardlocal
discard local mutation mkdir /coda/usr/tux/local/doc

repair> checklocal
local mutation: store /coda/usr/tux/local/fichero.txt
no conflict

repair> preservelocal
store /coda/usr/tux/global/fichero.txt succeeded

repair> checklocal
all local mutations processed

repair> quit
commit the local/global repair session?  [yes]
reintegrate
```

6.2.3 Otras órdenes

Otras órdenes importantes a tener en cuenta son:

- cliente:
  - `hoard`: hoard database front-end. *Front-end* de la base de datos Hoard (HDB), con el cual es posible añadir un fichero a la base de datos Hoard, borrar un fichero, listar el contenido del HDB, o modificar los atributos de un fichero hoard.
  - `ctokens`: lista los tokens del usuario con la fecha de expiración, la identificación del usuario y si está o no autenticado.
- Servidor:
  - `filcon`: utilidad de control de filtrado RPC2. Por ejemplo se puede aislar un servidor con:

    ```
    filcon isolate -s nombre-servidor
    ```

    y deshacer esta operación con:

    ```
    filcon clear nombre-servidor.
    ```

6.3 Herramientas de monitorización

Existen herramientas de monitorización del sistema Coda que ayudan a comprobar su correcto funcionamiento. Las siguientes herramientas se utilizan desde el cliente Coda:

- `cmon` para monitorizar desde un cliente el estado de los servidores Coda, útil entre otras cosas para comprobar la conexión entre un cliente y sus servidores. Por ejemplo, con

  ```
  cmon -t 1 sipt30 sipt31
  ```

  se comprueba y se visualiza cada segundo el estado de los servidores `sipt30` y `sipt31`. Si un servidor no es accesible se visualizará una interrogación en los resultados estadísticos referentes al diagnóstico de un mismo servidor.

- `codacon` para visualizar las acciones del cliente.

- Ficheros *log* del cliente:

  ```
  $ tail -f /usr/coda/venus.cache/venus.log
  $ tail -f  /usr/coda/etc/console
  ```

Como ya se ha mencionado anteriormente, los servidores Coda guardan sus ficheros *log* en `/vice/srv/`.

7 Pruebas realizadas y resultados

Para nuestras pruebas con el Sistema Coda hemos dispuesto de dos servidores Coda y un cliente Coda. Los dos servidores forman parte de un mismo VSG replicado con identificador E0000100 y coinciden con la configuración del ejemplo de configuración utilizado en este documento.

El cliente Coda funciona perfectamente en modo conectado actualizando inmediatamente los cambios del cliente en los servidores. Sin embargo tras una desconexión la reintegración no se produce inmediatamente y puede llegar a tardar un tiempo variable (este proceso sigue siendo transparente para el usuario).

Cuando uno de los servidores cae o resulta inaccesible durante un periodo de tiempo en el que el otro servidor se ha modificado, el primero también tarda un tiempo variable en actualizarse tras su reconexión e igualmente resulta transparente para el usuario.

Hemos probado todas las configuraciones expuestas en este documento. Sin embargo Coda es un sistema relativamente complejo y no hemos trabajado con la base de datos Hoard ni con el sistema de copias de seguridad de Coda. Existen múltiples configuraciones y pruebas a realizar pero hemos preferido centrarnos en sus características más relevantes: la replicación y la computación móvil.

8 Bibliografía

[BRA98] Peter J. Braam: *The Coda Distributed File System.*
http://coda.cs.cmu.edu/ljpaper/lj.html.School of Computer Science, Carnegie Mellon University, february 1998.

[HAR98] Jan Harkes: *User Administration in Coda 5.2.x.*
http://coda.cs.cmu.edu/doc/html/pdbtool-mini-howto.html. School of Computer Science, Carnegie Mellon University, 1998.

[MAR99] Juan Antonio Martínez: *El sistema de ficheros Coda.*
Revista *Linux Actual* nº 9, páginas 20-22, diciembre 1999.

[SAT97-1] M. Satyanarayanan, Maria R. Ebling, Joshua Raiff, Peter J. Braam: *Coda File System. User and System Administrators Manual (E. Unix Manual Pages).*
http://coda.cs.cmu.edu/doc/html/manual-19.html. School of Computer Science, Carnegie Mellon University, august 1997.

[SAT97-2] M. Satyanarayanan, Maria R. Ebling, Joshua Raiff, Peter J. Braam: Coda File System. User and System Administrators Manual (3.6 Repairing an Inconsistent Directory). *http://coda.cs.cmu.edu/doc/html/manual-3.html*. School of Computer Science, Carnegie Mellon University, august 1997.

9 Agradecimientos

A Alberto Lafuente y a Igor Armendariz, profesores de la Facultad de Informática de San Sebastián de la Universidad Pública del País Vasco. Gracias a ellos hemos dispuesto de los mejores recursos en el Laboratorio E04 de la facultad para Instalar Debian en tres equipos y probar Coda.También queremos agradecer la importante valoración de este documento dentro de la asignatura `Sistemas Distribuidos` (¡gracias Alberto!). Sin la ayuda de ambos este *COMO* nunca hubiera existido.

A Francisco José Montilla Blanco del Insflug, por ayudarme con el sgml y con mis múltiples preguntas sobre los *COMOs*.

A Juan Antonio Martínez, por permitirme refundir algunos párrafos de su artículo *El sistema de ficheros CODA*.

Y en general a todos aquellos que creen en mi.

10 Anexo: El INSFLUG

El *INSFLUG* forma parte del grupo internacional *Linux Documentation Project*, encargándose de las traducciones al castellano de los Howtos, así como de la producción de documentos originales en aquellos casos en los que no existe análogo en inglés, centrándose, preferentemente, en documentos breves, como los *COMOs* y *PUFs* (**P**reguntas de **U**so **F**recuente, las *FAQs*. `:)`), etc.

Diríjase a la sede del Insflug para más información al respecto.

En élla encontrará siempre las **últimas** versiones de las traducciones «oficiales»: *www.insflug.org*. Asegúrese de comprobar cuál es la última versión disponible en el Insflug antes de bajar un documento de un servidor réplica.

Además, cuenta con un sistema interactivo de gestión de fe de erratas y sugerencias en línea, motor de búsqueda específico, y más servicios en los que estamos trabajando incesantemente.

Se proporciona también una lista de los servidores réplica (*mirror*) del Insflug más cercanos a Vd., e información relativa a otros recursos en castellano.

En *http://www.insflug.org/insflug/creditos.php3* cuenta con una detallada relación de las personas que hacen posible tanto esto como las traducciones.

¡Diríjase a *http://www.insflug.org/colaboracion/index.php3* si desea unirse a nosotros!.

Francisco José Montilla, *pacopepe@insflug.org*.