7. Gestión de Archivos

7.1. Introducción

Introducción

Básicamente, el propósito de todo sistema de ficheros es representar y organizar la información que se guarda en el sistema de almacenamiento. En sistemas UNIX no sólo es esto, es decir, contener ficheros y directorios, además el sistema de ficheros puede contener procesos, puertos serie, canales de comunicación y segmentos de memoria compartida de forma más o menos natural.

Por lo tanto un sistema de ficheros comprende cuatro componentes:

1. Un espacio de nombres. Es decir, una forma de nombrar objetos y organizarlos de forma jerárquica.
2. Una API (*Application Programming Interface*). Un conjunto de llamadas para manipular estos objetos.
3. Seguridad. Un esquema para proteger, esconder y compartir los objetos.
4. Implementación. El código para mantener lo anterior sobre la estructura del disco o sistema de almacenamiento.

No está definido que componente del sistema operativo debe mantener los cuatro componentes, en algunos casos es el propio subsistema de almacenamiento, en otros los propios drivers del kernel. Para complicar más la cuestión, casi todos los sistemas operativos modernos trabajan con más de un sistema de ficheros: el del propio UNIX, compatibles con MS-DOS, de CD, etc.

Independientemente de esto, un sistema de ficheros en una jerarquía que comienza en el directorio raíz "`/`" y continúa hacia debajo de forma arbitrariamente profunda con un número de subdirectorios (el número y nombre se verá en el siguiente apartado).

En sistemas modernos, cada elemento dentro del directorio raíz debe tener un nombre de no más de 255 caracteres y su pathmane (camino hasta llegar a él) no debe superar los 1023 caracteres. Los nombres pueden contener cualquier carácter salvo el "`/`" y el nulo. Incluso podrían tener espacios, lo cual es bastante incómodo, ya que los intérpretes de comandos (ver apuntes de la shell) lo usan como separador.

Estos elementos pueden ser de varios tipos:

1. Ficheros regulares. Es simplemente un conjunto de bytes sin ninguna estructura de contenido. Pueden ser accedidos de forma secuencial o aleatoria. Un enlace "*hard link*" es equivalente a un fichero.
2. Directorios. Fichero especial que contiene nombres a otros ficheros. Existen dos directorios especiales "`.`" Que se refiere a si mismo y "`..`" a su inmediato superior o padre.
3. Dispositivos de tipo carácter. Ficheros que permiten a los programas comunicarse con el *hardware*. Hay módulos del *kernel* (*drivers*) que se enlazarán con el respectivo fichero. Si son de tipo carácter, el *driver* podrá hacer *buffering* de los datos comunicados.
4. Dispositivos de tipo bloque. Si son de tipo bloque, los datos manejados son grandes paquetes y es el *kernel* es el que hará *buffering*.
5. *Sockets*. Son conexiones entre procesos. Pueden referirse a procesos locales o remotos. Los ficheros asociados sólo podrán ser escritos o leídos por miembros involucrados en la conexión.
6. FIFOS (*pipes* con nombre). Otra forma de conexión en este caso totalmente local.
7. Enlaces simbólicos ("*soft links*"). En un fichero que contiene el nombre (la referencia) a otro fichero.

En el sistema van a existir comandos para hacer este tipo de ficheros (no se describen aquí ya que están más relacionados con la programación de sistemas que con su administración). Podremos distinguir al hacer un `ls -l` ya que cada uno tiene asociado un carácter:

1. Fichero regular: "-".
2. Directorio: "d".
3. Dispositivo carácter: "c".
4. Dispositivo bloque: "b".
5. Socket: "s".
6. FIFOS: "p".
7. Enlace simbólico: "l".

Por último, cada fichero puede tener sus propios atributos que se pueden cambiar con el comando `chmod`. Atributos típicos son: `seguid`, `setgid`, `sticky bit` y permisos. Tampoco se ven aquí ya que están más relacionados con la programación.

7.2. Árbol de directorios

Uno de los problemas que existían a la hora de utilizar un sistema UNIX era la estructura de ficheros, que aunque similar, difería de unos a otros, a veces de forma significativa. Eso provocó que en Agosto de 1993 se hiciera un esfuerzo por reestructurar la estructura de directorios y ficheros. Un hijo de este estándar fue el FSSTND, exclusivo para LINUX, que fue creado el 14 de febrero de 1994. Otras versiones posteriores aparecieron meses después.

Después de la aparición de esta versión, con la ayuda de miembros de BSD, se hizo otro esfuerzo por ampliarla a otro tipo de sistemas UNIX, a esta versión se le cambió de nombre en reconocimiento de la generalización y se le llamó FHS (*Filesystem Hierarchy Standard*), es decir, Estándar de Jerarquía del Sistema Ficheros.

El documento que define el FHS es la referencia vinculante para cualquier sistema compatible FHS, sin embargo el estándar da pie a la extensibilidad de unas áreas o no define otras. En esta sección se proporciona un resumen del estándar y una descripción de aquellas partes del sistema de ficheros que no cubre el estándar. El estándar completo se puede ver en: http://www.pathname.com/fhs/ (ver esta página para la versión actual, la última es de mayo de 2001, versión 2.2).

Los directorios definidos en FHS que cuelgan del directorio raíz aparecen a continuación:

```
/
  |- bin              binarios de comandos esenciales
  |- boot             ficheros estáticos del cargador
  |- dev              ficheros de dispositivos
  |- etc              ficheros de configuración del sistema
  |- mnt              punto de montaje temporal
  |- lib              modulos del kernel y librerías compartidas
esenciales
  |- opt              paquete añadidos de software
  |- root             directorio home para el usuario root
  |- sbin             binarios de sistema usados por el root
  |- tmp              ficheros temporales
  |- usr              jerarquía secundaria
  |- var              datos variables
```

Hay otros tres directorios que deberán estar en el directorio raíz si es que son necesarios para el sistema:

```
/
  |- home             directorio de usuarios
  |- lib<esp>         librerías esenciales que son compartidas
alternativas
  |- root             directorio home para el root
```

Algunos de estos directorios del estándar son descritos a continuación:

El directorio `/dev`

El directorio `/dev` contiene archivos que representan dispositivos del sistema. Estos archivos son esenciales para el correcto funcionamiento del mismo.

El directorio `/etc`

El directorio `/etc` está reservado para archivos de configuración que afectan directamente al ordenador. No deben colocarse ejecutables en `/etc`. Los ejecutables que antiguamente se colocaban en `/etc` deberían estar en `/sbin` o posiblemente en `/bin`.

Los directorios `X11` y `skel` deben ser subdirectorios de `/etc`. El directorio `X11` es el que contiene archivos de configuración de **`X11`** como `XF86Config`. El directorio `skel` (ya hablamos de él en el capítulo de usuarios) contiene archivos de "esqueleto" (del inglés "*skeleton*"), es decir, archivos que se utilizan para rellenar el directorio raíz de un usuario cuando es creado.

El directorio `/lib`

El directorio `/lib` debe contienen sólo las librerías que son necesarias para ejecutar los ejecutables `/bin` y `/sbin`.

El directorio `/sbin`

El directorio `/sbin` es para ejecutables utilizados sólo por el superusuario y aquellos que se necesitan durante el arranque y para montar `/usr` y poder realizar mantenimiento y recuperación del sistema. El FHS especifica:

"`/sbin` suele contener archivos esenciales para iniciar el sistema además de los ejecutables de `/bin`. Lo que quiera que se ejecute una vez que `/usr` esté montado (cuando no hay problemas) debería colocarse en `/usr/sbin`. Ejecutables de administración del sistema propios del ordenador deberían emplazarse dentro de `/usr/local/sbin`."

Como mínimo, los siguientes programas deberían estar en `/sbin`:

`arp, clock, getty, halt, init, fdisk, fsck.*, ifconfig, lilo, mkfs.*, mkswap, reboot, route, shutdown, swapoff, swapon y update.`

El directorio `/usr`

El directorio `/usr` es para archivos que pueden ser compartidos en toda una organización. El directorio `/usr` suele tener su propia partición y debería poder ser montado en modo sólo lectura. Los siguientes directorios deberían ser subdirectorios de `/usr`:

```
/usr
  |- X11R6
```

```
|- bin
|- doc
|- etc
|- games
|- include
|- lib
|- libexec
|- local
|- sbin
|- share
+- src
```

El directorio `X11R6` es para las X (XFree86 en Red Hat Linux), `bin` es para ejecutables, `doc` es para documentación varia, `etc` es para archivos de configuración comunes a la organización, `games` es para juegos, `include` es para archivos de cabecera de C, `lib` contiene librerías, `libexec` contiene pequeños programas auxiliares llamados por otros programas, `sbin` es para ejecutables de administración del sistema (aquellos que no son necesarios para la inicialización que estarían en `/sbin`), `share` contiene ficheros que no son de arquitecturas especificas y `src` es para código fuente.

El directorio `/usr/local`

El FHS dice:

"La jerarquía `/usr/local` será utilizada por el administrador del sistema al instalar software localmente. Debe estar a salvo de poder ser sobreescrita cuando se actualice software del sistema. Puede ser utilizada para programas y datos, compartibles entre un grupo de ordenadores, que no se encuentren en `/usr`."

El directorio `/usr/local` es similar en estructura al directorio `/usr`. Tiene los siguientes subdirectorios con una intención similar a la de aquellos que se encuentran en el directorio `/usr`:

```
/usr/local
       |- bin
       |- doc
       |- etc
       |- games
       |- info
       |- lib
       |- man
       |- sbin
       +- src
```

El directorio `/var`

Puesto que el FHS requiere que `/usr` pueda ser montado en modo sólo lectura, cualquier programa que escriba archivos de trazas o necesite los directorios `spool` o `lock` (por ejemplo la impresora) debería escribir en el directorio `/var`. FHS lo describe así:

"...archivos de datos variables. Incluye directorios y archivos de `spool`, datos administrativos y de traceo, y ficheros temporales y transitorios."

Los siguientes directorios deberían ser subdirectorios de `/var`:

```
/var
  |- cache
  |- db
  |- ftp
  |- gdm
  |- lib
  |- local
  |- lock
  |- log
  |- named
  |- nis
  |- opt
  |- preserve
  |- run
  +- spool
       |- anacron
       |- at
       |- cron
       |- fax
       |- lpd
       |- mail
       |- mqueue
       +- news
       |- rwho
       |- samba
       |- slrnpull
       |- squid
       |- up2date
       |- uucp
       |- uucppublic
       |- vbox
       |- voice
  |- tmp
  |- yp
```

Archivos de trazas del sistema como `wtmp` y `lastlog` van a `/var/log`. El directorio `/var/lib` también contiene las bases de datos de los RPMs del sistema. Los ficheros `lock` van a `/var/lock`. El directorio `/var/spool` contiene subdirectorios para los varios sistemas que necesitan almacenar archivos de datos.

`/usr/local` en Red Hat LINUX

En Red Hat LINUX, el uso que se pretende de `/usr/local` es ligeramente diferente del especificado por el FHS. El FHS dice que `/usr/local` debería ser donde se almacene software que debe permanecer intacto tras actualizaciones de software del sistema. Puesto que las actualizaciones desde Red Hat son seguras gracias al sistema RPM y Gnome-RPM, no es necesario proteger archivos colocándolos en `/usr/local`. En vez de ello, es mejor utilizar `/usr/local` para software que sólo se instale en el ordenador local. Por ejemplo, imaginemos que se ha montado `/usr` vía NFS en modo sólo lectura. Si hay un paquete de software o programa que se quiere instalar, pero no se permite la escritura, debería instalarse en `/usr/local`. Más adelante quizás, si se convence al administrador del sistema NFS para que instale el programa en `/usr`, puede desintalarse de `/usr/local`.

Hay directoriosespecíficos de Linux que también están en el estándar como:

El directorio `/proc`

El directorio `/proc` contiene archivos especiales para obtener o enviar

información del o hacia el *kernel*. Es un método sencillo de acceder a información sobre el sistema operativo usando el comando `cat`. En otros sistemas pueden ser imágenes de todos los procesos que se están ejecutando.

Otros directorios y ficheros no pertenecientes al estándar

Existen otros directorios que no pertenecen al estándar y que están en muchos sistemas entre ellos podemos destacar:

```
/
  |- sys          ficheros de configuración para BSD y área de trabajo del
kernel
  |- kernel       ficheros necesarios para cargar el kernel en Solaris.
```

7.3. Creación

Creación de un sistema de ficheros

Una vez que hemos descrito cual es la visión que un usuario tiene del sistema de ficheros, vamos a detallar cual sería el proceso para construir (crearla para que un usuario lo pueda utilizar) uno. Parte de los pasos necesarios ya los hemos realizado en la instalación del sistema, pero de todos modos vamos a resumirlos en los siguiente puntos:

1. Conexión del dispositivo al computador.
2. Creación del dispositivo a través del cual accedemos al disco.
3. Formateo del disco.
4. Etiquetado y particionado del disco.
5. Creación de un sistema de ficheros UNIX dentro de las particiones.
6. Verificación del sistema de ficheros.
7. Montaje del sistema.

Una vez que hemos superado el primer punto, esto es, conectar el dispositivo al computador, lo que tendremos que hacer es crear un fichero de dispositivo en `/dev` asignado al disco (disco duro, cdrom o disquete ...). Este paso dependerá en gran manera del sistema UNIX con que contemos, básicamente existen dos tipos de dispositivos, los de bloque o acceso aleatorio y los de carácter o acceso secuencial, la gran diferencia entre ellos es que el de bloque almacena los datos y estos pueden ser accedidos en cualquier orden (de forma aleatoria) y el de carácter no, simplemente nos dará la siguiente ristra de bytes. Al hacer un `ls -l /dev` veremos en la primera columna de datos una `b` o una `c` que nos indicará el tipo de dispositivo, además, antes de la fecha, veremos dos cifras que son el número mayor y el menor, el mayor indica al *kernel* cual de los controladores corresponde al dispositivo y el menor a la copia del mismo, por ejemplo un número mayor 3 indica un disco duro de tipo IDE y el número menor será cero para el primer disco y del uno al 15 posibles particiones del mismo. En la mayoría de los sistemas existen suficientes dispositivos en `/dev` para todos los dispositivos que podamos conectar a nuestra máquina, por ejemplo, en mi sistema hay 140 dispositivos de disco duro IDE ya realizados (en el capítulo 1 veíamos la correspondencia entre discos IDE y SCSI [y sus particiones] y los ficheros de dispositivos de `/dev`). Si creamos uno, tendremos que tener mucho cuidado con los permisos que coloquemos al fichero, dando sólo permiso de escritura al root y como mucho a un grupo especializado.

El siguiente paso será el formateo del disco, en algunos casos, como los discos SCSI, este paso viene ya realizado de fábrica. Después se procede al etiquetado y particionado del disco, si es que nos interesa que ese disco tenga particiones. Este paso lo hemos realizado en la instalación del sistema con el programa **diskdruid**, pero una vez instalado tenemos el programa **fdisk**.

Realizadas las particiones, tenemos que colocar sobre ellas un sistema de ficheros (salvo en la partición que hayamos dedicado a memoria secundaria [swap], ya que no tiene un sistema de ficheros normal y la partición se condiciona con **mkswap** y se activa o desactiva con **swapon** y **swapoff**). Esto se realiza normalmente con el comando ***mkfs*** o ***newfs***. Estos comandos son actualmente un *front-end* de una serie de programas a los que llaman, constituidos por el nombre "`mkfs`" y el sufijo con el tipo de sistema de ficheros que queremos, todos ellos en el directorio `/sbin`.

La estructura de cada sistema de ficheros está compuesta de cuatro grandes partes:

1. Bloque de arranque (boot), normalmente está en el primer sector del sistema de ficheros y contiene un pequeño programa que se encarga de buscar el sistema operativo y cargarlo en memoria.
2. Superbloque, es el bloque que describe el sistema de ficheros dando información sobre su tamaño, lista de bloques disponibles, primer bloque libre, número de iodos, lista de inodos libres, índice del primer inodo libre, flag de modificación, etc.
3. Lista de inodos, es una lista donde se guarda una entrada (inodo) por cada fichero, donde se almacena información respecto del mismo, como situación del mismo, propietario, permisos, fecha y hora de modificación, etc.
4. Bloque de datos, es la zona donde se almacenan los ficheros del sistema y a la cual hacen referencia los inodos, como hemos dicho, los bloques tienen un tamaño fijo en cada sistema de ficheros y sólo pueden estar ocupados por un sólo fichero, aunque éste no ocupe toda su longitud.

En LINUX, el comando **mkfs** servirá para construir típicamente hasta media docena de tipos de sistemas de ficheros, desde MS-DOS, al extendido nativo de Linux, terminando en el extendido doble **ext2**, que es el que se recomienda para uso normal. Además, permitirá formatear el sistema de forma automática, esto nos evita conocer el formateo para cada uno de los tipos de sistemas. A este comando se le dan como mínimo dos datos: el tipo de sistema de ficheros y el dispositivo sobre el que queremos construirlo. Dependiendo del primer argumento, **mkfs** llamará a otro programa **mkfs.sistema** residente en `/sbin` (estas terminaciones nos indicarán que tipo de sistemas de archivos podemos tener, suelen aparecer en `/proc/filesystems`), incluso existen dos comandos especializados que son **mkdosfs** y **mke2fs**.

Esto exige conocer con cierto detalle el dispositivo sobre el que queremos formar el sistema de ficheros. Así, otras opciones suplementarias para el comando pueden ser:

- `-c` para chequear los bloques del dispositivo.
- `-v` para que presente lo que va haciendo paso a paso o modo verboso.

Si se llama directamente a ***mk2efs*** se tendrá un control más amplio sobre el sistema de ficheros construido:

- `-b` para indicar el tamaño de bloques en bytes.
- `-i` para indicar el número de bytes por ***inodo***, o dicho de otra manera, el número de *inodos* que se crearán en el sistema de ficheros. Sólo se indicará este ratio si sabemos con seguridad el número de ficheros que habrá en ese sistema o el tamaño típico de los mismos.
- `-N` directamente se pone el número de inodos.
- -m para decir el porcentaje del sistema reservado para el superusuario, por defecto un 5%.
- -L coloca la etiqueta del sistema.
- -S sólo crea el superbloque, esto es útil cuando sólo este bloque está corrupto.

Así, si quisiéramos realizar un sistema de ficheros de tipo `ext2` en un disquete (dispositivo `fd0`), tendríamos que hacer simplemente (si quisiéramos hacerlo compatible con windows se pondría `-t vfat`):

```
mkfs -t ext2 /dev/fd0
```

Una vez que tenemos sobre cada partición un sistema de ficheros, nos queda un último paso más que es

el montaje del sistema de ficheros en el árbol de directorios general. Esto se hace con la orden **`mount`**, a la que normalmente hay que dar dos argumentos: el dispositivo que sostiene a la partición (puede también ser un nombre de ordenador si se monta por red) y un directorio vacío (si no está vacío todo lo que contiene se ocultará) del árbol general. Cada vez que se monte un sistema de ficheros se creará automáticamente un directorio llamado `lost+found` que es usado por el comando **`fsck`** (ver siguiente apartado) en situaciones de emergencia.

La órden `mount` tiene una serie de opciones, dos de las más importantes son:

- `-t` para dar el tipo de sistema.
- `-o` seguido de un nombre para decir como se debe efectuar el montaje (que en muchos casos depende del tipo de sistema), las más frecuentes son:
  - ***auto/noauto***, para indicar al sistema de arranque que monte esos sistemas automáticamente o no lo haga. Cuando se usa la opción `-a` , usualmente escogida por el arranque, se montarán todas las unidades especificadas que no tengan la opción ***noauto***.
  - ***exec/noexec***, para permitir o no al ejecutar programas de ese sistema de ficheros.
  - ***user/nouser***, para permitir que ese sistema pueda ser montado por un usuario normal, por defecto, para proceder al montaje de un sistema, hay que ser *root*.
  - ***sync/nosync***, después de realizada una escritura, el programa puede esperar a que se transfieran los datos al dispositivo o no, la opción por defecto es no, ya que suele ser mucho más óptima en discos duros y NFS, aunque también puede producir perdidas de datos en situaciones de emergencia.
  - ***ro/rw*** para decir si el sistema es de solo lectura o de lectura-escritura.

La orden `mount` se basa en un fichero de texto editable (con sumo cuidado, ya que cualquier error hará que no se monte una unidad) que contiene los sistemas que están montados o se pueden montar, este fichero se llama `/etc/fstab` (en algunos sistemas `/etc/vfstab` o `/etc/checklist`). La información está dividida en columnas para que pueda ser entendida tanto por "humanos como por programas":

- En la primera columna aparecen los dispositivos.
- En la segunda los directorios de montaje.
- En la tercera el tipo de sistema de ficheros.
- En la cuarta las opciones que existen para el montaje.
- En la quinta el número de días que han pasado sin realizar un **`dump`**. Esta información será utilizada por programas de *back up* (ver un próximo apartado en este capítulo), si es cero o no aparece no será necesario hacer *back up*.
- En la sexta el orden en que deberán hacerse los chequeos de esa partición (ver apartado posterior), empezando por uno. Si dos particiones tienen el mismo número, el chequeo se hará en paralelo (no tiene sentido poner el mismo número a particiones del mismo disco), si aparece un cero no será necesario hacer chequeo.

El aspecto de un fichero `/etc/fstab` podría ser:

```
LABEL=/           /                ext2    defaults         1 1
LABEL=/boot       /boot            ext2    defaults         1 2
LABEL=/home       /home            ext2    defaults         1 2
/dev/cdrom        /mnt/cdrom       iso9660 noauto,owner,ro  0 0
/dev/fd0          /mnt/floppy      msdos   noauto,owner     0 0
none              /proc            proc    defaults         0 0
none              /dev/pts         devpts  gid=5,mode=620   0 0
/dev/hda6         swap             swap    defaults         0 0
```

donde se indica en las tres primeras líneas las tres particiones del disco duro y donde tienen que ser montadas (en este caso a través de la etiqueta, pero también podría aparecer el dispositivo), en las dos siguientes el *cdrom* y la unidad de disquetes (se observa que el tipo de sistema de ficheros ha cambiado) y en las tres últimas, dispositivos especiales como el *swap* o el directorio `/proc`, usado por el kernel para informar del sistema a programas de usuario.

Si el sistema de ficheros que queremos montar aparece en el fichero `/etc/fstab`, bastará con que indiquemos el dispositivo, todas las demás instrucciones las tomará `mount` de ese fichero. Así, para montar la disquetera bastará con que hagamos:

```
mount /dev/fd0
```

para que se monte el disquete en `/mnt/floppy`. Si por cualquier causa el directorio no existe o el sistema ya está montado la orden puede generar un error. Muchas veces, como truco, se coloca un fichero con un nombre adecuado como "`disquete_no_montado`" en el directorio de montaje, de esta manera al hacer un `ls` podremos ver si el sistema está montado o no (si está montado, el fichero se oculta y aparecerá el directorio del disquete).

Para ver las particiones que están montadas, se puede utilizar `mount`, `mount -l` o `mount -v`. No montarán nada, sólo nos ofrecerán la información, en el caso `-l` incluyendo las etiquetas de los dispositivos.

Si queremos desmontar una unidad se emplea la órden **`umount`**, seguida del dispositivo. Normalmente si estamos usando un sistema de ficheros (fichero abierto o un proceso ejecutándose), el sistema no nos dejará desmontarlo, podemos comprobarlo con la orden **`fuser`** seguida del punto de montaje.

Si estamos en un sistema LINUX, distribución Red Hat, disponemos de una herramienta gráfica de gestión del sistema de código abierto que es `Linuxconf`. En ella hay un apartado referido a la gestión del sistema de ficheros.

7.4. Verificación

Una vez creados y montados los sistemas de ficheros, puede ocurrir que estos se deterioren por varias causas:

- Si es un dispositivo extraible, puede haber sido extraído a destiempo.
- Alguna pérdida de corriente eléctrica.
- Algún usuario, que no ha sido llevado por el sendero de la inteligencia, ha apagado el ordenador por las buenas o más dudosamente por error.
- Algún fallo hardware.

Un sistema UNIX, como parte del arranque (`rc.sysinit`), siempre ejecuta un mecanismo de chequeo de los sistemas de ficheros, esto se realiza a través del comando ***fsck*** (existe una versión especializada para sistemas extendidos llamada ***e2fsck***) y deberá ser ejecutado cuando la unidad esté desmontada. El arranque sabe cuando un sistema está "limpio" porque cuando se echa abajo un sistema se produce el desmontaje de las unidades y el kernel escribe una huella indicando que no ha habido errores, después cuando se arranque y se montan de nuevo, esta huella se elimina y no se hace reparación.

Si por cualquiera de las causas anteriores la huella no está escrita, el kernel lo sabrá y procederá a realizar el chequeo (en muchos casos también el chequeo se realiza de forma programada, por ejemplo cada 20 encendidos o cada seis meses) utilizando:

```
fsck -V -a /
```

para realizar el chequeo del sistema raíz de forma no interactiva (`-a`) y verbosa `(-V)`. Y a continuación se ejecuta:

```
fsck -R -A -V -a
```

para el resto de unidades (`-A`) menos la raíz (`-R`) que ya está y de la misma forma que antes.

Si el programa no puede reparar los errores, se ejecutará una *shell* para permitir que el administrador ejecute `fsck` directamente de forma interactiva (sin `-a`). Aquí pueden empezar los problemas, ya que el programa nos puede pedir que eliminemos restos del sistema de ficheros o nos irá indicando los problemas existentes de una forma poco comprensible, por ejemplo:

```
existe un problema en el inodo 40834, desea fijarlo ¿?.
```

no nos quedará otro remedio que decir que si.

Si aun así no se puede reparar el sistema, puede ser debido a que el superbloque se haya corrompido. Dependiendo del tipo de sistema, pueden guardarse copias de seguridad de este superbloque a intervalos definidos, por ejemplo en ***ext2*** cada 8 kilobytes (existe un comando para modificar los parámetros de un sistema extendido llamado ***tune2fs***) y así restaurar una de las copias con:

```
fsck -t ext2 -b 8193
```

Si incluso esto falla (extraño y doloroso), conviene recurrir a la religión o a algún gran experto. Lo más normal es simplemente chequear el sistema de ficheros que nos interese. Por ejemplo, si lo quisiéramos hacer con un disquete (desmontado y compatible con windows) deberíamos ejecutar:

```
fsck -t msdos /dev/fd0
```

y nos respondería con:

```
Parallelizing fsck version 1.18 (11-Nov-1999)
dosfsck 2.2, 06 Jul 1999, FAT32, LFN
/dev/fd0: 0 files, 0/2847 clusters
```

ya que siempre intentará hacer el chequeo en paralelo.

7.5. Control

Una tarea importante de un administrador es controlar el uso de disco por parte de los usuarios, esto se puede hacer de dos maneras: una sometiendo a presión a los usuarios “de alguna manera” y otra poniendo alguna cuota de disco (si no se tiene un sistema de cuotas otra solución consiste en poner a determinados usuarios en una partición con un tamaño limitado). Esto se puede hacer con antelación (a la hora de construir el sistema de ficheros) o a posteriori.

El gasto de disco usado se puede obtener con dos comandos: ***du*** y ***df***:

- ***du*** nos dará información sobre el gasto de disco por ficheros y directorios de forma recursiva, la información será ofrecida por directorios de abajo a arriba. Esta información puede darse de varias maneras dependiendo de las opciones que se utilicen: `-b` en bytes, `-k` en kilobyes (o bloques), `-h`

en formato “humano”, etc. Una opción muy útil es `-s` (sumario) que nos da sólo los totales, en vez de todo el árbol.
- ***df*** nos dará un resumen del gasto por sistemas de ficheros montados. Tiene las mismas opciones que el anterior, pero además, aparte de darnos la información en bloques, nos la puede dar en *inodos*, es decir, en número de ficheros. Esta información la podemos utilizar en las cuotas de disco, ya que se pueden poner de dos clases: bien por espacio o bien por número de ficheros.

A la hora de fijar un cuota, se podrá introducir un periodo de gracia en el que los usuarios puedan sobrepasar esa cuota antes de que el sistema les bloquee su funcionamiento normal, pidiendo perentoriamente que liberen espacio de disco. La filosofía de la cuota es la siguiente: hay dos parámetros: el límite blando y el duro. Cuando el usuario sobrepasa el blando es avisado por el sistema para que libere espacio, el duro jamás se puede sobrepasar. Como hemos dicho, se puede especificar un tiempo de gracia para el límite blando (por defecto está entre 3 y 7 días) pasado el cual, ambos límites igualan su propósito y el usuario será bloqueado (no puede hacer nada útil salvo liberar espacio).

Para que el sistema de cuotas funcione, se debe dar un requisito: el *kernel* debe estar notificado de que el sistema está implantado (conviene recordar que no siempre es así, y que en muchos casos no está ni siquiera instalado el sistema de cuotas). Esto se hace, o bien de forma permanente, poniendo para alguna unidad en el fichero de montaje `/etc/fstab` como opción `quota` (no hacer caso de que está opción sea ignorada), o bien de forma temporal utilizando el comando ***quotaon***. Una vez hecho, aparecerá un fichero llamado `quotas` en el directorio raíz de la unidad (partición) elegida.

Modernamente, y esto ocurre en GNU/Linux, el sistema de cuotas puede ser puesto a los usuarios individuales o a los grupos, lo que permite un poco más de juego a la hora de poner las mismas, desgraciadamente, cuando un usuario puede estar en varios grupos, no tiene mucho sentido. Si utilizamos este sistema, podremos tener dos ficheros de cuotas: `quota.user` y `quota.group` en el mismo sito de `quotas`, que no estará.

Los comandos relacionados con las cuotas son los siguientes:

- ***quotaon***. Comando en `/usr/sbin`. Tiene su contrapartida en ***quotaoff***. Es el encargado de notificar al *kernel* (sistema) que ha sido activado el servicio de cuotas.
- ***quotacheck***. Comando de `/sbin`. Realizará un chequeo del sistema de ficheros indicado (bien con la etiqueta o directorio de montaje, bien con la unidad de `/dev`) y creará o actualizará el fichero de cuotas, sea cual sea.
- ***repquota***. Comando en `/usr/sbin`. Dará un resumen de cómo están las cuotas para una unidad.
- ***edquota***. Comando en `/usr/sbin`. Servirá para cambiar las cuotas (o bien el tiempo de gracia con `-t`). Este comando llamará al editor por defecto (si no se ha cambiado será el `vi`) para editar un fichero donde están definidas las cuotas para el usuario (por defecto) o el grupo invocado. El aspecto de ese fichero podría ser:

```
Quotas for user alumno:

/dev/hda5: blocks in use: 384, limits (soft = 1000, hard = 1100)

    inodes in use: 94, limits (soft = 500, hard = 550)
```

Un inconveniente de las cuotas es que pueden reducir el rendimiento de un disco incluso hasta un 30%, por eso se suelen poner en determinados directorios de usuarios, pero casi nunca en el directorio raíz, que es donde se desarrolla la mayoría de actividad de disco.

Las cuotas en Red Hat también se pueden establecer con la herramienta gráfica ***Linuxconf***, en el apartado de gestión del sistema de ficheros.

7.6. Copias

Copias de seguridad

La realización de copias de respaldo (*backup*) es posiblemente la tarea más importante, conocida y tediosa de todo administrador de sistemas. Y es la más importante porque hay cientos de maneras "creativas" de perder datos, desde desastres naturales, a problemas hardware o software, pasando por acciones involuntarias no deseadas.

Antes de ver como se hace una copia de respaldo conviene estar familiarizados con los distintos dispositivos hardware que existen y las diferentes técnicas (sistemática) que se utilizan para realizar verdaderamente una copia de seguridad fiable y eficaz.

En lo que se refiere a los dispositivos ("media") sobre los que podemos realizar un *backup* los podemos dividir en tres tipo básicos:

- Disquetes. Es el peor media sobre el que realizar copias, ya que es con diferencia el más pequeño en capacidad y rapidez, lo que hace que el costo por *megabyte* sea uno de los más elevados. La gran ventaja este medio es su gran disponibilidad.

- Disco Duro. Sería la mejor forma de hacer un *backup* si su disponibilidad fuera adecuada, pero la verdad es que pocas veces se puede disponer de otro disco duro para hacer exclusivamente un *backup*, además sería conveniente que el disco duro estuviera conectado en otra máquina, de nada sirve hacer un *backup* de una parte de un disco en otra parte.

- Cinta y otros dispositivos. La cinta es el dispositivo genuino para hacer *backup*, pero actualmente existen otros dispositivos como los *flopticals* o los *cd-rom* de una o varias escrituras para hacer copias. A la hora de escoger el dispositivo, tendremos que tener en cuanta, de acuerdo a nuestras necesidades, sus características, las más importantes son: costo, capacidad de almacenamiento, disponibilidad y rapidez. Veamos en la siguiente tabla comparativa las características de diferentes medios de almacenamiento (M significa magnético y O óptico):

Medio	Cap. med.	Coste	Tipo	Reu.	Acceso
Disquete LS	120 Mb.	1800	M	SI	Random
Cartuchos (Travan)	1 Gb.	4000	M	SI	Secuencial
Cinta DLT	15 Gb.	7000	M	SI	Secuencial
JAZ	2 Gb.	17000	M	SI	Random
flopticals	500 Mb.	1500	M/O	SI	Random
Cartucho QIC	4 Gb.	3000	M	SI	Secuencial
Cintas (Obsoletas)	200 Mb.	2500	M	SI	Secuencial
CD-ROM	600 Mb.	200	O	NO	Random
CD-ROM WR	600 Mb.	500	O	SI	Random
Cartucho 8mm. (exabyte)	5 Gb.	2000	M	SI	Secuencial
DAT 4mm. (DDS)	5 Gb.	4000	M	SI	Secuencial

Un parámetro interesante de calcular es el coste en pesetas por *megabyte*, donde como se ve, obtendríamos ventajas en el último formato.

En lo que se refiere a la sistemática, tenemos que tener en cuenta que la copia de un sistema de ficheros completa es bastante costosa de hacer y puede requerirnos varias cintas (si éste es el dispositivo elegido). Por eso existen dos tipos de copias de respaldo: las absolutas que comprenden todo el sistema y las incrementales que son copias parciales con respecto a la anterior, es decir, sólo se copian los ficheros modificados respecto de la última copia, ya sea absoluta o incremental.

Por eso se debe diseñar una planificación de copias, "*scheduling*", de acuerdo a nuestras necesidades, es

decir, la actividad de nuestros sistemas de ficheros, la capacidad del dispositivo escogido, el nivel de redundancia que queremos y el número de cintas (u otro media) que podemos comprar.

Si el sistema de ficheros es menor que la capacidad de la cinta, el esquema puede resultar muy sencillo, hacer copias de seguridad absolutas por la noche cada día. Esto no suele ser así por lo que se escogen planificaciones más sofisticadas (incluso siguiendo el algoritmo de las torres de Hanoi), la más usual suele ser hacer una copia completa por ejemplo los domingos (día de menor uso supuestamente) y copias incrementales cada día por la noche. Esto se puede modular teniendo en cuenta, como hemos dicho, la características de nuestro equipo: no todos los sistemas de ficheros tienen la misma actividad, cuanta más actividad mayor deberá ser la frecuencia de la copia, siendo innecesario hacer copia de algunos directorios como el `/tmp`, si el número de cintas y su capacidad es pequeño, deberemos reducir el régimen de copiado y por último deberemos tener en cuenta en grado de redundancia, normalmente se deberán tener dos copias del sistema que deberán ser guardadas en distintos sitios por razones de seguridad y además porque cuando estemos haciendo una copia el sistema también puede fallar y quedarnos sin el sistema y sin la copia.

¿Cómo podemos realizar lo explicado hasta ahora con UNIX/LINUX? Existen una serie de comandos para realizar las copias de seguridad, además de para el empaquetado. Los más usuales son:

- dump / restore.
- tar, cpio.
- gzip / gunzip, zip / unzip, compress / uncompress.

La primera pareja de comandos es la más habitual a la hora de hacer copias de seguridad, las otras dos (o su combinación ***tar*** / ***gzip***) son más utilizadas para hacer copia de ficheros para mover entre equipos (o si se prefiere copias de seguridad domésticas).

El comando ***dump*** lo que hace básicamente es empaquetar un conjunto de ficheros (teniendo en cuenta el anterior ***dump*** a través de una pequeña base de datos) en un gran archivo que es volcado en un dispositivo externo. Además tiene como ventajas que ese archivo volcado puede estar localizado en varias cintas, ya que según sus cálculos lo dividirá en varios volúmenes, que puede estar compuesto de cualquier tipo de ficheros, incluso de `/dev` o con agujeros, que los permisos y fechas son guardados, y que las copias pueden seguir el esquema incremental que hemos fijado. La desventaja relativa (ya que cada sistema de ficheros tiene una actividad) es que ***dump*** sólo puede hacer copias de un único sistema de ficheros, por lo que si tenemos un disco partido, serán necesarios varios ***dump***, además estos sistemas de ficheros deberán ser locales (no NFS) aunque la unidad de cinta puede estar en otra máquina si usamos ***rdump*** (dump remoto).

Para poder hacer copias incrementales, ***dump*** utiliza un sistema de niveles, desde el cero hasta el nueve. El nivel cero indicará un volcado (*dump*) absoluto, el resto de niveles serán relativos al anterior (no tienen por qué ser consecutivos). Así, podemos utilizar, siguiendo nuestra supuesta planificación, un nivel cero los domingos y el resto de los días un nivel superior, de uno a seis (o cualquiera de las combinaciones siempre que se usen números de menor a mayor). Para ello ***dump*** utiliza el fichero `/etc/dumpdates` (pequeña base de datos) que actualiza en cada copia siempre que usemos la opción `-u`.

Los argumentos habituales de ***dump*** serán: el nivel de copia, las opciones (normalmente `-u`), el dispositivo de salida (en algunos casos son la opción `-f`) y el directorio (o sistema de ficheros) sobre el que queremos hacer la copia. Además, deberemos tener cuidado con la longitud de la cinta que estamos usando (si hacemos varios dump en una cinta no se deberá rebobinar automáticamente), ya que si sobrepasamos ésta, el ***dump*** se estropeará. ***dump*** asume unos determinados parámetros para cada dispositivo (incluido el por defecto si no usamos `-f`), si estos no coinciden con la cinta que hemos introducido en esa copia deberemos usar las opciones `-s` y `-d` que indicarán el tamaño y la densidad de la misma. También se puede usar la opción `-a` que indicará que se escriba hasta el final de la cinta, esto es útil cuando tenemos varios volúmenes en una cinta o se hace un *backup* con compresión.

En el siguiente ejemplo haremos una copia absoluta (0) adaptando el fichero `/etc/dumpdates`, en el dispositivo st0 del directorio `/usr/src`:

```
/sbin/dump -0u -f /dev/st0 /usr/src
```

Siempre que realicemos unas copias de respaldo tendremos que tener en cuenta los siguientes consejos de "la abuela":

- Es conveniente, cuando se tiene una red de computadores, centralizar las copias de *backup* en uno solo utilizando ***rdump***. Así la administración se hará más fácil.
- Las cintas u otros medios usados para hacer el *backup* deberán estar etiquetados de forma unívoca y clara. Una cinta sin etiquetar será en la práctica una cinta en blanco.
- Seleccionar un intervalo temporal entre copias adecuado, ni tan pequeño para sobrecargar al administrador y la máquina, ni tan grande como para correr riesgos de pérdidas de datos importantes.
- Escoger los sistemas de ficheros de forma adecuada. Si hay un sistema de ficheros que prácticamente no sufre cambios es innecesario hacer sobre él una copia absoluta de forma periódica. Al igual pasa con el directorio `/tmp`.
- Si se tienen dispositivos de alta capacidad como las *dds*, no cuesta mucho programar el comando ***cron*** para que nos haga una copia incremental cada noche de los sistemas con bastante actividad.
- Sería conveniente siempre hacer sistemas de ficheros más pequeños que el dispositivo de volcado.
- Cuando se pueda, sería conveniente guardar las cintas en un sitio distinto al de otra copia o del computador, y a ser posible, en un sitio seguro, dependiendo de las necesidades del usuario (empresa).
- Siempre es conveniente, para mantener la coherencia de las copias, hacerlas en periodos de baja actividad del sistema, incluso hay aplicaciones de *backup* que lo aseguran.
- Siempre que se hace una copia conviene, aunque se tarde más tiempo, hacer una comprobación de lo copiado. En muchas ocasiones cuando se va echar mano de un *backup* resulta que está estropeado o no realizado cuando ya es demasiado tarde. Veremos en los siguiente párrafos como hacerlo.
- Como regla general siempre hay que aplicar la ley de "*Murphy*" y ponerse en el peor caso.

Una vez que hemos realizado la copia de respaldo, deberemos tener una forma de poder comprobar lo escrito y además leerlo en caso de apuro. Esto se realiza con el comando ***restore***. Las opciones más normales en LINUX son las siguientes:

- –C: Hace una comprobación del sistema de ficheros guardado, comparándolo con el que está en disco.
- –i: Este modo permite la restauración interactiva de los ficheros guardados en una copia hecha con ***dump***. Es el método recomendado cuando se quieren hacer extracciones de ficheros seccionados. Cuando se ejecuta así, el comando ejecuta una *shell* que nos permitirá:
  - Añadir con ***add*** ficheros a restaurar.
  - Cambiar de directorio con ***cd***.
  - Quitar ficheros a extraer con ***delete***.
  - Extraer los ficheros seleccionados con ***extract***.
  - Ejecutar ***ls*** y ***pwd*** para ver un directorio y decirnos cuál es.
  - Ejecutar ***Quit*** para salir.
- –r: Extrae (restaura) un sistema de ficheros completo (antes de recuperar un sistema completo deberemos asegurarnos que el problema que lo causó ha sido eliminado). Antes de poder proceder, el sistema a restaurar deberá haber sido creado y montado. Una vez hecho esto, buscaremos el *backup* de nivel cero (absoluto o completo) más reciente y ejecutaremos ***restore –r***. Después restauraremos en el mismo orden en que fueron hechos con ***dump***, los posteriores respaldos incrementales. Si debemos movernos por una cinta con varios volúmenes, tenemos el comando ***mt*** para posicionarnos donde nos interese.
- –x: con un fichero o directorio de argumento lo extrae del respaldo instalado.

Los comandos ***dump*** y ***restore*** no son los únicos comandos para realizar copias, de hecho son los mejores para hacer *backup* de la forma que hemos indicado, pero por ejemplo son demasiado complejos para copiar en algún sitio (cinta, disquete, otra parte del disco, ...) un conjunto más o menos pequeño de ficheros.

El primero de ellos es el comando ***tar*** que lo que hace es empaquetar en un único fichero un conjunto de ficheros y directorios que queramos copiar, también serviría para realizar un *backup* de algo que preveamos con seguridad que necesitaremos restaurar posteriormente, es decir, es el perfecto sustituto para copiar una serie de ficheros de un medio a otro. Además tiene la ventaja que restituye los atributos del fichero como fueron almacenados (–p), pero también cuenta con las siguientes desventajas: no maneja varias cintas, ni ficheros de `/dev`, ni caminos de ficheros de más de 100 caracteres (la versión

GNU si lo hace pero desde un ***tar*** viejo no lo podríamos leer) y lo que escribamos tiene que caber en el tamaño de la cinta. Las principales opciones de tar son:

- -A, --catenate, --concatenate. Añade ficheros a un ***tar***.
- -c, --create. Crea un nuevo archivo ***tar***.
- -d, --diff, --compare. Encuentra diferencias entre los archivos de ***tar*** y los del disco.
- --delete. Borra ficheros del ***tar***, no se puede usar en cintas (acceso secuencial).
- -r, --append. Añade ficheros al final del ***tar***.
- -t, --list. Lista el contenido de un ***tar***.
- -u, --update. Sólo añade ficheros que sean más nuevos que los del ***tar***.
- -x, --extract, --get. Extrae ficheros de un ***tar***.

También se suelen usar: −v para que nos indique lo que está haciendo, −p para preservar los permisos de los ficheros y −f para indicar un fichero ***tar*** o un dispositivo distinto del de por defecto. Por último, indicar que los ficheros ***tar*** suelen tener esa extensión. Así, para copiar todo el sistema de directorios en una cinta por defecto se debería hacer algo tan sencillo como:

```
tar  -c /
```

Si fuera otro dispositivo lo tendríamos que indicar, por ejemplo copiar un directorio personal en el disquete:

```
tar  -cf /dev/fd0 /home/pepe
```

y si quisiéramos recuperar un fichero (fichero.doc) deberíamos ejecutar:

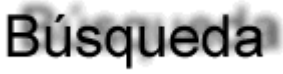
```
tar  -xvf /dev/fd0 /home/pepe/fichero.doc
```

Para los sistemas ATT, aunque está prácticamente generalizado, existe una versión de ***tar*** que es ***cpio***. Se puede mirar ***man cpio*** para los detalles de un sistema particular.

Hasta ahora hemos copiado ficheros empaquetados sin ningún tipo de compresión. Si queremos comprimir−expandir un fichero podremos utilizar la pareja de comandos ***compress***−***uncompress***, que es la utilidad más común en todos los sistemas UNIX. Si ***compress*** puede proceder a la compresión del fichero especificado (si no se especifica nada, se tomará la entrada y salida estándar [esto estará indicado para utilizar canalización]), creará un fichero con el mismo nombre al que se le ha añadido la extensión "***.Z***". Esto se suele utilizar mucho con ficheros "***.tar***". Utilizando la opción –r podemos hacer una compresión de un directorio de forma recursiva.

En sistemas GNU como LINUX, y actualmente en la mayoría de sistemas, existe una pareja de comandos que hace mejor esta labor, son los comandos ***gzip*** y ***gunzip***. En este caso, la extensión de los ficheros comprimidos será "***.gz***". Con ellos obtendremos en un poco más de tiempo que ***compress*** reducciones de tamaño más grandes, todo lo contrario que ocurre con la utilizad ***pack*** que es más rápida pero obtiene poca compresión, además de que prácticamente está obsoleta.

7.7. Búsqueda



Para acabar este capítulo, revisaremos brevemente algunos comandos que nos pueden servir para localizar ficheros o directorios dentro de un sistema de ficheros general.

- El primer método y más directo es usar el comando ***whereis***, que nos servirá para localizar ficheros binarios –b, fuente –s o en páginas de manual –m. Si no tenemos este comando en nuestro sistema siempre podemos utilizar la canalización clásica: `ls -l / | grep fichero`.
- Otra forma más sofisticada de buscar ficheros es usando el comando GNU ***locate*** o su versión segura ***slocate***. Este comando creará un índice asociado a una base de datos para hacer que las búsquedas sean mucho más rápidas, si queremos crear esa base de datos desde el directorio raíz deberemos usar la opción –u. Una vez construida, sólo le deberemos dar la palabra a buscar y el comando nos dará donde aparece la misma. Como se observa, este comando esta más orientado a buscar patrones de nombres de ficheros que ficheros en si, ya que para esta última labor ya tenemos el ***whereis***.
- Por último tenemos la orden ***find*** (vista anteriormente) que busca desde un directorio indicado, opcionalmente como primer argumento, los ficheros que nosotros le pedimos buscar de acuerdo a una condición expresada como segundo argumento opcional que comenzará usualmente con apóstrofes. Esta condición puede estar compuesta de opciones, preguntas y acciones separadas por operadores. Como preguntas se aceptan multitud de opciones como el tipo de fichero, la hora o minutos del último acceso, tipos de permiso, patrón de nombre, etc.

Por último, siempre podremos pedir al sistema información sobre “algo” con:

- ***man*** para información sobre un comando o llamada que esté en el manual.
- ***whatis***. Busca en la base de datos de comandos y palabras especiales para ofrecer información de ellos en pantalla.
- ***apropos***. Mismo propósito que el anterior.