Administración básica de **GNU/Linux**

José Angel de Bustos Pérez
<jadebustos@augcyl.org>
<joseangel.bustos@hispalinux.es>

II

Prefacio

Este manual fue concebido para una conferencia sobre "Administración básica de **GNU/Linux**". Soy consciente de que faltan temas de gran importancia, como el nucleo de **GNU/Linux**.

He decidido incluir los temas más "*esenciales*", por asi decirlo, que deben ser conocidos por todos aquellos que tengan que administrar una máquina **GNU/Linux**, especialmente máquinas que son utilizadas como estaciones de trabajo por varias personas.

GNU/Linux fue concebido como sistema operativo multiusuario y de red y es imposible tratar todos y cada uno de los temas relacionados con la interoperativilidad de **GNU/Linux**, via red, con otros sistemas operativos en una conferencia.

Todos los ejemplos de este manual estan realizados con Red Hat Linux, una de las distribuciones de **GNU/Linux** más difundidas. Posiblemente en otras distribuciones lo aquí expuesto cambie, sobre todo en lo concerniente al arranque y los niveles de ejecución. La idea es dar una visión general de como funcionan las cosas en **GNU/Linux** y que cada cual lo adapte a su sistema **GNU/Linux**.

Tampoco se han expuesto con todo lujo de detalles todos y cada uno de los comandos disponibles en **GNU/Linux**, ni tampoco todos y cada uno de los parámetros de dichos comandos. Para un estudio en detalle no hay nada como las páginas `man` y las páginas `info` y ponerse en una terminal a cacharrear.

Como no podia ser de otra forma este manual es libre y gratuito y se suministra tal cual, sin ninguna garantía tanto por parte de su autor como de **AUGCyL**. Lo único que puede garantizar este manual es el interés tanto del autor como de **AUGCyL** en promocionar el uso del software libre y la de facilitar una guia de administración a todos aquellos que deciden empezar a utilizar un sistema operativo desarrollado desde sus comienzos pensando en la fiabilidad y en la interoperativilidad con otras máquinas.

Índice general

Capítulo 1

Usuarios y permisos en GNU/Linux

GNU/Linux es un sistema operativo multitarea y multiusuario. Es por ello que en un sistema de este tipo tienen que convivir a la vez muchos usuarios, tienen que compartir los recursos del sistema ...

Cada usuario tiene sus archivos donde guarda sus datos, trabajo, música, ... y necesita que sus datos no puedan ser borrados por otro usuario y, en la mayoría de los casos, que no sean accedidos por otros usuarios.

1.1. El superusuario o `root`

Dentro de los sistemas `UNIX` existe un usuario "*especial*" que es quien se encarga de poner orden entre el resto de usuarios. Este usuario recibe el nombre de `root` y tiene acceso a la totalidad del sistema.

Este usuario es el encargado de realizar o delegar todas las tareas de mantenimiento y/o administración del sistema.

Los usuarios normales no pueden cambiar, ni modificar las configuraciones del sistema o de las aplicaciones. Normalmente las aplicaciones en este tipo de sistemas tienen una configuración, pudiendo el usuario adaptar dicha configuración a sus necesidades, gustos o manias. Pero estos cambios no afectan al resto de usuarios.

El usar esta cuenta es peligrosísimo ya que este usuario puede borrar cualquier parte del sistema y si se borra algo vital para el funcionamiento del sistema se puede llegar a inutilizar por completo el sistema, perdidas irrecuperables de datos, ...

Es practica habitual el no utilizar esta cuenta salvo para tareas específicas que no puedan ser realizadas con otra cuenta, por lo que pudiera pasar.

1.2. Grupos de usuarios

Para organizar todo esto en **GNU/Linux** se organizan los usuarios en grupos. Un grupo no es nada más que un conjunto de usuarios relacionados entre sí por la tarea que desempeñan dentro de la máquina.

Supongamos que tenemos una empresa y una máquina **GNU/Linux**, entonces podríamos tener los siguientes grupos:

adm que englobaría a aquellos usuarios que se dedican a la administración de la máquina.

contables que englobaría a aquellos usuarios que se dedican a llevar la contabilidad de nuestra empresa.

programadores que englobaría a aquellos usuarios que se dedican a tareas de programación.

...

Podemos encontrar la lista de grupos de un sistema en `/etc/group`. Además es muy frecuente encontrar usuarios del tipo:

apache que es un grupo especial para demonios o usuarios que van a administrar el servidor Apache.

mysql que es un grupo especial para demonios o usuarios que van a administrar el servidor de bases de datos MySQL.

1.2.1. Estructura de `/etc/group`

Como ya hemos dicho en este archivo se encuentran presentes todos los grupos definidos en el sistema. Si lo listamos tendremos entradas de este tipo:

```
users:x:100:tux,pepito
```

La estructura de este archivo son varios campos separados por ":".

1. En el primer campo tenemos el nombre del grupo.
2. En el segundo campo tenemos el password para el grupo. Normalmente contiene una "`x`" ya que no se usa habitualmente.
3. En el tercer campo tenemos el `GID` que es el número de identificación del grupo. **GNU/Linux** conoce el grupo de cada usuario por este campo y no por el nombre del grupo.
4. En el cuarto campo tenemos todos los usuarios que pertenecen a dicho grupo separados por comas.

Si listamos el fichero `/etc/group` veremos:

```
-rw-r--r-- 1 root root 537 feb 26 17:32 /etc/group
```

Nos indica que el fichero pertenece al usuario `root` que a su vez pertenece al grupo `root` y que tiene permisos de lectura y escritura para el propietario y unicamente de lectura para el resto de usuarios del grupo del propietario y resto de usuarios.

Por ejemplo si listamos un fichero cualquiera:

```
-rwxr-xr-x 1 jose users 362 mar 18 14:14 Adm-Basica.tex
```

Podemos ver los permisos que tiene el fichero, además que es propiedad del usuario `jose` y que pertenece al grupo `users`.

Si quitamos los permisos de lectura para todos usuarios excepto para el propietario de `/etc/group` al listar el fichero veremos:

```
-rwxr-xr-x 1 jose 100 362 mar 18 14:14 Adm-Basica.tex
```

donde vemos que pertenece al usuario `jose` y que el grupo del usuario `jose` es 100. Esto es debido a que no tenemos permiso de lectura del archivo `/etc/group` que es de donde **GNU/Linux** relaciona el grupo con GID 100 con el nombre que tiene dicho grupo (`users`).

1.2.2. Añadir grupos al sistema

Unicamente el `root` debería poder añadir grupos al sistema, pero puede dar privilegios a otros usuarios para hacerlo.

El comando que se utiliza para añadir grupos al sistema es `groupadd`:

```
[root@localhost root]# groupadd augcyl
```

De esta forma añadiriamos el grupo `augcyl` al sistema. Por defecto el sistema establece como `GID` números a partir de 500 ya que los números menores que 500 estan reservados para cuentas especiales del sistema.

Si queremos especificar que el grupo `augcyl` tenga como `GID` un número determinado:

```
[root@localhost root]# groupadd -g 1000 augcyl
```

en este caso debemos asegurarnos de que no existe ningún grupo con `GID` 1000.

1.2.3. Modificar los grupos del sistema

Para modificar los grupos que existen en el sistema podemos utilizar el comando `groupmod`. Su forma de uso es fácil con lo cual si tienes interés consulta la página del manual (`man groupmod`).

1.2.4. Borrar grupos del sistema

Unicamente el `root` debería poder borrar grupos del sistema, pero puede dar privilegios a otros usuarios para hacerlo.

El comando que se utiliza para borrar grupos del sistema es `groupdel`:

```
[root@localhost root]# groupdel augcyl
```

De esta forma borraríamos el grupo `augcyl` del sistema.

1.3. Gestión de usuarios

Como en **GNU/Linux** conviven muchos usuarios hay que tenerlos organizados de alguna forma para que cada uno tenga su espacio y no interfiera en el trabajo del resto de usuarios.

1.3.1. Como accede un usuario al ordenador

Para acceder al ordenador cada usuario lo hace mediante un `login` o nombre de usuario y un `password` o contraseña. El `login` de cada usuario se puede hacer público para que el resto de usuarios de la máquina puedan interactuar con él, pero el `password` se mantiene oculto ya que permite el acceso a la máquina como dicho usuario y da acceso a todos sus ficheros.

1.3.2. Zona de disco reservada para cada usuario

La jerarquía más utilizada es situar los archivos personales de cada usuario dentro de `/home`

```
[root@localhost root]# ls -l /home
total 8
drwx------    3 icarus   users        4096 mar 18 16:04 icarus
drwxr-xr-x   25 jose     program      4096 mar 18 23:22 jose
```

otra forma de situarlos es crear dentro de `/home` un directorio para cada grupo y dentro de ese directorio crear los directorios personales de cada usuario:

```
[root@localhost root]# ls -l /home
total 8
drwxr-xr-x    3 root   root        4096 mar 18 12:00 users
drwxr-xr-x    3 root   root        4096 mar 18 12:22 program
```

dentro de su directorio personal cada usuario guarda sus ficheros. Los administradores suelen establecer una cuota[1] de uso de disco para evitar que un usuario monopolice el disco duro.

El directorio de trabajo de cada usuario esta en la variable de entorno `$HOME`:

[1] Apartado 6.11 en la página 111.

```
[root@localhost root]# $HOME
bash: /root: is a directory
[root@localhost root]#
```

A este directorio también se le conoce como ∼.

1.3.3. Estructura de `/etc/passwd`

En este fichero se guarda la información que tiene el sistema de los usuarios. Debido a problemas de seguridad eso se cambió y se utiliza de forma conjunta con el fichero `/etc/shadow`.

Una entrada típica del fichero `/etc/passwd`:

```
augcyl:x:501:100:AUGCyL:/home/jose:/bin/bash
```

La estructura de este archivo son varios campos separados por ":":

1. En el primer campo tenemos el `login` del usuario.
2. En el segundo campo originariamente teniamos un hashing de la contraseña, podríamos decir que es una especie de encriptación de la contraseña. Normalmente se habilita el "*shadowing*" de contraseñas, en este caso la contraseña estará almacenada en el fichero `/etc/shadow` y en este campo habrá un carácter especial, normalmente "`x`".
3. En el tercer campo tenemos el `UID` o número de identificación de usuario.
4. En el cuarto campo tenemos el `GID` o número de identificación del grupo.
5. En el quinto campo suele haber información sobre el usuario, nombre, teléfono, ... , lo más normal es que unicamente esté el nombre.
6. En el sexto campo esta el directorio de trabajo del usuario.
7. En el septimo campo esta la `shell` que utiliza el usuario. En este campo se pone el comando que queremos que se ejecute cada vez que entremos en sesión. Lo normal es indicar una `shell` o interprete de comandos para que podamos interactuar con el sistema operativo.

Podemos deshabilitar temporalmente una cuenta de un usuario sin eliminarlo del sistema. Para ello bastaría con añadir el carácter "*" como primer carácter en el segundo campo y dicho usuario no podría acceder al sistema hasta que eliminasemos dicho carácter.

Imaginemos que tenemos montado un servidor de correo electrónico y necesitamos tener instalado el servidor de `telnet` para administrarlo de forma remota. Entonces todas las personas que tengan una cuenta de correo en el servidor podrán hacer un `telnet` y utilizar la máquina, una solución es colocar en el último campo de `/etc/passwd` el siguiente comando `/bin/exit`. Esto hará que cuando intenten hacer un `telnet` al entrar en sesión se ejecute el comando `exit` en lugar de una `shell` con lo cual se cerrará la sesión.

1.3.4. Añadir usuarios al sistema

Unicamente el `root` debería poder añadir usuarios al sistema, pero puede dar privilegios a otros usuarios para hacerlo.

El comando que se utiliza para añadir usuarios al sistema es `useradd`:

```
[root@localhost root]# useradd -g users -G users,program augcyl
```

Esto añadiría al sistema el usuario `augcyl` y tendría como grupo por defecto `users` y pertenecería a los grupos `users` y `program`.

Además de esto se pueden establecer más "cosas" a la hora de crear un usuario utilizando las siguientes flags:

-d establece el directorio de trabajo del usuario.

-m crea el directorio de trabajo del usuario si no existe.

-e establece la caducidad de la cuenta.

-n en sistemas Red Hat se crea por defecto un grupo con el nombre del usuario. Con este flag se evita el crear ese grupo[2].

-s indica la `shell` que utilizara el usuario.

-u establece el `UID` del usuario. Por defecto se empiezan a asignar a partir del 500 de forma correlativa (en sistemas Red Hat, en Debian se empieza a partir de 1.000).

[2]Si lo creamos como en el ejemplo anterior también se evita.

Cuando creamos un usuario se copian en su directorio de trabajo los ficheros presentes en `/etc/skel`, esto nos permite una mayor comodidad y rápidez a la hora de crear cuentas de usuario.

1.3.5. Establecer el password del usuario

Esta tarea está resevada para el `root` o a aquellas personas a las que se les haya dado privilegios para hacerlo y se realiza con el comando `passwd`:

```
[root@localhost root]# passwd augcyl
Changing password por user augcyl
New password:
BAD PASSWORD: it is based on a dictionary word
Retype new password:
password: all authentication tokens updated successfully
[root@localhost root]#
```

1.3.6. Borrar usuarios del sistema

Unicamente el `root` debería poder borrar usuarios del sistema, pero puede dar privilegios a otros usuarios para hacerlo.

El comando que se utiliza para borrar usuarios del sistema es `userdel`:

```
[root@localhost root]# userdel bill
```

De esta forma borraríamos del sistema la cuenta `bill`, pero no borrariamos su directorio de trabajo. Si quisieramos borrar su directorio de trabajo deberíamos especificar el flag `-r`.

Si un usuario está conectado no podremos borrar su cuenta, para ello se puede anular temporalmente su cuenta utilizando un `*` y después matar todos sus procesos[3]. A continuación podemos borrar su cuenta tranquilamente ya que no podrá volver a entrar en el sistema.

1.3.7. Modificar la cuenta de un usuario ya existente en el sistema

Si queremos modificar los datos de la cuenta de un usuario que ya existe en el sistema, por ejemplo cambiar su directorio de trabajo, su nombre de

[3]Ya veremos más adelante esto.

acceso al sistema (`login`), caducidad de la clave de acceso, ... podemos utilizar el comando `usermod`. Su forma de uso es fácil con lo cual si tienes interés consulta la página del manual (`man usermod`).

1.3.8. El comando `id`

Este comando se utiliza para obtener información sobre el `UID` y `GID` de los usuarios:

```
[root@localhost root]# id
id=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),
           6(disk),10(wheel)
[root@localhost root]#
```

1.4. Permisos en GNU/Linux

Para evitar que otros usuarios accedan a nuestros archivos **GNU/Linux** al igual que otros sistemas `UNIX` utiliza permisos y cada usuario puede acceder unicamente a aquellos ficheros a los que tiene concedido el acceso.

Esto es muy importante ya que para los sistemas `UNIX` todo son ficheros, el monitor, la impresora, el ratón y todo tipo de dispositivos son ficheros. Debido a esta característica un tanto "rara" tenemos un sistema altamente configurable.

Los permisos de un fichero se almacenan como un entero de doce bits y se dividen en ternas:

- La terna más significativa se utiliza para especificar unos permisos especiales que son los SUID, SGID y el Sticky bit.
- La terna siguiente se utiliza para especificar los permisos del propietario del fichero.
- La terna siguiente se utiliza para especificar los permisos del grupo del propietario del fichero.
- La terna menos significativa se utiliza para especificar los permisos del resto de usuarios del sistema.

1.4.1. Tipos de permisos

Los permisos típicos que nos podemos encontrar son:

ejecución es denotado como "`x`".

escritura es denotado como "`w`".

lectura es denotado como "`r`".

Dependiendo de si estan aplicados a un fichero o a un directorio pueden tener un comportamiento un "poco" diferente.

Permiso de ejecución

fichero podemos ejecutar el fichero.

directorio podemos entrar en el directorio mediante `cd`.

Permiso de escritura

fichero podemos modificar el contenido del fichero.

directorio podemos modificar el contenido del directorio. Si no tenemos concedido este permiso no podremos crear ni borrar ficheros, pero podremos modificar aquellos ficheros en los que tengamos permiso de escritura.

Permiso de lectura

fichero podemos leer el contenido del fichero.

directorio podemos leer el contenido del directorio mediante `ls`.

1.4.2. Como comprobar los permisos de un fichero

Para comprobar los permisos que tiene un fichero basta con hacer un `ls -l`:

```
[root@localhost root]# ls -l
-rwxr-x---    1 jose     users         385 mar 19 11:55 Adm-Basica.tex
[root@localhost root]#
```

La linea que nos indica los permisos que tenemos sobre el fichero `Adm-Basica.tex` es:

```
-rwxr-x---
```

Esta linea esta dividida en cuatro apartados:

$$\underbrace{-}_{\text{(I)}} \quad \underbrace{rwx}_{\text{(II)}} \quad \underbrace{r-x}_{\text{(III)}} \quad \underbrace{---}_{\text{(IV)}}$$

donde:

(I) nos indica el tipo de fichero que es:

- **-** fichero normal.
- **d** directorio.
- **s** socket.
- **l** enlace.
- **c** dispositivo carácter (monitor,impresora).
- **b** dispositivo de bloques (discos).
- **p**

(II) nos indica los permisos que tiene el propietario del fichero.

(III) nos indica los permisos que tienen los usuarios que pertenecen al mismo grupo que el propietario.

(IV) nos indica los permisos que tienen el resto de usuarios.

1.4.3. Quien puede cambiar los permisos de los ficheros

Los permisos unicamente los pueden cambiar dos personas, que son el propietario del fichero y el `root`.

1.4.4. Como cambiar los permisos de los ficheros

Para cambiar los permisos de los ficheros se utiliza el comando `chmod`. Cuando utilizemos este comando hay que tener presente que unicamente afectará la los ficheros en el directorio que especifiquemos, no realizará cambios dentro de los directorios que haya en el directorio en el que actue. Si queremos que actue de una forma recursiva y cambie los permisos de todos los ficheros y directorios que haya en el directorio objetivo tendemos que utilizar el flag `-R`.

Cambiar permisos de forma intuitiva

Podemos utilizar `u` para referirnos a los permisos del propietario, `g` para referirnos al resto de usuarios del grupo del propietario[4], `o` para referirnos al resto de usuarios y `a` para referirmos a todos los usuarios.

Además utilizaremos `=` para establecer los permisos, `+` para añadir permisos a los ya existentes y `-` para eliminar permisos. Por ejemplo:

```
[root@localhost root]# ls -l Adm-Basica.tex
-rwxrwx---    1 jose     users         385 mar 19 11:55 Adm-Basica.tex
[root@localhost root]# chmod u=rw,g-w,o+r Adm-Basica.tex
[root@localhost root]# ls -l Adm-Basica.tex
-rw-r-xr--    1 jose     users         385 mar 19 11:55 Adm-Basica.tex
[root@localhost root]#
```

Si añadimos el flag `-v` nos informa del resultado.

Cambiar permisos en octal

La forma anterior es muy intuitiva, pero es un poco tediosa. Podemos establecer los permisos utilizando notación octal. ¿Por que en octal? pues es muy sencillo. Si establecemos que 1 indica que tenemos concedido un permiso y 0 que no lo tenemos, y nos fijamos que tanto para el usuario, grupo del usuario y resto de usuarios utilizamos tres caracteres para representar sus permisos tendremos que con tres dígitos en binario podemos representar ocho números diferentes.

[4] En el caso de que el propietario pertenezca a varios grupos hará referencia al grupo al que pertenezca el fichero.

Es muy sencillo utilizar esta forma, de hecho la mayoría de la gente lo hace de esta forma:

	Propietario	Grupo	Resto
Permisos	rwx	rwx	rwx
Binario	111	101	100
Octal	7	5	4

Luego para establecer estos permisos se haría:

```
[root@localhost root]# chmod 754 Adm-Basica.tex
[root@localhost root]# ls -l Adm-Basica.tex
-rwxr-xr--    1 jose     users         385 mar 19 11:55 Adm-Basica.tex
[root@localhost root]#
```

1.4.5. Permisos por defecto

Es lógico preguntarse que permisos se ponen por defecto cuando creamos un fichero o un directorio. Se puede ver ejecutando el comando `umask`:

```
[root@localhost root]# umask
022
[root@localhost root]#
```

Si utilizamos el flag `-S` lo entenderemos mejor:

```
[root@localhost root]# umask -S
u=rwx,g=rx,o=rx
[root@localhost root]#
```

Estableciendo la máscara de forma intuitiva

```
[root@localhost root]# umask u=rwx,o-rwx,g-rwx
[root@localhost root]# umask -S
u=rwx,g=,o=
[root@localhost root]#
```

Estableciendo la máscara en octal

Procedemos de una forma parecida a `chmod` sólo que ponemos a 1 los permisos que queremos quitar:

	Usuario	Grupo	Resto
Permisos	rwx	rwx	rwx
Binario	000	010	110
Octal	0	2	6

1.4.6. Cambio del grupo del usuario, el comando `newgrp`

Muchas veces un usuario pertenece a varios grupos, por ejemplo a los grupos `contable` y `program`. Puede darse el caso en el que un usuario se dedique a realizar tareas adminstrativas, como por ejemplo contabilidades, pero que de vez en cuando necesite crear y compilar un programa utilizando uno de los muchos lenguajes de programación existentes para **GNU/Linux**.

Es altamente recomendable el dar permisos de ejecución sobre los compiladores del sistema unicamente a las personas que necesiten usarlos, evitando de esta manera que nuestros usuarios se bajen un exploit, lo compilen y se dediquen a probarlo en nuestra máquina. También para evitar que un atacante que haya podido hacerse con una cuenta en el sistema tenga acceso a los compiladores, al menos le será más complicado ya que no todos los usuarios tienen privilegios para el uso de los compiladores.

Volviendo al caso anterior, como nuestro usuario se dedica principalmente a realizar contabilidades su grupo por defecto será `contable`, pero cuando necesite compilar un programa que haya creado para calcular la conversión de Pesetas a Euros no podrá compilar ya que el compilador que necesita utilizar unicamente tendrá permisos de ejecución para el dueño y para aquellos que pertenezcan a su grupo `program`.

Aunque nuestro usuario pertenece a ambos grupos no podrá, a priori, utilizar el compilador ya que su grupo por defecto es `contable`. Para ello necesitamos cambiar de grupo utilizando `newgrp`:

```
[bender@localhost bender]$ newgrp program
```

de esta forma el grupo del usuario `bender` será a partir de ese momento `program` y todos sus accesos serán en función de dicho grupo. Si el grupo `program` tuviera establecida una contraseña sería solicitada y dicha contraseña estaría almacenada en `/etc/group` o si se tiene activado el "*shadowing*" de contraseñas en `/etc/gshadow`.

1.4.7. El comando su

Este comando nos permite ejecutar una `shell` como otro usuario. Imaginemos que estamos en una sesión como el usuario `bender` y queremos abrir una sesión como usuario `lila` ya que `lila` tiene unos privilegios que no tiene `bender`:

```
[bender@localhost bender]$ su lila
Password:
[lila@localhost bender]$ whoami
lila
[lila@localhost bender]$ pwd
/home/bender
[lila@localhost bender]$
```

utilizando la misma conexión hemos asumido la identidad de `lila` y podemos acceder a los recursos del sistema que tenga acceso `lila`. Una vez que hayamos terminado la tarea podemos volver a ser `bender` pulsando `Ctrl + D` o tecleando `exit`:

```
[lila@localhost bender]$ exit
[bender@localhost bender]$ whoami
bender
[bender@localhost bender]$
```

Podemos observar que al ejecutar `su` no cambia el directorio en el que estamos. Aunque hemos asumido la identidad de `lila` no hemos entrado en sesión como lo haría `lila` haciendo un `telnet` ya que no hemos cargado sus archivos de personalización, `profiles`, para ello necesitamos hacer `su` anteponiendo `-` al nombre de usuario:

```
[bender@localhost bender]$ su - lila
Password:
[lila@localhost lila]$ pwd
/home/lila
[lila@localhost lila]$
```

de esta forma habremos abierto una `shell` de la misma forma que si hubieramos hecho un `telnet`.

En caso de no pasarle ningún argumento a `su` se entenderá que se quiere abrir una sesión como `root`, sin cargar los archivos de personalización. Si el único argumento que se le pasa es - se entenderá que lo que se quiere es abrir una sesión de `root` cargando los archivos de personalización.

También podemos ejecutar comandos como otros usuarios ejecutando `su`, siempre y cuando conozcamos su contraseña:

```
[bender@localhost bender]$ su lila -c ‘‘rm -Rf /home/lila’’
Password:
[bender@localhost bender]$
```

1.4.8. ¿A que grupo pertenezco?

Podemos comprobar a que grupo pertenecemos utilizando los comandos `groups` y `id`:

```
[bender@localhost bender]$ groups bender
bender : robot mafioso borrachin
[bender@localhost bender]$
```

Nos indica que el usuario `bender` pertenece a los grupos `robot`, `mafioso` y `borrachin`. El primer grupo que aparece, `robot`, es el grupo según el cual se estan aplicando las restricciones de acceso. Si necesitaramos utilizar alguno de los privilegios que tienen los otros grupos necesitariamos utilizar el comando `newgrp`:

```
[bender@localhost bender]$ newgrp mafioso
[bender@localhost bender]$ groups bender
bender : mafioso robot borrachin
[bender@localhost bender]$
```

Otra forma sería utilizar el comando `id`[5]:

```
[bender@localhost bender]$ id -Gn bender
robot mafioso borrachin
[bender@localhost bender]$
```

[5] En el apartado 1.3.8 en la página 17.

1.5. El permiso SUID

Hay veces que es necesario que un programa se ejecute con los privilegios que tiene dentro del sistema su propietario y no del usuario que lo esta ejecutando.

Un ejemplo de esto es el comando `passwd`, el cual necesita tener privilegios de `root` ya que necesita acceder a ciertos ficheros a los que sólo el usuario `root` puede acceder (`/etc/passwd`, `/etc/shadow`). Este comando permite cambiar los passwords de acceso al sistema a los usuarios. Es práctica habitual, aunque muchas veces no recomendable, el permitir a los usuarios cambiar su contraseña de acceso al sistema.
La forma en la que trabaja este comando es muy sencilla:

- Si no se le especifica un usuario cambia la contraseña del usuario que lo esta ejecutando:

  ```
  [bender@localhost bender]$ passwd
  New password:
  Retype new password:
  passwd: all authentication tokens updated successfully
  [bender@localhost bender]$
  ```

  Con esto el usuario `bender` ha cambiado su contraseña de acceso al sistema.

- Si se le especifica el nombre de un usuario cambia la contraseña del usuario especificado:

  ```
  [bender@localhost bender]$ passwd root
  passwd: Only root can specify a username
  [bender@localhost bender]$
  ```

  El usuario `bender` pensó que si podía cambiar su contraseña podría cambiar la de cualquier otro usuario en el sistema, y ya puestos decidió cambiar la contraseña del `root`. Que cacho-perro!!!

Ya hemos visto que el único usuario en el sistema que puede cambiar las contraseñas de los demás es el `root`. Pero aún existe un problema del que no hemos hablado. Nosotros podemos cambiar nuestra contraseña, pero los ficheros en los que se almacena son `/etc/passwd`, en un sistema sin "*shadowing*" de contraseñas, y `/etc/shadow`, en un sistema con "*shadowing*" de contraseñas, tienen los siguientes permisos:

```
[bender@localhost bender]$ ls -l /etc/passwd
-rw-r--r--    1 root     root         1237 abr  4 13:58 /etc/passwd
[bender@localhost bender]$ ls -l /etc/shadow
-r--------    1 root     root          854 abr  4 13:58 /etc/shadow
```

podemos ver que no tenemos permiso de escritura en ninguno de los ficheros, sólo el `root` puede hacerlo. Entonces ¿como demonios el sistema se va a acordar de nuestra nueva contraseña cuando hayamos terminado nuestra sesión?, ya que al no poder escribir en los ficheros en los que se almacenan las contraseñas una vez terminada nuestra sesión la contraseña se perderá.

Si ejecutaramos el comando `passwd` como usuario `bender` pasaría lo siguiente:

```
[bender@localhost bender]$ passwd
passwd: Authentication token manipulation error
[bender@localhost bender]$ ls -l /usr/bin/passwd
-r-x--x--x    1 root     root        13476 ago  7  2001 /usr/bin/passwd
```

Los permisos con los que se está ejecutando son los permisos del usuario `bender`, el cual no tiene privilegios suficientes para escribir en los archivos de almacenamiento de claves.

Para ello es necesario contar con un permiso especial, el SUID, que lo que hace es que cuando se ejecuta un archivo que tiene este permiso activado se ejecuta con los privilegios del propietario y no de la persona que lo esta ejecutando.

Veamos como hacer que el comando `passwd` permita a los usuarios cambiar sus contraseñas:

```
[root@localhost root]# ls -l /usr/bin/passwd
-r-x--x--x    1 root     root        13476 ago  7  2001 /usr/bin/passwd
[root@localhost root]# chmod 4511 /usr/bin/passwd
[root@localhost root]# ls -l /usr/bin/passwd
-r-s--x--x    1 root     root        13476 ago  7  2001 /usr/bin/passwd
[root@localhost root]#
```

Si nos fijamos detenidamente en los permisos del propietario veremos que en lugar de la "`x`" correspondiente al permiso de ejecución tenemos una "`s`". Esta "`s`" nos indica que dicho fichero tiene activado el permiso SUID. Cada vez que se ejecute será ejecutado con los privilegios de su propietario, independientemente de quien lo ejecute, de esto viene su nombre **S**et **U**ser **ID**.

1.5.1. ¿Como activar el permiso SUID?

Para activar este permiso tenemos que conceder permisos de ejecución a aquellos que vayan a utilizar el fichero y anteponer un 4 a la terna de números, notación octal o absoluta, que determinan los permisos de un fichero. Por ejemplo, supongamos que a un fichero le queremos dar los permisos 755 y ademas el permiso SUID entonces tendremos que especificar con `chmod` 4755:

4 activa el permiso SUID.

7 concede todos los permisos al propietario.

5 concede todos los permisos, excepto el de escritura, al grupo del propietario.

5 concede todos los permisos, excepto el de escritura, al resto de usuarios.

En caso de no conceder permisos de ejecución a los usuarios del grupo del propietario o al resto de usuarios estos no podrán ejecutar el fichero con permisos SUID. Sólo podrán beneficiarse del permiso SUID aquellos que tengan permiso para ejecutar el fichero.

Otra cosa a tener en cuenta es que este permiso funciona **UNICAMENTE** con ficheros binarios y no con scripts[6].

1.5.2. El permiso SUID y los directorios

Este permiso no tiene ningún efecto en directorios.

[6]Salvo scripts de perl.

1.6. El permiso SGID

Este permiso es igual que el SUID salvo que durante la ejecución se ejecuta con los privilegios del grupo del propietario.

Para comprobar si un fichero tiene activado este permiso:

```
[bender@localhost bender]$  find /usr/bin -perm +2000 -exec ls -l {} \;
-rwxr-sr-x    1 root     mail        12500 jun 30  2001 /usr/bin/lockfile
-rwxr-sr-x    1 root     slocate     25020 jun 25  2001 /usr/bin/slocate
-r-xr-sr-x    1 root     tty          6444 ago 29  2001 /usr/bin/wall
-rwxr-sr-x    1 root     tty          8744 ago 27  2001 /usr/bin/write
-rwxr-sr-x    1 root     root        54752 sep  8  2001 /usr/bin/kdesud
[bender@localhost bender]$
```

Con esto lo que hemos hecho ha sido buscar en el directorio `/usr/bin` todos los ficheros que tienen activado el permiso SGID y hacer sobre ellos un "`ls -l`". Podemos ver que sobre el permiso de ejecución del grupo del propietario tenemos una "`s`" en lugar de la "`x`", eso nos indica que dicho fichero tiene activado el permiso SGID. Cada vez que se ejecute será ejecutado con los privilegios del grupo de su propietario, independientemente de quien lo ejecute, de esto viene su nombre **S**et **G**roup **ID**.

1.6.1. ¿Como activar el permiso SGID?

Para activar este permiso tenemos que conceder permisos de ejecución a aquellos que vayan a utilizar el fichero y anteponer un 2 a la terna de números, notación octal o absoluta, que determinan los permisos de un fichero. Por ejemplo, supongamos que a un fichero le queremos dar los permisos 755 y ademas el permiso SGID entonces tendremos que especificar con `chmod` 2755:

2 activa el permiso SGID.

7 concede todos los permisos al propietario.

5 concede todos los permisos, excepto el de escritura, al grupo del propietario.

5 concede todos los permisos, excepto el de escritura, al resto de usuarios.

Otra cosa a tener en cuenta es que este permiso funciona **UNICAMENTE** con ficheros binarios y no con scripts[7].

[7] Salvo scripts de perl.

1.6.2. El permiso SGID y los directorios

Cuando un directorio tiene activado el permiso SGID todos los archivos creados dentro del directorio perteneceran al grupo del propietario de dicho directorio.

Esto es útil cuando se trabaja en directorios compartidos. Normalmente un determinado grupo de usuarios pueden estar realizando un proyecto y trabajar en un directorio común. Para evitar problemas con los permisos si se activa el permiso SGID en dicho directorio todos los archivos creados dentro de el pertenecerán al mismo grupo. Como todos los usuarios pertenecientes al proyecto tendrán un grupo en común todos tendrán los mismos privilegios sobre los archivos.

1.7. El Sticky bit

Como ya hemos visto los permisos de un archivo se representan por doce bits. Los tres bits más significativos representan unos permisos especiales:

- El bit más significativo es el permiso SUID o s-bit de usuario.
- El segundo bit más significativo es el permiso GUID o s-bit de grupo.
- El tercer bit más significativo es el Sticky bit, también conocido como bit pegajoso o de adhesión.

Cuando este permiso está concedido el programa permancece en memoria aún después de que haya terminado su ejecución. Si ejecutamos otra vez el programa lograremos que se carge antés ya que ya está en memoria, pero tiene un coste y es la reducción de la memoria disponible. No combiene abusar mucho de este permiso ya que podemos saturar la memoria de nuestro equipo.

Para comprobar si un fichero tiene ese permiso activado:

```
[bender@localhost bender]$ find /usr -perm +1000 -exec ls -l {} \;
total 4
drwxr-xr-t    2 root     root         4096 ene 13 20:07 ec
total 4
-rw-r--r--    1 root     root         3148 ene 13 20:07 ecrm1000.tfm
total 4
drwxr-xr-t    3 root     root         4096 ene 13 20:07 ljfour
total 4
drwxr-xr-t    3 root     root         4096 ene 13 20:07 jknappen
total 4
drwxr-xr-t    2 root     root         4096 ene 13 20:07 ec
total 24
-rw-r--r--    1 root     root        23964 ene 13 20:07 ecrm1000.600pk
[bender@localhost bender]$
```

Con esto lo que hemos hecho ha sido buscar en el directorio `/usr` todos los ficheros que tienen activado el Sticky bit y hacer sobre ellos un "`ls -l`". Podemos ver que sobre el permiso de ejecución del resto de usuarios tenemos una "`t`" en lugar de la "`x`", eso nos indica que dicho fichero tiene activado el Sticky bit. Hay que tener controlados estos ficheros ya que una vez terminada la ejecución de uno de estos programas quedará en memoria ocupando espacio y si un fichero demasiado grande es ejecutado o varios podemos colapsar la memoria del sistema (podríamos ocasionar una denegación de servicio).

1.7.1. ¿Como activar el Sticky bit?

Para activar este permiso tenemos que anteponer un 1 a la terna de números, notación octal o absoluta, que determinan los permisos de un fichero. Por ejemplo, supongamos que a un fichero le queremos dar los permisos 755 y ademas activar el Sticky bit entonces tendremos que especificar con `chmod` 1755:

1 activa el Sticky bit.

7 concede todos los permisos al propietario.

5 concede todos los permisos, excepto el de escritura, al grupo del propietario.

5 concede todos los permisos, excepto el de escritura, al resto de usuarios.

1.7.2. El Sticky bit y los directorios

Este bit se utiliza sobre directorios para tener una mayor seguridad sobre los ficheros contenidos en él. Si un directorio tiene activado este bit no importa los permisos que tengan los ficheros en el contenidos, los unicos usuarios que podrán borrar o renombrar ficheros serán el propietario del fichero, el propietario del directorio o el `root`.

Esta característica permite que dentro de un directorio todos los usuarios, con acceso, puedan cambiar el contenido de los ficheros. Pero que ninguno, excepto los anteriormente citados, pueda borrar o cambiar el nombre de los ficheros.

1.8. Atributos de los ficheros

Hemos visto que el usuario `root` es el único que puede acceder a todo el sistema sin restricciones. Esto puede ser un peligro ya que puede borrar cualquier archivo del sistema por error.

Los archivos además de permisos también tienen atributos. Mediante estos atributos podemos lograr que nadie, ni siquiera el `root`, borre archivos por equivocación.

1.8.1. El comando `chattr`

Este comando sirve para cambiar los atributos de los archivos en el sistema de ficheros ext2. Algunos de los atributos que podemos establecer con este comando son:

no modificable el fichero no se puede modificar, renombrar, ni hacer enlaces a un fichero con este atributo activado. Este atributo sólo puede ser activado/desactivado por el `root`. Flag "*i*".

añadir unicamente se puede añadir información al fichero. Sólo se permite abrir el fichero en modo "*append*". Este atributo sólo puede ser activado/desactivado por el `root`. Flag "`a`".

borrado seguro antes de borrar el fichero lo sobreescribe con ceros y lo guarda en disco. Flag "`s`".

Para añadir atributos utilizamos "+", para eliminar atributos utilizamos "-" y para establecer atributos utilizamos "=". Por ejemplo:

```
[root@localhost root]# chattr +i /etc/hosts.deny
```

Añadiría el atributo de "*no modificable*" al archivo `/etc/hosts.deny`.

```
[root@localhost root]# chattr -a /etc/profile
```

Quitaría el atributo de "*añadir*" al archivo `/etc/profile`.

```
[root@localhost root]# chattr =s /etc/fstab
```

Establecería que el archivo `/etc/fstab` sólo tendría el atributo de "*borrado seguro*".

1.8.2. El comando `lsattr`

Este comando sirve para ver los atributos de los archivos en el sistema de ficheros ext2.

```
[bender@localhost bender]$ lsattr agenda
---i--------- agenda
[bender@localhost bender]$
```

Vemos que el fichero `agenda` tiene activado el atributo de "*no modificable*".

Capítulo 2

Ayuda en GNU/Linux

En los sistemas `UNIX` existe una gran cantidad de comandos y cada uno de ellos tiene unas opciones determinadas, por lo cual es "*imposible*" conocer todos y cada uno de esos comandos al dedillo. En estos sistemas podemos encontrar ayuda sobre los comandos y como utilizarlos.

2.1. Páginas `man`

Las páginas `man` es la forma más rápida y sencilla de obtener ayuda. Para ello basta con pasar como argumento a `man` el comando sobre el que queremos la ayuda.

```
[root@localhost root]# man man
```

La ayuda esta estructurada en varias secciones, cada una de ellas numerada:

1. Comandos generales del sistema.
2. Llamadas al sistema.
3. Rutinas de las bibliotecas del sistema.
4. Ficheros especiales (normalmente en `/dev`).
5. Formatos de los ficheros.
6. Juegos.
7. Macros.
8. Comandos de administración.

9. Kernel (no es estándar).

esta numeración puede variar de un sistema a otro.

Pudiera ocurrir que hubiera varias referencias a un comando, o que existiera un comando con el mismo nombre que un fichero de configuración. En este caso si quisieramos consultar la ayuda del fichero de configuración haríamos lo siguiente:

```
[root@localhost root]# man 5 crontab
```

Lo cual haría que se mostrará la información referente al fichero de configuración `crontab`. Pero si hacemos:

```
[root@localhost root]# man 1 crontab
```

Obtendríamos la información referente al comando `crontab`.

2.1.1. Las páginas `man` en el sistema X-Window

También es posible consultar las páginas `man` en el sistema X-Window, para ello basta invocar `xman` desde una consola y podremos acceder a las páginas `man` en las X y navegar por ellas.

2.2. Uso de `info`

Otra forma de obtener ayuda es utilizando el comando `info`. Con este comando podemos "navegar" en la ayuda, saltando de un lado a otro. Para utilizarlo basta con ejecutar `info` o para obtener ayuda sobre `info` teclear `info info`.

Si queremos obtener ayuda sobre un comando:

```
[root@localhost root]# info comando
```

Si presionamos la tecla `return` sobre una linea que empieza con "*" accedemos a la ayuda sobre ese tema. Para salir de `info` en cualquier momento basta con pulsar "*q*".

2.3. Más información presente en el sistema

Es habitual que cuando se instalan programas, paquetes, se instale en el sistema información sobre dichos programas.

Lo más habitual es encontrar esa informacion en los siguientes directorios:

- `/usr/doc`
- `/usr/share/doc`

2.4. Busqueda de información sobre ficheros

Estos comandos nos ofrecen información sobre ficheros que estan presentes en el sistema. Muchas veces nos volvemos locos buscando algo y no sabemos como encontrarlo. En sitemas `UNIX` existe tal cantidad de comandos que es imposible conocerlos todos y conocer su ubicación u otros datos de interés.

GNU/Linux proporciona una serie de comandos de gran utilidad para estos menesteres.

2.4.1. El comando `file`

Este comando nos ofrece información sobre que tipo de fichero es un determinado fichero:

```
[jose@localhost adm]$ file Adm-Basica.tex Adm-Basica.pdf Adm-Basica.ps
Adm-Basica.tex: LaTeX 2e document text
Adm-Basica.pdf: PDF document, version 1.2
Adm-Basica.ps:  PostScript document text conforming at level 2.0
[jose@localhost adm]$
```

Aunque parezca que este comando funciona examinando la extensión del fichero, la realidad es bastante diferente:

```
[jose@localhost adm]$ file sorpresa.pdf
sorpresa.pdf: gzip compressed data, deflated,
              last modified: Mon Mar 25 13:35:07 2002, os: Unix
[jose@localhost adm]$
```

Aunque pudiera parecer que `sorpresa.pdf` es un fichero `pdf` en realidad son varios ficheros empaquetados con la herramienta `tar` y comprimidos después con `gzip`. Podemos ver como el comando `file` no se ha dejado engañar por la extensión y ha detectado el tipo correcto de archivo que es.

Para detectar que tipo de archivo es `file` busca un dato que existe en los ficheros llamado número mágico y le indica el tipo de archivo que es.

2.4.2. El comando `find`

Este comando es uno de los comandos más utiles que hay para localizar ficheros, ya que nos permite realizar busquedas de todo tipo, nombre, propietario, permisos, ...

Este comando tiene una gran cantidad de opciones, por lo cual sólo indicaremos algunas de las más utiles y las demás `man find` ;-).

La forma más sencilla de utilizarlo es:

```
[bender@localhost bender]$ find / -name cerveza
[bender@localhost bender]$
```

Esta línea de comando lo que hace es buscar a partir del directorio "/", incluyendo sus subdirectorios, un fichero que tenga por nombre `cerveza`. Otras opciones que podemos indicar en la busqueda son:

-group realiza la busqueda por el grupo del propietario del fichero.

-iname realiza la busqueda por nombre pero sin hacer distinción entre mayúsculas y minúsculas.

-inum realiza la busqueda por inodo.

-nouser realiza la busqueda de ficheros tales que el `UID` de su propietario no se corresponde con ninguno en el sistema.

-nogroup realiza la busqueda de ficheros tales que el `GID` de su propietario no se corresponde con ninguno en el sistema.

-type realiza la busqueda de un fichero por su tipo, enlace simbólico, dispositivo de bloques, socket, ...

También se pueden realizar acciones, como escribir a un fichero, cuando se termina la ejecución de `find`, además de realizar busquedas buscando ficheros que fueron modificados en una determinada fecha, que fueron accedidos, ...

Este comando es tan amplio e importante que lo mejor sería que le echaras un ojo a `man find` o `info find`. Volveremos a ver más sobre este comando en el apartado 11.3 en la página 178.

2.4.3. El comando `slocate`

Este comando nos ayuda a encontrar en el sistema ficheros o directorios que se llamen de una forma determinada. Para encontrar ficheros buscando unicamente por nombre es una opción mejor que `find` ya que no muestra los accesos denegados:

```
[jose@localhost adm]$ slocate Adm-Basica.tex
/home/jose/adm/Adm-Basica.tex
[jose@localhost.adm]$
```

2.4.4. El comando `whatis`

Este comando nos indica qué es lo que en realidad hace un comando:

```
[jose@localhost jose]$ whatis gcc xmms emacs gimp
gcc                  (1)  - GNU project C and C++ Compiler (gcc - 2.96)
XMMS [xmms]          (1)  - an audio player for X
emacs                (1)  - GNU project Emacs
gimp                 (1)  - an image manipulation and paint program
[jose@localhost jose]$
```

Esto lo realiza buscando en una base de datos que está presente en el sistema.

2.4.5. El comando `whereis`

Este comando nos indica donde estan los binarios, fuentes y páginas del manual de un determinado comando:

```
[jose@localhost jose]$ whereis gimp
gimp: /usr/bin/gimp /etc/gimp /usr/lib/gimp /usr/share/gimp
      /usr/share/man/man1/gimp.1.gz
[jose@localhost jose]$
```

2.4.6. El comando `which`

Este comando nos indica la ruta completa en la que se encuentra el comando o los comandos que se le pasan como argumentos:

```
[jose@localhost jose]$ which gcc X gimp latex
/usr/bin/gcc
/usr/X11R6/bin/X
/usr/bin/gimp
/usr/bin/latex
[jose@localhost jose]$
```

2.5. Comunicación con otros usuarios

La verdad que este no es el mejor lugar para introducir estos comandos y lo más indicado sería hacerlo en un capítulo sobre comunicaciones en **GNU/Linux**, pero muchas veces la mejor ayuda nos la pueden proporcionar otros usuarios.

Dado que **GNU/Linux** fue diseñado como un sistema multiusuario y en el conviven multitud de usuarios, pudiendo utilizar la máquina al mismo tiempo es posible comunicarse con el resto de usuarios en "directo".

2.5.1. El comando `mail`

Este comando se utiliza para leer y mandar correos electrónicos. Es bastante rudimentario y normalmente no se utiliza a no ser para cosas puntuales. Es preferible utilizar otros gestores de correo como `elm` o `pine`, que funcionan en modo texto, o cualquiera de los muchos que hay que funcionan en el entorno gráfico X-Window.

En el apartado 3.3, en la página 45, se ve un ejemplo del uso de `mail` con "*pipes*" o "*tuberías*". También podemos ver otro ejemplo en el apartado 3.4, en la página 46, del uso de `mail` con las "*redirecciones*".

2.5.2. El comando `talk`

Este comando nos permite hablar con otros usuarios en nuestra máquina o en otra máquina. Divide la pantalla en dos partes y se puede "chatear". Para que funcione es necesario que esté instalado el servidor `talkd`.

2.5.3. El comando `who`

Este comando nos da información sobre que usuarios están conectados al sistema en este momento:

```
[jose@localhost jose]$ who
jose     tty1     Mar 25 12:54
jose     pts/0    Mar 25 12:55
jose     pts/1    Mar 25 12:56
jose     pts/2    Mar 25 13:25
[jose@localhost jose]$
```

Podemos ver que el usuario `jose` esta conectado en modo terminal de texto, `tty1`, y tiene otras tres consolas abiertas, `pts/*`.

2.5.4. El comando `whoami`

Este comando nos indica quien somos en el sistema. Aunque pueda parecer una tontería es muy util ya que es muy frecuente el asumir la identidad de varios usuarios, mediante `su`, para realizar varias tareas y a veces no sabemos quien somos:

```
[jose@localhost jose]$ whoami
jose
[jose@localhost jose]$
```

2.5.5. Los comandos `write` y `mesg`

Con el comando `write` podemos mandar mensajes a un usuario que esté conectado y los mensajes le apareceran el la terminal, lo cual suele ser un pelín molesto. Para evitar esto podemos ver si nuestra terminal permite la aparación de esos mensajes con el comando `mesg`:

```
[jose@localhost jose]$ mesg
is y
[jose@localhost jose]$
```

lo cual nos indica que cualquiera que nos haga un `mesg` escribirá en nuestra terminal. Para permitir o no que otros usuarios escriban en nuestra terminal bastará con pasarle como argumento a `mesg` "y" en caso de que queramos ver esos mensajes y "n" en caso contrario:

```
[jose@localhost jose]$ mesg n
[jose@localhost jose]$
```

2.6. Ayuda en internet

GNU/Linux se desarrolló gracias a internet y es en internet donde podemos encontrar una gran cantidad de información sobre **GNU/Linux**:

1. El proyecto Lucas. Traducción al Español de los numerosos HOW-TO, ...

   `http://lucas.hispalinux.es`

2. Insflug. Traducción al Español de pequeñas guias sobre **GNU/Linux**.

   `http://www.insflug.org`

3. The Linux Documentation Proyect. En Inglés.

   `http://www.tldp.org`

4. Linux Software Map. Base de datos sobre software disponible para **GNU/Linux**. En Inglés.

   `http://www.boutella.com/lsm`

Capítulo 3

La shell en GNU/Linux

En este capítulo vamos a introducir un poco las "*shell*" y sobre todo la shell `bash` que es la más utilizada.

3.1. ¿Que es una shell?

Los ordenadores son máquinas que lo único que entienden son "ceros" y "unos", para funcionar necesitan un programa especial llamado **Sistema Operativo**.

El sistema operativo es el que interactua entre el ordenador y el ser humano y lo hace mediante la `shell` o interprete de comandos como es más conocida en otros sistemas comerciales.

La `shell` no es más que un programa que transmite nuestras ordenes al sistema operativo y este al ordenador.

En **GNU/Linux** tenemos varias `shells`, cada usuario puede utilizar la que más le guste o mejor se adapte a sus necesidades. Algunas de las `shells` presentes son:

- La shell **bash** o GNU **B**ourne **A**gain **Sh**ell.
- La shell **ksh** o **K**orn **Sh**ell.
- La shell **csh** o Berkeley UNIX **C Sh**ell.
- La shell **tcsh** versión mejorada de **csh**.

Utilicemos la `shell` que utilicemos tenemos que tener en cuenta que las `shell` son "**CASE SENSITIVE**", es decir distinguen entre mayúsculas y minúsculas. También hay que tener cuidado con algunos caracteres que pudieran dar problemas, como por ejemplo nuestra queridisima **Ñ**.

3.1.1. ¿Que `shells` hay instaladas en el sistema?

Podemos ver las `shells` que hay instaladas en el sistema en el archivo:

```
/etc/shells
```

3.1.2. Cambiando nuestra `shell`

La `shell` que utilizamos está especificada en el archivo `/etc/passwd`, para cambiar nuestra `shell` hay que cambiar ese archivo, lo cual no podremos hacerlo a menos que tengamos privilegios de `root` o podamos conseguirlos ;-).

Si queremos cambiar nuestra `shell` lo podemos hacer nosotros mismos utilizando el comando `chsh`, a continuación nos pedirá nuestra contraseña y luego nos pedirá que le indiquemos la `shell` que queremos utilizar, tenemos que tener en cuenta que:

- La `shell` que queremos utilizar tiene que estar listada en `/etc/shells`. Para ver las `shells` listadas basta con invocar el comando `chsh` con la opción `-l`:

  ```
  [bender@localhost bender]$ chsh -l
  ```

- Le tenemos que pasar toda la ruta de acceso a la `shell`.

3.2. La shell BASH

Es la `shell` más ampliamente utilizada dentro del mundo **GNU/Linux** y es una versión libre de la `shell` Bourne originaria de `UNIX`. Esta `shell` pertenece a la Free Software Foundation y se distribuye de forma libre con lo que cualquiera puede instalarla y usarla en su sistema.

3.2.1. ¿Donde puedo encontrar ayuda sobre la `shell` BASH?

Pues dentro del sistema:

```
[bender@localhost bender]$ man bash
```

o bien:

```
[bender@localhost bender]$ info bash
```

3.2.2. Variables de entorno

Podemos tener definidas unas variables de entorno que tengan almacenada información util como por ejemplo:

HOME tiene almacenado el directorio personal.

LOGNAME tiene almacenado el nombre de usuario.

MAIL tiene almacenado el directorio que contiene el correo.

PATH tiene almacenado el `PATH`. Los directorios en los que buscará los comandos a ejecutar cuando no le indiquemos la ubicación de los comandos.

TMOUT tiene almacenado el tiempo tras el cual la consola se cerrará si ha estado inactiva. Tiene que ser mayor que cero y expresada en segundos.

podemos ver todas las variables de entorno que tenemos definidas ejecutando `set`.

Por el contrario si queremos ver el contenido de una variable tenemos que anteponerle el signo del dolar:

```
[bender@localhost public_html]$ $HOME
/home/bender
[bender@localhost public_html]$
```

3.2.3. Ficheros importantes

La `shell` BASH utiliza unos ficheros en los que almacena la información para personalizar las cuentas de los usuarios:

`/etc/profile` este fichero unicamente lo puede modificar el `root` y contiene los perfiles generales para todos los usuarios.

`~/.bash_profile` este fichero está en el directorio de trabajo de cada usuario y puede ser modificado por el usuario para personalizar su cuenta.

`~/.bash_login` igual que `~/.bash_profile`.

`~/.profile` igual que `~/.bash_profile`.

`~/.bash_logout` este fichero está en el directorio de trabajo de cada usuario y puede ser modificado por el usuario. Este fichero se utiliza para indicar al sistema lo que tiene que hacer cada vez que terminemos la sesión.

`~/.bash_history` contiene el histórico de comandos.

Cuando se invoca la `shell` BASH lo que hace es:

1. Leer los perfiles generales de `/etc/profile`.
2. Busca en el directorio de trabajo del usuario el fichero `~/.bash_profile` y carga los perfiles particulares del usuario, en caso de no existir busca el fichero `~/.bash_login` y si este no existe busca el `~/.profile`.
3. El prompt, línea de comandos, del sistema queda libre para darle ordenes.
4. Cuando se teclea `exit` o se pulsa `CTRL + D` la `shell` ejecuta `.bash_logout` y termina la sesión.

3.3. Pipes o tuberías

En UNIX se puede utilizar la salida de un comando como entrada de otro comando, esto se conoce como "*pipes*" o "*tuberías*". Un ejemplo de esto sería:

```
[bender@localhost robopilingis]$ ls -l | mail calculon
```

Si ejecutaramos esto se haría lo siguiente:

1. Se ejecuta `ls -l`.
2. Se ejecuta `mail calculon`, pero utilizando la salida del anterior comando como entrada para este comando.

Lo que haría esta línea de comando sería mandar un correo electrónico al usuario `calculon`, de la máquina local, conteniendo el listado del directorio `robopilingis`.

Podemos utilizar todos los pipes que queramos, unicamente hemos de poner el símbolo "|" entre los comandos. Por ejemplo, supongamos que `bender` quiere mandar a `calculon` la dirección de una robopilingi que le gustó especialmente y `bender` tiene su "*agenda*" especial en un fichero de texto llamado direcciones:

```
[bender@localhost robopilingis]$ cat direcciones | grep plugme |
                                 mail calculon
```

Esto haría lo siguiente:

1. Consideraría todas las líneas del fichero `direcciones`.
2. Unicamente escogería aquellas en las que apareciera la palabra `plugme`, que sería la robopilingi que quiere recomendarle a su amigo `calculon`.
3. Mandaría esas líneas por correo electrónico al usuario `calculon` en la máquina local.

3.4. Redireccionando la entrada y la salida

Normalmente la salida de un comando se hace por pantalla, en **GNU/Linux** podemos redireccionar esa salida hacía otro fichero o dispositivo.

3.4.1. Redirección con >

Podemos redireccionar la salida de un comando con el símbolo > hacía un fichero o dispositivo. Sí lo hacemos hacía un fichero borrará el contenido del fichero y luego escribirá en él la salida del comando. En caso de que dicho fichero no existiera lo crearía:

```
[bender@localhost robopilingis]$ cat agenda > nuevaAgenda
[bender@localhost robopilingis]$
```

con esto se crearía, en caso de no existir, un nuevo fichero llamado `nuevaAgenda` que contendría los mismos datos que `agenda`. En caso de que ya existiera antes de escribir los datos borraría el contenido.

3.4.2. Redirección con >>

Podemos redireccionar la salida de un comando con el símbolo >> hacía un fichero o dispositivo. Sí lo hacemos hacía un fichero lo añadiría al final del fichero, sin borrar lo que previamente hubiera en el fichero. En caso de que dicho fichero no existiera lo crearía:

```
[bender@localhost robopilingis]$ cat agenda1 > nuevaAgenda
[bender@localhost robopilingis]$ cat agenda2 >> nuevaAgenda
```

con esto se crearía, en caso de no existir, un nuevo fichero llamado `nuevaAgenda` que contendría los mismos datos que `agenda1`. En caso de que ya existiera antes de escribir los datos borraría el contenido. Luego añadiría el contenido de `agenda2` al fichero `nuevaAgenda`.

3.4.3. Redirección con <

Con este símbolo indicamos el lugar de procedencia de los datos de entrada de un comando:

```
[bender@localhost bender]$ mail calculon < cachoperro
[bender@localhost bender]$
```

Con esto el usuario `bender` mandaría un correo al usuario `calculon`, pero el mensaje del correo sería el contenido del fichero `cachoperro`.

Capítulo 4

Los procesos en GNU/Linux

En este capítulo vamos a tratar los procesos y algunos comandos relacionados con ellos.

4.1. ¿Qué son los procesos?

Los procesos no son "*nada*" más que programas que estan en memoria ejecutandose o listos para ejecutarse.

Cada vez que ejecutamos un programa se crea un proceso. Todos los programas en ejecución son procesos. Para el nucleo todo lo que se está ejecutando son procesos y la comunicación entre programas en ejecución, el acceso de los programas en ejecución a los recursos se realiza mediante la gestión de procesos.

4.2. Estados de los procesos

Cuando un programa se ejecuta se crea un proceso y cada proceso puede tener un estado que nos indica su estado actual. Estos estados son necesarios ya que muchas veces un proceso necesita interactuar con otros procesos, ya que la CPU sólo puede ejecutar un proceso en cada instante, o puede necesitar acceder a algún recurso del sistema que esté siendo utilizado por otro proceso, con lo cual tendrá que esperar a que ese proceso termine de utilizar el recurso y lo libere.

4.2.1. El estado RUN

Cuando el estado de un proceso es **RUN** dicho proceso está utilizado la CPU en ese momento.

4.2.2. El estado READY

Cuando el estado de un proceso es **READY** dicho proceso está en condiciones de ser ejecutado, tiene el control de los recursos necesarios para su ejecución, pero en esos momentos la CPU está ejecutando otro proceso, con lo cual tiene que esperar a que la CPU quede libre.

4.2.3. El estado WAIT/SLEEP

Cuando el estado de un proceso es **WAIT** dicho proceso está esperando a que quede libre un recurso del sistema que necesita para su ejecución, o bien necesita datos que estan siendo procesados por otros procesos, con lo cual tendrá que esperar a que terminen y le faciliten esos datos.

4.2.4. El estado STOPPED

El proceso ha sido parado, por ejemplo mediante el envio de una señal[1].

4.2.5. El estado ZOMBIE

El nucleo mantiene en todo momento una tabla con todos los procesos, cuando un proceso termina su ejecución pasa a estado **ZOMBIE** y permanece en ese estado hasta que es borrado de la tabla de procesos.

4.3. La multitarea y los procesos

GNU/Linux es un sistema multitarea, eso significa que puede realizar varias tareas a la vez. Existe un problema y es que la CPU no puede ejecutar más de una tarea a la vez. Para realizar multitarea lo que hace el nucleo del sistema operativo es dividir el tiempo de CPU y asignarle un poco de dicho tiempo a cada proceso.

[1] Apartado 4.9.2 en la página 61.

Por ejemplo, la CPU puede asignar 30 milisegundos a cada proceso, una vez que un proceso ha utilizado su tiempo de CPU es guardado en el estado actual y la CPU deja que otro proceso utilice sus 30 milisegundos, y asi sucesivamente, cuando todos los procesos han utilizado sus 30 milisegundos entonces la CPU deja que el primer proceso vuelva a usar 30 milisegundos la CPU, luego el segundo proceso, ...

Esto es una forma bastante sencilla de explicar como funciona la multitarea en un sistema operativo. En realidad la forma en que funciona la multitarea depende del algoritmo(s) de "**scheduling**", planificación, que tenga implementado el nucleo del sistema.

Algunos algoritmos de scheduling:

- Round Robin, el que acabamos de describir. Supone que todos los procesos tienen la misma prioridad dentro del sistema (democracia pura ;-)).
- Scheduling con prioridad, cada proceso consume tiempo de CPU dependiendo de la prioridad que le haya sido asignada. A mayor prioridad más tiempo de CPU.

GNU/Linux implementa un algoritmo de Scheduling con prioridad.

4.3.1. Cambios de contexto

Cuanto un proceso termina su tiempo de uso de CPU ha de ser guardado en su estado para que continue su ejecución la próxima vez que le corresponda utilizar la CPU. Esta tarea de guardar un proceso y hacer que el sisguiente proceso utilice la CPU consume tiempo de CPU y recibe el nombre de "*cambio de contexto*". A la hora de diseñar un sistema operativo hay que tener cuidado con esto.

Supongamos que un cambio de contexto consume un milisegundo. Si los tiempos que asignamos a cada proceso son de 10 milisegundos tendremos que cada 10 milisegundos hemos perdido un milisegundo con los cambios de contexto, luego en un segundo de uso de CPU habremos perdido 100 milisegundos. Si aumentamos el tiempo de uso de CPU de los procesos, por ejemplo a 500 milisegundos tendremos que por cada segundo de uso de CPU sólo habremos perdido dos milisegundos, lo cual está muy bien, pero plantea un problema. Si el tiempo asignado a cada proceso, por ejemplo 500 milisegundos, es grande tendremos que los procesos interactivos tendran una respuesta

mediocre, fijaos cuando vuestro ordenador esta saturado y tratais de escribir, el tiempo de respuesta es lentísimo[2].

Resumiendo:

- Si el tiempo de uso de la CPU es pequeño se perderá mucho tiempo en los cambios de contexto y bajará la eficacia de la CPU.
- Si el tiempo de uso de la CPU es grande la respuesta a los procesos interactivos se verá perjudicada.

4.4. El comando `jobs`

Mediante este comando podemos ver todos los procesos que se estan ejecutando en la terminal que lo hemos ejecutado:

```
[jose@localhost jose]$ jobs
[1]-  Running                 gv Adm-Basica.ps &
[2]+  Running                 emacs Adm-Basica.tex &
[jose@localhost jose]$
```

Podemos ver los procesos que hemos ejecutado en la terminal y su estado, los dos procesos que nos muestra estan ejecutandose "`Running`".

Con este comando se pueden utilizar dos flags:

-l que además nos lista los `PID` de cada proceso:

```
[jose@localhost jose]$ jobs -l
[1]-  1033 Running                 gv Adm-Basica.ps &
[2]+  1035 Running                 emacs Adm-Basica.tex &
[jose@localhost jose]$
```

-p que unicamente nos muestra el `PID` de los procesos:

```
[jose@localhost jose]$ jobs -l
1033
1035
[jose@localhost jose]$
```

[2]Los que utiliceis Windows estareis bastante acostumbrados.

4.5. Ejecución en background y los comandos Control+Z, bg y fg

Cuando ejecutamos un programa, en la consola, la consola queda ocupada hasta que el programa termina su ejecución y libera la consola. Mientras el programa se está ejecutando no podemos utilizar esa consola, con lo cual necesitaremos abrir otra o esperar a que termine.

Podemos ejecutar programas y/o comandos en background con sólo añadir "`&`" al final del comando que queremos ejecutar:

```
[bender@localhost bender]$ emacs Adm-Basica.tex &
[2] 4725
[bender@localhost bender]$
```

Con esto tendremos la consola libre para seguir trabajando. Podemos observar que después de ejecutar el comando obtenemos dos números, uno de ellos entre corchetes, y luego el prompt de la consola para poder seguir trabajando. La interpretación de los números que hemos obtenido es:

- "`[2]`", nos indica que es el segundo comando que lanzamos en background.
- "`4725`", nos indica el número del proceso.

Un proceso interactivo, que necesite interactuar con el usuario, no tiene sentido lanzarlo en background ya que ocupará la terminal en el momento en que necesite pedir datos al usuario o mostrarselos al usuario.

4.5.1. Como parar un proceso en ejecución

Hay veces que lanzamos un proceso, ejecutamos un programa, y se nos olvida el ponerlo en background, con lo que tenemos la terminal ocupada. Por ejemplo supongamos en estamos en X-Window y desde una consola arrancamos el programa **xmms** para escuchar musica:

```
[jose@localhost jose]$ xmms
```

La consola queda ocupada ya que no lo hemos lanzado en background, si queremos parar el proceso, sin matarlo, bastaría con que pulsasemos "`Control+Z`", veriamos en la consola:

```
[jose@localhost jose]$ xmms

[3]+  Stopped                 xmms
[jose@localhost jose]$
```

A continuación la musica seguiría oyendose hasta que se acabe la canción, pero veremos que la interface gráfica del programa se ha parado. A partir de este momento podemos hacer lo que queramos con el proceso, podríamos pasarlo a segundo plano con el comando `bg`.

4.5.2. El comando `bg`

Este comando se utiliza para pasar procesos a segundo plano, es muy útil cuando hemos parado la ejecución de un proceso con "`Control+Z`". Siguiendo con el ejemplo anterior, si quisieramos pasar la ejecución de `xmms` a segundo plano para que nos deje libre la consola entonces:

```
[jose@localhost jose]$ xmms

[3]+  Stopped                 xmms
[jose@localhost jose]$ bf %3
[3]+ xmms &
[jose@localhost jose]$
```

Tendríamos ahora que `xmms` se estaría ejecutando en background y la terminal estaría libre.

Este comando pasa a segundo plano aquellos procesos que se le indiquen utilizando "`%`" seguido del número del trabajo, no confundir con el `PID`, número entre corchetes.

Una vez presionado `Control+Z` para poner ese proceso en segundo plano no es necesario especificar su número, bastaría con ejecutar el comando `bg` para ponerlo en segundo plano.

4.5.3. El comando `fg`

Este comando se utiliza para pasar a primer plano procesos que están parados o en segundo plano. Funciona igual que el comando `bg`. Probar a lanzar un programa, no en background, pulsar `Control+Z`, entonces tendríamos la terminal libre y nos indicaría el número del trabajo entre corchetes, por ejemplo `4`. Si ejecutamos en la terminal:

fg

pondríamos otra vez en primer plano el proceso cuyo número de trabajo es `4` y tendríamos ocupada otra vez la terminal.

4.6. Atributos de los procesos

Todos los procesos tienen unos atributos para que el sistema operativo pueda distinguir unos de otros y para controlar a que recursos puede acceder cada proceso y con qué privilegios puede acceder.

A continuación detallamos algunos de esos atributos.

4.6.1. El atributo `PID`

Cuando se crea un proceso se le asigna un número de identificación, el `PID`. El sistema operativo identificará a cada proceso por su `PID`. Cuando lanzamos un proceso en background el segundo número que nos aparece en pantalla, el que no está entre corchetes, es el `PID` del proceso.

4.6.2. El atributo `PPID`

Todo proceso es lanzado por otro proceso, que es conocido como proceso padre. El proceso padre, por ser un proceso, tiene un `PID`, el atributo `PPID` de un proceso no es más que el `PID` del proceso que lo lanzó.

Por ejemplo cuando entramos en sesión se ejecuta una shell, la cual es un proceso y tiene asignado un `PID`, todos los procesos que lancemos desde esa shell serán hijos de la shell y su `PPID` será el `PID` de la shell. Si un proceso que hemos lanzado lanza a su vez otro proceso, este nuevo proceso será nieto de shell y asi sucesivamente.

4.6.3. El usuario real (`RUID`) y el efectivo (`EUID`)

En el apartado 1.5, página 25, vimos que existian unos determinados permisos que se ejecutaban con los privilegios del propietario en lugar de los privilegios del usuario que lo ejecuta. El atributo `RUID` es el `UID` del usuario que lanza un proceso y el atributo `EUID` es el `UID` del usuario que asume la identidad el proceso (permisos SUID).

4.6.4. El grupo real (`RGID`) y el efectivo (`EGID`)

El grupo real y el efectivo funcionan igual que los usuarios reales y efectivos, ver permisos SGID en el apartado 1.6 en la página 28.

4.7. Prioridad de los procesos y los comandos `nice` y `renice`

Todos los procesos tienen asignada una prioridad que indica el tiempo que pueden utilizar la CPU del sistema de forma continuada. Los procesos se van turnando para utilizar la CPU del sistema y cada proceso la ocupa más o menos tiempo dependiendo de la prioridad que tenga asignada.

La forma en la que se gestionan las prioridades en cada sistema depende del algoritmo de **planificación** o **scheduling** que tenga implementado el sistema operativo.

Las prioridades de los procesos en ejecución se pueden modificar, pero sólo pueden hacerlo:

- El propietario del proceso, y nunca puede dar mayor prioridad al proceso de la que le ha asignado el sistema.
- El `root`, que puede modificar la prioridad de todos los procesos, aumentarla y disminuirla.

El rango de prioridades suele ir desde -20 (máxima prioridad) hasta 20 (mínima prioridad). El rango de prioridades negativas está reservado al superusuario, cualquier proceso lanzado por un usuario normal tendrá una prioridad entre 0 y 20, a menos que el `root` le establezca una prioridad negativa.

4.7.1. El comando `nice`

Este comando permite ejecutar un comando con una determinada prioridad. Si ya estas pensando en ejecutar un comando con la mayor prioridad dentro del sistema te tengo que dar una mala noticia, a menos que seas el superusuario no podrás.

Para lanzar un comando con una prioridad determinada:

```
[jose@localhost jose]$ nice -n 10 xdvi &
[4] 1277
[jose@localhost jose]$
```

4.7.2. El comando `renice`

Este comando nos permite cambiar la prioridad de nuestros procesos, reducirla, nunca aumentarla. Si se pudiera aumentar todos los usuarios, o casi todos, aumentarían la prioridad de sus procesos al máximo, lo cual no sería muy bueno para el sistema.

En el ejemplo anterior le hemos puesto una prioridad de 10 al programa `xdvi`, si quisieramos darle más prioridad:

```
[jose@localhost jose]$ renice +5 1277
renice: 1277: setpriority: Permiso denegado
[jose@localhost jose]$
```

Veríamos que no es posible, pero si queremos reducir la prioridad:

```
[jose@localhost jose]$ renice +12 1277
1277: prioridad antigua 10, nueva prioridad 12
[jose@localhost jose]$
```

La sintaxis es muy sencilla indicamos la prioridad y despues el `PID` del proceso.

4.8. Obtención de información sobre los procesos

En **GNU/Linux** existen comandos para obtener información sobre los procesos que se estan ejecutando, su prioridad, estado, `PID`, ...

4.8.1. El comando ps

Este comando nos da información sobre los procesos que se estan ejecutando, es bastante amplio, por lo cual unicamente veremos algunos ejemplos.

Si ejecutamos `ps` en una terminal obtendremos los procesos que se están ejecutando en esa terminal:

```
[jose@localhost jose]$ ps
  PID TTY          TIME CMD
 1006 pts/1    00:00:00 bash
 1035 pts/1    00:00:00 gv
 1037 pts/1    00:00:05 emacs
 1044 pts/1    00:00:10 gs
 1045 pts/1    00:00:00 ps
[jose@localhost jose]$
```

Podemos ver varios campos:

PID número de identificación del proceso.

TTY terminal en la que se está ejecutando.

TIME tiempo de CPU consumido por el proceso.

CMD comando que se está ejecutando.

Vamos a ver algunos ejemplos del uso de `ps`, para más información mira la página del manual o la información en `info`.

Podemos obtener más información de los procesos utilizando los flags "`-f`":

```
[jose@localhost jose]$ ps -f
UID        PID  PPID  C STIME TTY          TIME CMD
jose      1006  1005  0 20:40 pts/1    00:00:00 /bin/bash
jose      1035  1006  0 20:40 pts/1    00:00:00 gv Adm-Basica.ps
jose      1037  1006  0 20:40 pts/1    00:00:16 emacs Adm-Basica.tex
jose      1087  1035  1 21:07 pts/1    00:00:08 gs
jose      1098  1006  0 21:16 pts/1    00:00:00 ps -f
[jose@localhost jose]$
```

o con el flag "-l":

```
[jose@localhost jose]$ ps -l
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME CMD
000 S   500  1006  1005  0  69   0    -   647 wait4  pts/1    00:00:00 bash
000 S   500  1035  1006  0  69   0    -   962 do_sel pts/1    00:00:00 gv
000 S   500  1037  1006  0  69   0    -  2274 do_sel pts/1    00:00:18 emacs
000 S   500  1087  1035  1  69   0    -  2473 do_sel pts/1    00:00:08 gs
000 R   500  1099  1006  0  78   0    -   773 -      pts/1    00:00:00 ps
[jose@localhost jose]$
```

Veamos que significan los campos nuevos:

F flags asociados con el proceso.

S estado del proceso.

UID identificador del usuario que es propietario del proceso.

PPID `PID` del proceso padre.

C uso del procesador para el scheduling.

PRI prioridad del proceso.

NI número nice.

ADDR dirección del proceso.

SZ tamaño del proceso.

WCHAN evento por el que está esperando el proceso.

Además podemos elegir que información queremos ver con el comando `ps` utilizando el flag "`-o`" seguido del identificador del campo. Algunos de los identificadores de campo son:

UID `ruser`.

PID `pid`.

PPID `ppid`.

NI `nice`.

SZ `sz,vsz`.

TTY `tty`.

TIME `time`.

CMD `comm,args`.

```
[jose@localhost jose]$ ps -o pid -o ppid -o ruser -o nice -o comm
  PID  PPID RUSER     NI COMMAND
 1006  1005 jose       0 bash
 1035  1006 jose       0 gv
 1037  1006 jose       0 emacs
 1106  1035 jose       0 gs
 1181  1006 jose       0 ps
[jose@localhost jose]$
```

Además `ps` puede efectuar la salida utilizando el estilo de otros sistemas como BSD, AIX, ...

Ver los procesos asociados a un determinado usuario

Para ver todos los procesos asociados al usuario 43:

```
[jose@localhost jose]$ ps -U 43
  PID TTY          TIME CMD
  810 ?        00:00:03 xfs
[jose@localhost jose]$
```

En lugar del UID también podemos especificar su nombre.

Ver los procesos asociados a un determinado grupo

Para ver todos los procesos asociados al grupo 29:

```
[jose@localhost jose]$ ps -G 29
  PID TTY          TIME CMD
  521 ?        00:00:00 rpc.statd
[jose@localhost jose]$
```

En lugar del GID también podemos especificar su nombre.

Ver información sobre varios procesos determinados

Si queremos ver la información referente a los procesos cuyos `PID` son 1037 y 1087:

```
[jose@localhost jose]$ ps -p ``1037,1087''
  PID TTY          TIME CMD
 1037 pts/1    00:00:15 emacs
 1087 pts/1    00:00:08 gs
[jose@localhost jose]$
```

Ver los procesos asociados a un determinado terminal

Si queremos ver la información referente a los procesos ejecutados en las terminales `tty1` y `pts/1`:

```
[jose@localhost jose]$ ps -t pts/1 -t tty1
  PID TTY          TIME CMD
  853 tty1     00:00:00 login
  861 tty1     00:00:00 bash
  902 tty1     00:00:00 startx
  909 tty1     00:00:00 xinit
  913 tty1     00:00:00 startkde
  987 tty1     00:00:01 ksmserver
 1006 pts/1    00:00:00 bash
 1035 pts/1    00:00:01 gv
 1037 pts/1    00:00:30 emacs
 1106 pts/1    00:00:10 gs
 1224 pts/1    00:00:00 ps
[jose@localhost jose]$
```

4.8.2. El comando `pstree`

Este comando nos da información sobre los procesos que se están ejecutando y nos la ofrece en forma de árbol. De esta forma podemos rastrear que procesos han lanzado a otros procesos:

```
[jose@localhost jose]$ pstree
init-+-artsd
     |-atd
     |-crond
     |-gpm
     |-kapm-idled
     |-7*[kdeinit]
     |-kdeinit-+-kdeinit
     |         `-kdeinit---bash-+-emacs
     |                          |-gv---gs
     |                          `-pstree
     |-kdeinit---cat
     |-keventd
     |-klogd
     |-login---bash---startx---xinit-+-X
     |                               `-startkde---ksmserver
     |-mdrecoveryd
     |-5*[mingetty]
     |-nfsd---lockd---rpciod
     |-7*[nfsd]
     |-portmap
     |-raid1d
     |-rpc.mountd
     |-rpc.rquotad
     |-rpc.statd
     |-sendmail
     |-syslogd
     |-xfs
     `-xinetd---fam
[jose@localhost jose]$
```

4.8.3. El comando `top`

El comando `top` nos ofrece información continua sobre los procesos que se están ejecutando, usos de CPU, ... para verlo mejor que lo ejecutes.

4.9. Eliminación de procesos

Hay veces que un proceso no funciona correctamente y hay que terminar su ejecución. Existen comandos para forzar a un proceso a que termine su ejecución. Todo esto se hace mediante el uso de señales.

4.9.1. Como termina la ejecución de un proceso

El nucleo del sistema mantiene una tabla con todos los procesos. Cuando un proceso termina su ejecución, se completa el programa, realiza una llamada al sistema `exit ()` y son liberados todos los recursos que estaba utilizando y pasa a estado ZOMBIE, es decir que permanece en la tabla de procesos aunque ya ha terminado su ejecución.

Un proceso ZOMBIE ocupa sitio en la tabla de procesos, el sistema puede liberar esa zona o lo puede hacer el proceso padre del proceso ZOMBIE.

En el mundo `UNIX` todo proceso es hijo de otro, con la excepción del proceso `init`, todo proceso puede realizar varias tareas con sus hijos, una de ellas es matarlos. Para matar un proceso se suele hacer mediante el uso de señales. Las señales son una de las formas que tenemos para realizar comunicaciones entre procesos.

4.9.2. Señales

En un sistema informático es necesario que los diferentes procesos que se estan ejecutando se comuniquen entre si, existen varias formas:

- Mediante el uso de sockets.
- Mediante el uso de pipes.
- Mediante el uso de colas de mensajes.
- Mediante el uso de semáforos.
- Mediante el uso de memoria compartida.
- Mediante el uso de señales.

Una señal es una forma de llamar la atención de un proceso para que realice una determinada acción que no estaba prevista:

- Si estamos imprimiendo un trabajo demasiado grande y necesitamos dejar la impresora libre para que el jefe imprima unas fotos que se acaba de bajar de internet ;-). En ese caso le mandamos una señal al proceso que está imprimiendo para que termine en ese mismo momento.
- Cuando hemos hecho un programa y comprobamos al ejecutarlo que no funciona como debería y no nos deja la terminal libre. Abrimos otra terminal y le mandamos una señal para que se muera y deje la teminal libre.

Las señales son un mecanismo existente desde las primeras versiones de `UNIX`. Las señales se representan en el sistema con nombres de la forma "`SIGXXX`". Podemos ver el número total de señales presentes en el sistema curioseando un poco en los fuentes del nucleo dentro del fichero:

`/usr/src/linux-x.y/asm/signal.h`

La variable `NSIG` nos indica el número de señales presentes en el sistema.

Las señales se pueden generar de varias formas:

1. Por una excepción del hardware. Si un proceso intenta escribir en una zona de memoria en la que no tiene privilegios para escribir se produce una excepción que es atrapada por el nucleo y produce una señal `SIGSEGV` que es enviada al proceso que intentó el acceso no autorizado y se produce un "`core dumped`".
2. Si estamos ejecutando un proceso en una terminal y pulsamos `Control+C` entonces se genera la señal `SIGINT` y se finaliza el proceso que se está ejecutando en primer plano.
3. Mediante una llamada a la función del sistema `kill` o mediante la ejecución del mismo comando.

Cuando un proceso recibe una señal puede realizar tres acciones:

1. Ignorar la señal.
2. Terminar el proceso.
3. Terminar el proceso y hacer un volcado de la memoria (generar un archivo core).

Cada señal tiene una acción predeterminada que ha de realizarse cuando el proceso recibe la señal, pero los procesos pueden modificar esa acción. Para evitar que un proceso sea incontrolable existen dos señales que no pueden ser modificadas y son `SIGKILL` y `SIGSTOP`, estas señales permiten al `root` interrumpir o suspender la ejecución de cualquier proceso dentro del sistema.

GNU/Linux dispone de dos señales a disposición del programador que son `SIGUSR1` y `SIGUSR2`.

Supongamos que iniciamos en una terminal un proceso y no lo hacemos en background, entonces la terminal queda ocupada, si pulsamos `Control+C` mandamos al proceso la señal `SIGINT` que, normalmente, terminaría el proceso:

```
[jose@localhost jose]$ gv
gv: terminated by signal 2
[jose@localhost jose]$
```

4.9.3. El comando `kill`

Este comando nos permite mandar señales a nuestros procesos, podemos ver la señales presentes en el sistema ejecutando este comando con el flag "`-l`":

```
[jose@localhost jose]$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR      31) SIGSYS      32) SIGRTMIN    33) SIGRTMIN+1
34) SIGRTMIN+2  35) SIGRTMIN+3  36) SIGRTMIN+4  37) SIGRTMIN+5
38) SIGRTMIN+6  39) SIGRTMIN+7  40) SIGRTMIN+8  41) SIGRTMIN+9
42) SIGRTMIN+10 43) SIGRTMIN+11 44) SIGRTMIN+12 45) SIGRTMIN+13
46) SIGRTMIN+14 47) SIGRTMIN+15 48) SIGRTMAX-15 49) SIGRTMAX-14
50) SIGRTMAX-13 51) SIGRTMAX-12 52) SIGRTMAX-11 53) SIGRTMAX-10
54) SIGRTMAX-9  55) SIGRTMAX-8  56) SIGRTMAX-7  57) SIGRTMAX-6
58) SIGRTMAX-5  59) SIGRTMAX-4  60) SIGRTMAX-3  61) SIGRTMAX-2
62) SIGRTMAX-1  63) SIGRTMAX
[jose@localhost jose]$
```

Algunas señales a tener en cuenta son:

SIGKILL Termina un proceso (no puede ser ignorada).

SIGSTOP Suspensión del proceso (no puede ser ignorada).

SIGTERM Terminación del proceso (puede ser ignorada).

Por ejemplo:

```
[jose@localhost jose]$ ps -lU 500 | grep xmms
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME CMD
000 S   500  1786  1053  6  69   0    -  4269 do_pol ?        00:00:02 xmms
040 S   500  1787  1786  0  69   0    -  4269 do_pol ?        00:00:00 xmms
040 S   500  1788  1787  0  69   0    -  4269 do_sel ?        00:00:00 xmms
040 S   500  1789  1787  0  69   0    -  4269 do_sel ?        00:00:00 xmms
[jose@localhost jose]$
```

La columna de la "**S**" nos muestra el estado del proceso `xmms`, en este caso nos muestra una "`S`" indicandonos que el proceso está en estado `SLEEP`. Si a continuación le mandamos la señal `SIGSTOP`:

```
[jose@localhost jose]$ kill -s SIGSTOP 1786 1787 1788 1789
[jose@localhost jose]$ ps -lU 500 | grep xmms
  F S   UID   PID  PPID  C PRI  NI ADDR    SZ WCHAN  TTY          TIME CMD
000 T   500  1786  1053  2  69   0    -  4269 do_sig ?        00:00:02 xmms
040 T   500  1787  1786  0  69   0    -  4269 do_sig ?        00:00:00 xmms
040 T   500  1788  1787  0  69   0    -  4269 do_sig ?        00:00:00 xmms
040 T   500  1789  1787  0  69   0    -  4269 do_sig ?        00:00:00 xmms
[jose@localhost jose]$
```

Si nos fijamos en la columna de "**S**" veremos que ahora nos muestra un a"`T`", que nos indica que el proceso está parado. Si hicieramos un `top` veriamos que tenemos, por lo menos cuatro procesos en `stopped` (procesos parados).

Para acabar con los procesos anteriores:

```
[jose@localhost jose]$ kill -9 1786 1787 1788 1789
```

4.9.4. El comando `killall`

Este comando funciona de forma muy parecida a `kill` pero tiene la ventaja de que no es necesario suministrar el número del proceso, `PID`, basta con especificar el nombre del proceso y matará a todos los procesos que se correspondan con el nombre suministrado, y que sean de nuestra propiedad.

4.10. El comando nohup

Todo proceso es hijo de alguien, con la excepción del proceso `init`. Cuando entramos en sesión se crea un proceso que es una copia de la shell que estemos utilizando. Todos los procesos que lancemos serán, hijos, nietos, ... de la shell que tenemos.

Cuando un proceso muere automáticamente mueren todos sus descendientes, hijos, nietos, ... Si estas en X-Window abre una terminal y ejecuta un programa en background, a continuación cierra la terminal y verás como el programa que abriste se cierra también (o debería).

Cuando un proceso recibe la señal `SIGKILL` manda la señal `SIGCHLD` a todos sus hijos para que terminen su ejecución.

Hay veces que necesitamos que un proceso que hemos lanzado siga funcionando después de haber terminado nuestra sesión. Por ejemplo supongamos que hemos creado un programa para factorizar números primos, algo realmente costoso si estamos intentando factorizar un número grande. Supongamos que el programa se llama `factorizar` y el número a factorizar está en el fichero `datos`:

```
[jose@localhost jose]$ ./factorizar datos &
[1] 2044
[jose@localhost jose]$
```

este proceso se estará ejecutando hasta que termine o sea terminado. En el momento en el que terminamos nuestra sesión se envia la señal `SIGKILL` a la copia de la shell que teniamos abierta y esta manda la señal `SIGCHLD` a todos sus hijos, lo que implica que el proceso con `PID` 2044 recibirá esa señal y terminará su ejecución:

```
[jose@localhost jose]$ ./factorizar datos &
[1] 2044
[jose@localhost jose]$ ps -o pid -o ppid -o cmd
  PID  PPID CMD
 1987  1986 -bash
 2044  1987 ./factorizar datos
 2045  1987 ps -o pid -o ppid -o cmd
[jose@localhost jose]$
```

Podemos ver como el `PID` del proceso padre de `factorizar` es 1987 y ese `PID` se corresponde con la copia de la shell `bash`. Cuando terminemos la sesión tanto la shell como todos sus descendientes morirán, si nuestro proceso no ha terminado su trabajo tendremos que volver a empezar.

Para evitar esto tenemos el comando `nohup`, el cual lanza un proceso que cuando el proceso padre muere el proceso lanzado con `nohup` no muere. Si el proceso lanzado con `nohup` produce alguna salida por la salida estándar está se grabará en el fichero "`nohup.out`":

```
[jose@localhost jose]$ nohup ./factorizar datos &
[5] 2241
[jose@icarus jose]$ nohup: appending output to 'nohup.out'

[jose@icarus jose]$ ps -o pid -o ppid -o cmd
  PID  PPID CMD
 1125  1124 /bin/bash
 1607  1125 gv Adm-Basica.ps
 1655  1125 emacs Adm-Basica.tex
 1917  1607 gs
 2241  1125 ./factorizar datos
 2244  1125 ps -o pid -o ppid -o cmd
[jose@localhost jose]$
```

Podemos ver como el `PID` del proceso padre del proceso que hemos lanzado es el 1125, la copia de la shell. Si terminamos la sesión e iniciamos otra veremos lo siguiente:

```
[jose@icarus jose]$ ps -o pid -o ppid -o cmd -U 500
  PID  PPID CMD
  928   853 -bash
 2241     1 ./factorizar datos
 2388  2347 ps -o pid -o ppid -o cmd -U 500
[jose@icarus jose]$  ps -p "1"
  PID TTY          TIME CMD
    1 ?        00:00:04 init
[jose@localhost jose]$
```

Vemos que el `PID` del proceso padre es 1 el cual se corresponde con el proceso `init`. Si realizas todo esto en una consola del entorno X-Window obtendrás unos resultados que no serán exactamente estos, pero eso es debido a que todos los procesos que ejecutes en X serán descendientes del proceso `startx`.

Pero si lanzas un proceso en una terminal X con `nohup`, sales de las X, cierras la sesión y luego abres una nueva podrás ver como tu proceso depende del proceso `init`.

Los procesos lanzados con `nohup` hay que lanzarlos en background, para tener la terminal libre y seguir trabajando. Si no lo lanzamos en background tendremos que interrumpir el proceso con `Control+Z` y ponerlo en segundo plano con el comando `bg`, ya que si lo interrumpimos con `Control+C` habremos matado el proceso.

Capítulo 5

El proceso `login` en GNU/Linux

Cada vez que queremos utilizar nuestro **GNU/Linux** necesitamos autenticarnos, o lo que es lo mismo "*convencer*" al sistema de que somos quien decimos ser.

Esto es necesario para tener a salvo nuestros documentos y que sólo sean accesibles por nosotros.

5.1. ¿Como nos autenticamos en GNU/Linux?

El proceso de autenticación, más utilizado, en **GNU/Linux** consta de dos pasos:

1. Nombre de acceso o `login`.

   Con este dato el sistema conoce al usuario y le permite el acceso a sus datos y establece los privilegios que tiene dentro del sistema.

2. Contraseña o `password`.

   Mediante este dato el sistema averigua si la persona que está intentando acceder al sistema es en realidad el usuario al que corresponde el `login` y no es otro usuario intentando suplantar su personalidad.

5.1.1. Necesidad del uso de contraseñas

Hay gente que piensa que no es necesario el uso de contraseñas, ya que bastaría con ocultar nuestro nombre de usuarios. Esto plantea varios problemas:

1. No podriamos relacionarnos con otros usuarios, ya sean de nuestra máquina u otras máquinas. Si lo hicieramos conocerían nuestro `login` y nuestra máquina y al no tener contraseña podrían acceder tranquilamente al sistema suplantando nuestra personalidad. Además tampoco podríamos utilizar algo tan potente y necesario en nuestros días como el correo electrónico.

2. Todo usuario de una máquina puede conocer los `login` del resto de usuarios, ya que el fichero `/etc/passwd` tiene que ser accesible, en modo lectura, por todos los usuarios del sistema. Además en el caso de que el sistema no utilize "*shadowing*" de contraseñas podrá saber que usuarios no usan contraseña para acceder al sistema.

3. Además si nuestra máquina tiene activado el servicio `finger` y es accesible desde fuera de nuestra red cualquier usuario podrá conocer los `login` de nuestra máquina. Una vez conocidos los `login` se puede empezar a intentar averiguar las contraseñas:

```
[bender@localhost bender]$ telnet 192.168.0.1 79
Trying 192.168.0.1...
Connected to 192.168.0.1.
Escape character is '^]'.

Login   Name  Tty    Idle  Login Time   Office  Office Phone
icarus        tty3         Apr 14 19:15
jose          tty1   1:38  Apr 14 17:37
jose          pts/0  1:37  Apr 14 17:37
jose          pts/1        Apr 14 17:38
jose          pts/2        Apr 14 18:01
root    root  tty2         Apr 14 19:15
Connection closed by foreign host.
[bender@localhost bender]$
```

   De esta forma hemos obtenido los `login` de los usuarios que estan conectados en el sistema. Cuando obtenemos el mensaje:

```
Escape character is '^]'.
```

pulsamos `return` y obtedremos la información anterior. Si quisieramos obtener más información sobre un determinado usuario:

```
[bender@localhost bender]$ telnet 192.168.0.1 79
Trying 192.168.0.1...
Connected to 192.168.0.1.
Escape character is '^]'.
root
Login: root                             Name: root
Directory: /root                        Shell: /bin/bash
On since Sun Apr 14 19:15 (CEST) on tty2   1 minute 32 seconds idle
No mail.
No Plan.
Connection closed by foreign host.
[bender@localhost bender]$
```

cuando obtenemos el mensaje:

```
Escape character is '^]'.
```

introducimos el `login` del usuario y a continuación el sistema nos dará la información que obtendriamos utilizando `finger login`. Con la salvedad de que no necesitamos tener cuenta en el sistema.

Es una buena medida de seguridad el no activar el demonio `fingerd` ya que facilita demasiada información sobre el sistema.

Si por algún extraño motivo necesitas utilizarlo arrancalo con la opción "*-u*", entonces todas las peticiones del tipo "*finger usuario@maquina*" serán ignoradas. Este tipo de peticiones se utilizan para recoger información sobre un sistema, normalmente con fines algo oscuros ...

También existen otros demonios para `finger` que ofrecen mayor seguridad, como por ejemplo:

cfingerd `http://www.infodrom.org/projects/cfingerd/`

pfinger `http://www.xelia.ch/unix/pfinger/`

5.1.2. Necesidad del uso de buenas contraseñas

Es una practica habitual el utilizar contraseñas faciles de recordar, lo cual estaría muy bien si los usuarios las escogieran con algo de imaginación en lugar de utilizar alguno de los siguientes métodos:

- `login` = pepito y `password` = pepito. Original donde los haya. ;-)
- `login` = pepito y `password` = pepitoXX, donde XX puede ser el día que nació `pepito`, las dos últimas cifras del año en que nació `pepito` o alguna "medida" de la que está orgulloso[1] `pepito`.
- Un `password` basado en la matricula del coche, D.N.I., fecha de nacimiento, nombre de la novia/o, mujer/marido, ...

Todas estas `contraseñas` son malas ya que son facilmente accesibles, estan basadas en datos más o menos públicos.

Es muy importante utilizar buenas contraseñas ya que la elección de contraseñas malas como las anteriores puede hacer que alguien acceda al sistema suplantando nuestra identidad y aunque no tengamos privilegios dentro del sistema un usuario experimentado podría conseguir privilegios de administrador aprovechando descuidos del administrador o fallos de seguridad. En cualquier caso podría borrar TODOS nuestros datos.

Además si la máquina contiene datos confidenciales el atacante podría robarlos y la empresa verse obligada a pagar una cuantiosa multa si los datos se hicieran públicos[2] o perder dinero[3].

[1] Normalmente es la que le gustaría y no la verdadera. ;-)
[2] Ley de protección de datos estadíticos.
[3] Espionaje industrial.

5.1.3. Elección de buenas contraseñas

Una buena contraseña debe cumplir los siguientes requisitos:

1. No ser un dato relacionado con nosotros, matricula del coche, fecha de nacimiento, ...
2. No ser una palabra perteneciente a algún idioma.
3. Debe tener una longitud de al menos diez caracteres.
4. Debe contener letras mayúsculas y minúsculas, números y, si es posible, algún signo de puntuación como puntos y coma, barra de subrayado, ...
5. Facil de recordar, algo más complicado de lograr.
6. Debe cumplir el protocolo **KISS**:

Keep **I**t **S**ecret **S**tupid

5.2. Almacenamiento de contraseñas en GNU/Linux

Como ya vimos en la sección 1.3.3 en la página 14 las contraseñas de los usuarios son almacenadas en el fichero `/etc/passwd`. Hoy en día es raro almacenarlas en este fichero ya que se utiliza un técnica llamada "*shadowing*" de contraseñas.

En cada línea de estos ficheros se encuentra la información referente a un usuario, y en el segundo campo se encuentra almacenada la contraseña de forma ilegible. Mucha gente, erroneamente, piensa que la contraseña esta encriptada, y, que para comprobar la identidad de un usuario **GNU/Linux** desencripta la contraseña y la comprueba.

Esto no es del todo cierto, en el segundo campo no se encuentra la contraseña encriptada, sino un "*hashing*" o "*resumen*" de la contraseña.

Un "*hashing*" es una técnica criptográfica, que no vamos a explicar. Básicamente un "*hashing*" produce una cadena de caracteres, de longitud fija, para cada texto sobre el que la apliquemos. Ha de cumplir varias condiciones:

- Conocido un "*hashing*" ha de ser computacionalmente imposible obtener el texto que lo genero.
- Una buena función "*hashing*" no producirá[4] dos "*hashing*" iguales para dos textos diferentes.
- Será practicamente imposible[5] encontrar aleatoriamente dos textos con el mismo "*hashing*".

5.2.1. Hashing Vs Encriptación

Si almacenaramos las contraseñas de acceso encriptadas también tendríamos que almacenar la clave para desencriptarlas, y el superusuario debería tener acceso a la clave. Si hubiera un fallo en la seguridad, o simplemente el administrador estuviera descontento con la empresa, podría facilitar las claves a alguien ajeno al sistema y comprometer la seguridad.

Utilizando "*hashing*" logramos que nadie dentro del sistema pueda conseguir las contraseñas de los usuarios, ya que las funciones de "*hashing*" no lo permiten (si fueron bien escogidas).

5.2.2. La salt

Si dos usuarios tienen contraseñas identicas y comprobaramos el fichero en el que se almacenan las contraseñas veriamos que el campo en el que se almacenan sería idéntico. Para evitar esto se utiliza algo conocido como "*salt*".

La "*salt*" son doce bits que se eligen aleatoriamente cuando se establece la contraseña del usuario utilizando el comando `passwd`. A la hora de producir el "*hashing*" de la contraseña se utiliza la "*salt*" para alterarlo. De esta forma dos usuarios con la misma contraseña tendrán un "*hashing*" diferente.

En campo en el que se almacena la contraseña del usuario también se almancena la "*salt*". La "*salt*" suelen ser los dos primeros caracteres del campo en el que esta almacenada la contraseña.

[4] Existirá una probabilidad practicamente nula.
[5] Existirá una probabilidad practicamente nula.

5.2.3. ¿Como se comprueba la identidad de los usuarios?

Cuando hacemos `login` y suministramos la contraseña **GNU/Linux** verifica la contraseña de la siguiente forma:

1. Lee la "*salt*" del fichero de contraseñas.
2. Produce el "*hashing*" de la contraseña facilitada por el usuario utilizando la "*salt*" almacenada en el campo de la contraseña del usuario.
3. Comprueba el "*hashing*" obtenido con el almacenado en el fichero de contraseñas.
4. Si coinciden permite el acceso al usuario, en caso contrario lo niega. Además si existe el fichero `/var/log/btmp`[6] registrará el intento fallido de conexión.

5.2.4. ¿Que funciones de hashing utiliza GNU/Linux?

Antiguamente se utilizaba el comando `crypt` para generar los "*hashings*" de las contraseñas. Este comando está basado en D.E.S., es una versión modificada de este algoritmo simétrico de encriptación.

D.E.S. es un algoritmo de encriptación, no de "*hashing*", con lo cual produce encriptaciones. Se modificó el algoritmo para producir "*hashings*" en lugar de encriptaciones.

En la actualidad cuando instalamos una distribución de **GNU/Linux** nos pregunta si queremos utilizar **MD5**. **MD5** es un algoritmo de "*hashing*" más avanzado y seguro que el utilizado en `crypt` y es recomendable su uso ya que ofrece una mayor seguridad.

[6] Apartado 11.2.3 en la página 177.

5.3. Shadowing de contraseñas

Tradicionalmente se almacenaban las contraseñas en el fichero `/etc/passwd`. Este fichero tiene que tener permisos de lectura para todo el mundo del sistema, lo que implica que todo el mundo puede conocer los "*hashing*" de todo el mundo.

Luego un usuario podría dedicarse a realizar ataques de diccionario[7] hasta adivinar alguna clave, por esto es importante elegir buenas contraseñas.

Para evitar esto se empezo a utilizar el "*shadowing*" de contraseñas, que no es ni mas ni menos que almacenar los "*hashings*" de las contraseñas en un fichero que sólo puede leer el superusuario.

5.3.1. Problemas del shadowing

Esto plantea un problema y es que todos aquellos programas que tienen que comprobar la identidad de un usuario, como `su`, no podrán hacerlo ya que estos programas buscan los "*hashings*" de las contraseñas en `/etc/passwd`.

Para solucionar esto hubo que "reajustar" todos estos programas para que buscaran las contraseñas en el fichero adecuado.

El cambiar todos estos programas habría sido una tarea dura, pero gracias a P.A.M. se simplificó bastante esta tarea. P.A.M. permite al administrador de la máquina elegir como se van a autenticar los usuarios y las aplicaciones. Gracias a esta biblioteca es muy fácil y comodo el adaptar los programas a nuevos métodos de autenticación. También ayuda el tener el código fuente de los programas, sin el no sería posible.

[7] Coger una de las muchas herramientas existentes y liarse a generar "*hashings*" hasta encontrar el que coincida con la cuenta de un usuario. Como él ha generado los "*hashings*" conoce el texto, la clave, que lo generó.

5.3.2. El fichero `/etc/shadow`

Este fichero unicamente es accesible por el superusuario y es en el que se almacenan los "*hashings*" de las contraseñas.

Cada línea de este fichero contiene información acerca de un usuario. Por ejemplo:

```
root:$1$S38vbAa7$m8hY6WoZWz1DuBPJoTrtg.:11781:0:99999:7:::
```

Unicamente vamos a explicar los dos primeros campos, el resto `man 5 shadow`:

1. En el primer campo se encuentra el nombre del usuario.
2. En el segundo campo se encuentra el "*hashing*" de la contraseña del usuario.

5.3.3. El fichero `/etc/gshadow`

En el caso de utilizar contraseñas para los grupos si utilizamos "*shadowing*" de contraseñas en lugar de almacenar estas contraseñas en el fichero `/etc/groups` se almacenaran en el fichero `/etc/gshadow`, unicamente accesible por el superusuario.

Capítulo 6

Los sistemas de ficheros en GNU/Linux

En el mundo UNIX todo se entiende como ficheros, debido a esta abstracción se goza de una gran potencia a la hora de configurar y administrar un sistema. No existe el concepto de dispositivo físico, por lo cual es habitual encontrarnos una máquina UNIX que trabaja con un directorio /home, que contiene los directorios de trabajo de los usuarios, pero que está en un disco duro en otra máquina, sin embargo para la máquina UNIX que da servicio a esos usuarios es como si ella misma tuviera ese disco duro.

6.1. Los ficheros estándar

En los sitemas UNIX y en **GNU/Linux** en particular existen tres ficheros estándar:

stdinput o entrada estándar.

stdoutput o salida estándar.

stderr o salida estándar de errores.

6.1.1. La entrada estándar

GNU/Linux entiende como entrada estándar o **stdinput** a aquel dispositivo por el cual se producirán la mayoría de las entradas de datos. Normalmente es el teclado.

Podemos redirigir la entrada estándar utilizando < como vimos en el apartado 3.4 en la página 46.

La entrada estándar tiene por defecto el descriptor "**0**".

6.1.2. La salida estándar

GNU/Linux entiende como salida estándar o **stdoutput** a aquel dispositivo por el cual mostrará el resultado de las ejecuciones de los comandos del usuario, mientras no se le diga lo contrario. Normalmente es la pantalla.

Podemos redirigir la salida estándar utilizando > o >> como vimos en el apartado 3.4 en la página 46.

La salida estándar tiene por defecto el descriptor "**1**".

6.1.3. La salida estándar de errores

Linux entiende como salida estándar de errores o **stderr** a aquel dispositivo por el cual mostrará los errores que se cometan. Normalmente es la pantalla.

Por ejemplo si ejecutamos un `find / -name pepe > res` como un usuario normal veremos que nos saldrán por pantalla mensajes del tipo:

```
find: /etc/httpd/conf/ssl.crl: Permiso denegado
```

los cuales no irán a parar al fichero `res` que es donde le hemos dicho que vayan los resultados de la busqueda. La explicación es muy sencilla, los resultados de la busqueda son los que irán a la salida estándar, luego irán al fichero `res` ya que hemos redirigido esa salida hacía ese fichero, pero los errores van hacía la salida estándar de errores que no está redirigida y como por defecto es el monitor, los mensajes de error saldrán por él.

La entrada estándar de errores tiene por defecto el descriptor "**2**". Mediante el uso de este descriptor podemos redirigir la salida estándar de errores:

```
[jose@localhost jose]$ find / -name pepe > res 2> /dev/null
[jose@localhost jose]$
```

con esto almacenaremos en el fichero `res` donde estan todos los ficheros llamados `pepe` y que esten en las partes del sistema a las que tengamos acceso. Cuando el comando `find` intente acceder a una parte del sistema a la que no tenga acceso se producirá un mensaje de error que debería ser mostrado

en la salida estándar de errores, pero en este caso lo hemos redirigido hacía `/dev/null` que es un agujero sin fondo, todo lo que se escriba en ese fichero será borrado de forma inmediata.

También podríamos redigir la salida estándar de errores hacía la salida estándar mediante:

```
find / -name Adm-Basica.pdf >pepe 2>&1
```

donde 1 es el descriptor de la salida estándar. Con este comando los resultados de la busqueda irían al fichero `pepe` y los errores también ya que fueron redirigidos hacía la salida estándar y esta estaba redirigida al fichero `pepe`.

Este tipo de redirecciones son muy útiles cuando compilamos proyectos grandes, ya que podemos guardar el resultado de la compilación en un fichero y los errores en otro.

6.2. Organización de los directorios

Si hacemos un `ls -l /` veremos varios directorios, todos ellos estan presentes en casi todas las distribuciones de **GNU/Linux** y otros `UNIX`:

```
bin
boot
dev
etc
home
lib
lost+found
mnt
opt
proc
root
sbin
tmp
usr
var
```

6.2.1. El directorio `bin`

En este directorio se encuentran los comandos esenciales para el funcionamiento del sistema:

```
chgrp, chmod, cp, gzip, kill, mount, umount, tar
```

6.2.2. El directorio `boot`

En este directorio se encuentran los ficheros necesarios para el arranque del sistema.

6.2.3. El directorio `dev`

En este directorio se encuentran los ficheros de dispositivos:

```
hda1, hdb2, hdc3, hdd4, sda1, fd0, lp1
```

6.2.4. El directorio `etc`

En este directorio se encuentran los archivos de configuración del sistema.

6.2.5. El directorio `home`

En este directorio se encuentran los directorios de trabajo de los usuarios normales.

6.2.6. El directorio `lib`

En este directorio se encuentran las librerías básicas del sistema.

6.2.7. El directorio `lost+found`

Este directorio se encuentra si tenemos un sistema de archivos ext2 y se utiliza para almacenar la información recuperada cuando ha habido una inconsistencia en el sistema de ficheros.

6.2.8. El directorio `mnt`

En este directorio se montan otros sistemas de ficheros.

6.2.9. El directorio `opt`

En este directorio se suelen instalar programas que normalmente no vienen con la distribución de **GNU/Linux**.

6.2.10. El directorio `proc`

En este fichero se encuentra toda la información del sistema:

- Sistemas de ficheros montados.
- Interrupciones utilizadas.
- Información sobre la memoria.
- Carga del sistema.
- ...

6.2.11. El directorio `root`

Este directorio es el directorio de trabajo del `root`.

6.2.12. El directorio `sbin`

En este directorio se encuentran los comandos de administración del sistema. Unicamente accesibles por el `root`.

6.2.13. El directorio `tmp`

En este directorio se encuentran los archivos temporales del sistema.

6.2.14. El directorio `usr`

En este directorio se encuentran las aplicaciones para los usuarios.

6.2.15. El directorio `var`

En este directorio hay un poco de todo:

- Archivos de "*spooling*", colas de impresión, correo electrónico, de los demonios `at` y `cron`, ...
- Las páginas web de nuestro servidor de páginas web, si lo tenemos.
- Los archivos de `log` del sistema.
- ...

6.3. ¿Qué es un sistema de ficheros?

Todos nuestros archivos están almacenados en un soporte físico, discos duros, disquetes, CD-Rom's, ... Para que el sistema operativo sepa donde buscar un determinado fichero cuando lo necesita es necesario que toda esta información esté almacenada de "alguna" forma organizada para que sea facilmente accesible.

Un sistema de ficheros no es nada "más" que una forma de organizar los ficheros en el disco duro.

6.3.1. ¿Qué sistema de ficheros utiliza GNU/Linux?

GNU/Linux normalmente utiliza los sistemas de archivos:

- **ext2** o extended filesystem version II.
- **ext3** o extended filesystem version III.

además existen otros sistemas de ficheros para **GNU/Linux**:

- ReiserFS (Hans Reiser).
- XFS "*Extended File System*" (Silicon Graphics).
- JFS "*Journaling File System*" (I.B.M.).
- LFS "*Linux Log File System*" basado en ext2.

6.3.2. ¿Puede GNU/Linux "entender" sistemas de ficheros de otros Sistemas Operativos?

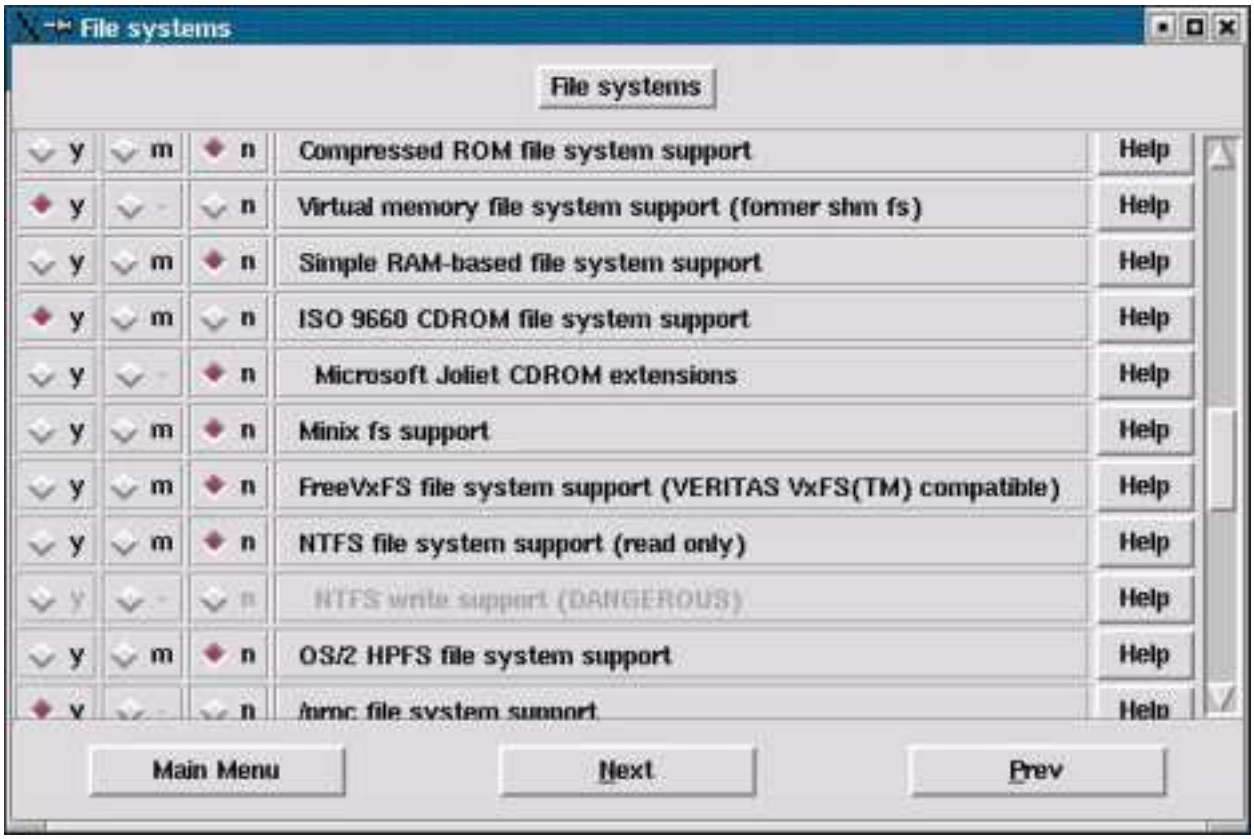
Pues sí, **GNU/Linux** es capaz de leer y escribir en sistemas de ficheros de otros sistemas operativos, como por ejemplo en los sistemas de ficheros de Windows. **GNU/Linux** puede entender:

- MS-DOS (lectura y escritura).
- FAT 16 y FAT 32 de Windows (lectura y escritura).
- NTFS de Windows NT (sólo lectura).
- FFS o **F**ast **F**ile **S**ystem, un sistema de ficheros utilizado por los ordenadores AMIGA.
- HPFS o **H**igh **P**erfomance **F**ile **S**ystem, el sistema de ficheros de OS/2.
- MINIX sistema operativo del que surgió **GNU/Linux**.
- iso9660, CD-Roms.

6.3.3. ¿Que sistemas de ficheros "entiende" mi GNU/Linux?

Aunque **GNU/Linux** puede entender muchos sistemas de ficheros no quiere decir que el **GNU/Linux** que tengo en casa sea capaz de enterder un determinado sistema de ficheros.

Cuando compilamos el nucleo de **GNU/Linux** se añade soporte para los sistemas de ficheros que vamos a utilizar, seran estos sistemas de ficheros los que nuestra máquina **GNU/Linux** será capaz de entender. Podemos ver los sistemas de archivos soportados por el nucleo o kernel de nuestra máquina en el fichero `/proc/filesystem`.



6.3.4. Los i-nodos, esos grandes desconocidos

Los i-nodos son como una especie de indice que nos indica donde tenemos que buscar un fichero en el sistema de ficheros. Aunque se tiende a asociar i-nodos con ficheros esta asociación no es del todo correcta. Si bien es cierto que todo fichero tiene asociado un i-nodo, de forma única.

Un i-nodo contiene toda la información referente a un fichero. Un i-nodo también puede contener información de los enlaces símbolicos, sockets y dispositivos especiales, los cuales son parte de un sistema de ficheros, pero no tienen asociado ningún fichero.

6.3.5. El Virtual File System o VFS

Aunque parece sencillo el hecho de poder acceder a varios sistemas de ficheros diferentes es bastante complicado el poder hacerlo, ya que cada sistema de ficheros es de una forma y tiene sus propias características.

GNU/Linux puede acceder a varios sistemas de ficheros diferentes gracias al VFS o Virtual File System, que es un "*sistema de archivos genérico*". La forma más facil de explicarlo es recurrir a la programación orientada a objetos. Podemos entender el VFS como una clase y los diferentes sistemas de ficheros como clases derivadas del VFS. El VFS tendría unas características comunes a todos los sistemas de ficheros, propiedades, y unos métodos comunes a todos ellos, operaciones que se pueden realizar sobre cada uno, luego cada sistema de ficheros como el ext2, vfat, ... tendría esas propiedades y métodos heredadas del VFS y las suyas propias.

6.4. Dispositivos de almacenamiento en GNU/Linux

Antes de acceder a un sistema de ficheros tenemos que acceder al dispositivo físico en el que reside, **GNU/Linux** tiene una forma de organizar estos dispositivos bastante diferente a la de otros sistemas operativos no `UNIX`.

Para **GNU/Linux** todos los dispositivos de almacenamiento seran un fichero que estará en el directorio `/dev`.

6.4.1. Disqueteras (/dev/fd?)

Para **GNU/Linux** las disqueteras serán un archivo en `/dev` que tendrá por nombre:

`/dev/fd?`

donde el carácter "`?`" es un número que nos indica el número de disquetera:

- `/dev/fd0`, primera disquetera.
- `/dev/fd1`, segunda disquetera.
- `/dev/fd2`, tercera disquetera.
- `/dev/fd`n, n-ésima disquetera.

6.4.2. Dispositivos IDE (/dev/hd*tn*)

En los PC's existen dos canales IDE y en cada canal IDE se pueden conectar dos dispositivos:

- Uno primario.
- Uno secundario.

con lo cual en un PC podemos tener:

- Dos dispositivos primarios.
- Dos dispositivos secundarios.

Estos dispositivos pueden ser discos duros o CD-Roms. Dentro de cada disco duro podemos tener una o más particiones.

GNU/Linux entiende los dispositivos IDE como ficheros y esos ficheros estarán dentro del directorio `/dev` y tendremos que serán de la forma:

`/dev/hd`*tn*

donde:

t nos indica el canal IDE al que está conectado el dispositivo. Puede tomar los siguientes valores:

- **a**, dispositivo primario conectado al primer canal IDE.
- **b**, dispositivo esclavo conectado al primer canal IDE.
- **c**, dispositivo primario conectado al segundo canal IDE.
- **d**, dispositivo esclavo conectado al segundo canal IDE.

n nos indica la partición dentro del disco duro:

- **1**, primera partición dentro del dispositivo.
- **2**, segunda partición dentro del dispositivo.
- **3**, tercera partición dentro del dispositivo.

Por ejemplo:

/dev/hd*a*1 , primera partición dentro del dispositivo primario conectado al primer canal IDE.

/dev/hd*b*3 , tercera partición dentro del dispositivo esclavo conectado al primer canal IDE.

/dev/hd*c* , dispositivo primario conectado al segundo canal IDE. La ausencia de número de partición nos indica que es un CD-Rom.

/dev/hd*d*2 , segunda partición dentro del dispositivo esclavo conectado al segundo canal IDE.

6.4.3. Dispositivos SCSI (/dev/sd*tn*)

Normalmente los PC's no traen controladoras SCSI en placa, por lo que si necesitamos utilizar un dispositivo de este tipo necesitaremos comprar una y "pincharla" en nuestra placa. Suelen ser tarjetas PCI aunque también las hay ISA. Después de "pincharla" necesitamos cargar el modulo correspondiente para hacerla funcionar y una vez esté funcionando ya podemos acceder a todos los dispositivos que esten conectados a ella.

Los dispositivos que se pueden conectar a una controladora de este tipo dependen del tipo de controladora que sea, las hay que admiten un único dispositivo y también que admiten hasta quince dispositivos.

GNU/Linux entiende estos dispositivos como un fichero en `/dev` y su nombre será de la forma:

`/dev/sd`*tn*

donde:

t indica el número del dispositivo SCSI:

- `a`, primer dispositivo SCSI.
- `b`, segundo dispositivo SCSI.
- `c`, tercer dispositivo SCSI.
- ...

n nos indica la partición dentro del disco duro:

- **1**, primera partición dentro del dispositivo.
- **2**, segunda partición dentro del dispositivo.
- **3**, tercera partición dentro del dispositivo.

Por ejemplo:

/dev/sd*a*1 , primera partición dentro del primer dispositivo SCSI.

/dev/sd*b*3 , tercera partición dentro del segundo disspositivo SCSI.

/dev/sd*c* , tercer dispositivo SCSI.

/dev/sd*d*2 , segunda partición dentro del cuarto dispositivo SCSI.

6.4.4. Unidades de cinta

Podemos encontrar varios tipos de unidades de cinta:

/dev/st*n* *n*-ésima unidad de cinta SCSI.

/dev/ft*n* *n*-ésima unidad de cinta.

6.5. Como acceder a un sistema de ficheros

Ya hemos visto que **GNU/Linux** es capaz de acceder a otros sistemas de ficheros, pero ¿como hacerlo?.

6.5.1. El comando `mount`

Para acceder a un sistema de ficheros debemos montarlo, es decir hacerlo accesible. Para ello debemos darle al nucleo cierta información sobre el sistema de ficheros:

- Que tipo de sistema de ficheros es. `ext2`, `ext3`, `nfs`, `vfat`, ...
- En que dispositivo se encuentra. `/dev/hda1`, `/dev/hdc`, `/dev/sda1`, ...
- Donde queremos montarlo. Este es el punto de montaje, será un directorio y para acceder al sistema de ficheros montado en ese directorio bastará con entrar en el.

Supongamos que tenemos un CD-Rom que está conectado como primer dispositivo primario del segundo canal IDE, la orden para montar el dispositivo será:

```
[root@localhost root]# mount -t iso9660 /dev/hdc /mnt/cdrom
```

El comando `mount` sirve para montar sistemas de ficheros, las opciones que le pasamos en la linea de comandos significan:

-t iso9660 indica el tipo de sistema de archivos que tiene el dispositivo al que queremos acceder.

/dev/hdc indica que dispositivo es. En este caso el dispositivo primario del segundo canal IDE.

/mnt/cdrom indica donde vamos a montar el dispositivo. Después de montarlo si queremos acceder a la información del dispositivo tendremos que ir al directorio `/mnt/cdrom`.

6.5.2. El comando `umount`

Cuando no necesitamos utilizar un dispositivo podemos desmontarlo. Por ejemplo, supongamos que necesitamos las fuentes del nucleo de **GNU/Linux** ya que queremos compilarlo para tener soporte en nuestro nucleo de un nuevo y maravilloso sistema de ficheros. Dichos fuentes estan en un CD-Rom, entonces introducimos el CD-Rom en la unidad y lo montamos:

```
[root@localhost root]# mount -t iso9660 /dev/hdc /mnt/cdrom
```

una vez montado accedemos al CD y copiamos los fuentes del nucleo a nuestro disco duro. Una vez que ya los hemos copiado no nos hace falta el CD-Rom, con lo cual podemos desmontarlo y dejar la unidad libre para montar otro CD-Rom:

```
[root@localhost root]# umount /mnt/cdrom
```

Ahora ya tenemos libre la unidad y podemos montar otro CD-Rom para, por ejemplo, tener acceso a algún documento que nos indique como hemos de compilar el nucleo.

Es necesario que ningún recurso del sistema este accediendo a un dispositivo montado para poder desmontarlo:

```
[root@localhost cdrom]# pwd
/mnt/cdrom
[root@localhost cdrom]# umount /mnt/cdrom
umount: /mnt/cdrom: dispositivo ocupado
[root@localhost cdrom]#
```

Si estamos dentro de `/mnt/cdrom` que es el directorio en el que hemos montado el CD-Rom e intentamos desmontar el CD-Rom veremos un mensaje de error:

```
umount: /mnt/cdrom: dispositivo ocupado
```

no nos dejará desmontar el CD-Rom ya que hay un recurso que lo está utilizado. Si salimos del directorio `/mnt/cdrom` podremos desmontar el CD-Rom, si ningún otro recurso lo está utilizando.

Es muy **IMPORTANTE** desmontar un sistema de ficheros antes de quitar el medio físico, por ejemplo antes de quitar un disquete de la disquetera hay que desmontar el disco. Igual pasa al apagar un equipo, antes de apagarlo físicamente hay que pararlo utilizando `shutdown`:

```
[root@localhost root]# shutdown -h now
```

La razón es que al ser **GNU/Linux** un sistema multitarea cuando realizamos una operación de escritura en realidad no se realiza en ese momento, se almacena en un buffer[1] y cuando el sistema operativo tiene que realizar operaciones de escritura se encarga de ir escribiendo todas las pendientes. Al realizar un `umount` sobre un medio físico el sistema operativo comprueba las operaciones de escritura pendientes sobre ese medio físico y las realiza, igual pasa con el `shutdown`. Si quitamos un disco de la disquetera sin realizar el `umount` probablemente no se hayan almacenado todos los datos y perdamos datos, lo mismo pasaría si apagamos el ordenador sin utilizar el comando `shutdown`. En estos casos se dice que se produce una "*inconsistencia en el sistema de ficheros*[2]"

6.5.3. El comando `df`

Este comando nos da información sobre los sistemas de archivos montados en el sistema:

```
[bender@localhost bender]$ df
Filesystem    1k-blocks      Used Available Use% Mounted on
/dev/hda2       1313600   1224452     22420  99% /
none              31172         0     31172   0% /dev/shm
/dev/md0        2015984     18584   1894992   1% /mnt/raid
/dev/hda6        624752    505552    119200  81% /mnt/vfat
[bender@localhost bender]$
```

[1] Buffer Cache.

[2] Ver apartado 6.8.1 en la página 107.

Podemos ver la información referente a un determinado sistema de ficheros pasandole como argumento el dispositivo, `/dev/md0`, o el punto de montaje, `/mnt/raid`:

```
[bender@localhost bender]$ df /mnt/raid
Filesystem    1k-blocks      Used Available Use% Mounted on
/dev/md0        2015984     18584   1894992   1% /mnt/raid
[bender@localhost bender]$
```

Por defecto muestra la información en bloques de 1 kb, podemos hacer que la información salga en un formato más comprensible utilizando el flag "`-h`":

```
[bender@localhost bender]$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda2             1.3G  1.2G   65M  95% /
none                   30M     0   30M   0% /dev/shm
/dev/md0              1.9G   47M  1.7G   3% /mnt/raid
/dev/hda6             610M  494M  116M  81% /mnt/vfat
[bender@localhost bender]$
```

También podemos ver la información referente a los inodos de los sistemas de ficheros:

```
[bender@localhost bender]$ df -i
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/hda2             166848   78975   87873   48% /
none                    7793       1    7792    1% /dev/shm
/dev/md0              256512    1056  255456    1% /mnt/raid
/dev/hda6                  0       0       0    -  /mnt/vfat
[bender@localhost bender]$
```

6.5.4. El comando `du`

Este comando se utiliza para conocer la cantidad de disco utilizada por ficheros, directorios, ... Si a este comando no se le pasa como argumento ningún directorio indica el tamaño en disco que ocupa el directorio actual, incluyendo subdirectorios:

```
[jose@localhost conf]$ du
84      ./user
44      ./ayuda
36      ./sistArchivos/.xvpics
808     ./sistArchivos
44      ./shell
12      ./login
16      ./runlevels
8       ./procesos
24      ./automatizacion
8       ./inet
2204    .
[jose@localhost conf]$
```

la información que ofrece es en kb. Utilizando el flag "`-h`" podemos ver la información de una forma más comprensible:

```
[jose@localhost conf]$ du -h
84k     ./user
44k     ./ayuda
36k     ./sistArchivos/.xvpics
808k    ./sistArchivos
44k     ./shell
12k     ./login
16k     ./runlevels
8.0k    ./procesos
24k     ./automatizacion
8.0k    ./inet
2.2M    .
[jose@localhost conf]$
```

Si le pasamos como argumento un directorio en concreto nos ofrecerá la cantidad de disco que ocupa dicho directorio junto con sus subdirectorios. Si unicamente queremos conocer la cantidad de espacio utilizada en disco podemos utilizar el flat "`-s`":

```
[jose@localhost conf]$ du -hs
2.2M    .
[jose@localhost conf]$
```

6.5.5. ¿Quién puede montar sistemas de ficheros?

Unicamente el superusuario o `root` puede montar o desmontar sistemas de ficheros. La razón es que normalmante un sistema **GNU/Linux** tiene montados varios sistemas de ficheros y si alguien desmontara uno de ellos el sistema no funcionaría correctamente. Por ejemplo, es muy habitual que el directorio `/home`, que es donde residen los archivos de los usuarios, este en una partición o disco diferente del resto de datos, o quizá en un ordenador diferente. Si cualquier usuario pudiera montar y desmontar sistemas de ficheros a su antojo podría desmontar el directorio `/home` y los usuarios se quedarían sin poder utilizar el sistema ya que no tendrían acceso a sus directorios de trabajo.

Debido a que **GNU/Linux** además de funcionar como servidor puede ser utilizado como estación de trabajo es posible que usuarios normales monten y desmonten sistemas de ficheros, por ejemplo CD-Roms, disquetes, ...

El superusuario o `root` puede autorizar a los usuarios normales a que monten y desmonten determinados sistemas de ficheros. Por ejemplo podría autorizar a los usuarios normales para que monten y desmonten la disquetera y puedan llevarse o traerse ficheros.

6.5.6. El fichero `/etc/fstab`

En este fichero reside la información acerca de los sistemas de ficheros que existen en la máquina, además de cuales se montan cuando se arranca la máquina y de quien puede montarlos. Un ejemplo de este archivo podría ser:

```
LABEL=/         /              ext2    defaults         1 1
none            /proc          proc    defaults         0 0
/dev/hda5       swap           swap    defaults         0 0
/dev/cdrom      /mnt/cdrom     iso9660 noauto,user,ro   0 0
/dev/fd0        /mnt/floppy    auto    noauto,user      0 0
/dev/hda6       /mnt/vfat      auto    noauto,user      0 0
/dev/sda1       /mnt/scsi      ext2    auto             0 0
```

Podemos ver que la información está distribuida en seis columnas:

1. En la primera columna estará el dispositivo físico en el que reside el sistema de ficheros.

2. En la segunda columna estará el punto del montaje.

3. En la tercera columna estará el tipo de sistema de ficheros:

   - **ext2**.
   - **ext3**.
   - **vfat** (Windows 9* con nombres largos).
   - **swap** para particiones de swaping.
   - **iso9660** para CD-Rom.
   - **auto** para que determine automaticamente el tipo de sistema que es al intentar montarlo.
   - Cualquier otro sistema de ficheros que soporte el nucleo.

4. En la cuarta estan las opciones de montaje, separadas por comas, con las cuales se montará el sistema de ficheros:

   - **auto**, el sistema de ficheros se intentará montar al arrancar.
   - **noauto**, el sistema de ficheros no se intentará montar al arrancar.
   - **exec**, se permite ejecutar ficheros binarios en ese sistema de ficheros.
   - **noexec**, no permite ejecutar ficheros binarios en ese sistema de ficheros.

- **user**, cualquier usuario podrá montar y desmontar este sistema de ficheros.
- **nouser**, sólo el `root` podrá montar ese sistema de ficheros.
- **ro**, se montará en modo sólo lectura.
- **rw**, con permisos de lectura y escritura.
- **quota**, con cuotas de disco.
- **noquota**, sin cuotas de disco.
- **suid**, con acceso a SUID.
- **nosuid**, sin acceso a SUID.
- **defaults**, monta el sistema de fichero con las opciones por defecto[3].

Para ver todas las opciones de montaje ver la página del manual del comando `mount`.

5. En la quinta columna añade información para `dump`, si encuentra un cero indica que no es necesario hacer un `dump` del sistema de ficheros. En caso de no encontrar nada asume que es un cero.

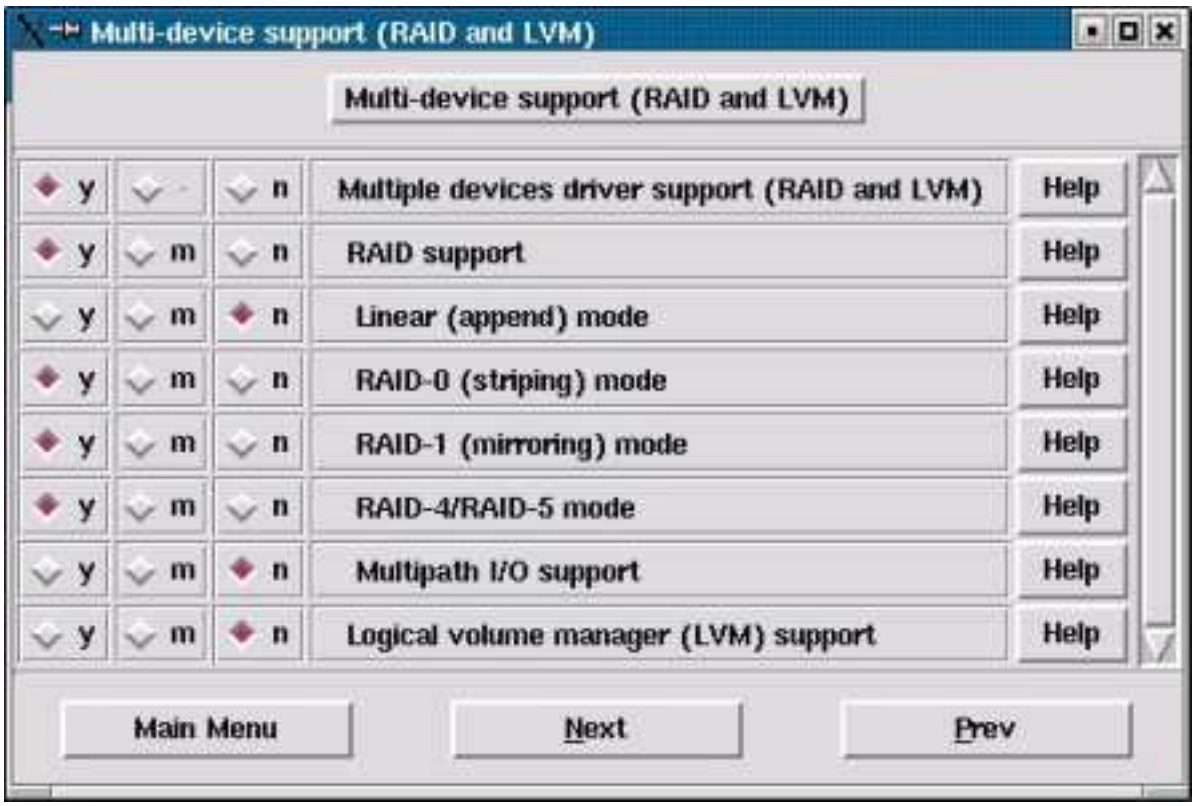
6. En la sexta columna se indica a `fsck` el orden en el que chequeará los sistemas de ficheros al arrancar. Si hay un cero no chequeará el sistema de ficheros al arrancar.

[3]Ver la página del manual del comando `mount`.

6.6. Sistemas RAID

GNU/Linux soporta el estándar RAID[4]. Este estándar permite gestionar varios discos duros como si fueran uno solo y conseguir de esta forma un mayor rendimiento y/o protección contra fallos.

Los sistemas RAID pueden funcionar por software o hardware. Para realizar RAID por software es necesario que el nucleo de **GNU/Linux** esté compilado con soporte para RAID:



[4]Redundant Array of Inexpensive Disks o Array Redundante de Discos Asequibles.

Los sistemas RAID tienen varios niveles:

RAID 0 o Data Striping Without Parity (DSA).

RAID 1 o Mirrowed Disk Array (MDA).

RAID 2 o Hamming Code for Error Correction.

RAID 3 o Parallel Disk Array (PDA).

RAID 4 o Independient Disk Array (IDA).

RAID 5 o Independient Disk Array (IDA).

La ventaja que tienen este tipo de sistemas, dependiendo del nivel de RAID escogido, es que si uno de los discos falla la unidad continua funcionando ya que inmediatamente es sustituido por otro. Además la reconstrucción de los datos del disco que ha fallado se hace de forma automática.

6.6.1. Niveles RAID soportados por GNU/Linux

Antes de trabajar con RAID asegurate de que tienes una versión del nucleo 2.2.x o 2.0.x parcheada para funcionar con las "*raidtools 0.90*". **GNU/Linux** soporta los siguientes niveles de RAID:

- **Linear Mode**, dos o más discos son utilizados como si fueran uno. Empezando a escribir en el primero y cuando este esté lleno se empieza con el segundo, ...
- **RAID 0**, funciona igual que el modo anterior, pero las operaciones de lectura y escritura son hechas en paralelo en cada disco.
- **RAID 1**, en este nivel se duplican todos los datos en los discos. De este modo todos los datos estan duplicados en todos y cada uno de los discos (también es conocido como "*mirroring*").
- **RAID 4**, en este nivel no se duplican los datos, sino que utiliza un disco para guardar datos de paridad y detectar errores. Si un disco falla podemos utilizar la información de paridad para reconstruir los datos, si fallan dos discos perdemos todos los datos.
- **RAID 5**, este nivel es igual que el RAID 4 pero la información de paridad se reparte entre los discos que forman el RAID en lugar de estar toda almacenada en uno solo.

6.6.2. Spare Disks

Estos discos son discos de repuesto, en el momento que un disco del sistema RAID falla lo sustituyen y se regeneran todos los datos en ellos. Esto hace que el sistema continue funcionando y no se vea afectado por la perdida de datos. Aunque durante la regeneración de los datos perdidos se incrementa la carga del sistema :-(.

6.6.3. El fichero `/etc/raidtab`

En este fichero se guarda la configuración del sistema RAID. Los dispositivos RAID en **GNU/Linux** se especifican como `/dev/md*`, es decir:

- `/dev/md0` es el primer dispositivo RAID.
- `/dev/md1` es el segundo dispositivo RAID.
- `/dev/md2` es el tercer dispositivo RAID.
- ...

Un ejemplo del fichero `/etc/raidtab` para RAID 1 sería:

```
raiddev /dev/md0         # DISPOSITIVO
        raid-level     1 # NIVEL DE RAID
        nr-raid-disks  4 # NUMERO DE DISCOS DEL RAID
        nr-spare-disks 2 # NUMERO DE DISCOS DE REPUESTO
        chunk-size     4
        persistent-superblock 1
        device        /dev/sda1 # DISPOSITIVO DEL PRIMER DISCO
        raid-disk      0         # NUMERO DEL DISCO DENTRO DEL RAID
        device         /dev/sdb1 # DISPOSITIVO DEL SEGUNDO DISCO
        raid-disk      1         # NUMERO DEL DISCO DENTRO DEL RAID
        device         /dev/sdc1 # DISPOSITIVO DEL TERCER DISCO
        raid-disk      2         # NUMERO DEL DISCO DENTRO DEL RAID
        device         /dev/sdd1 # DISPOSITIVO DEL CUARTO DISCO
        raid-disk      3         # NUMERO DEL DISCO DENTRO DEL RAID
        device         /dev/hdb1 # DISPOSITIVO DEL PRIMER DISCO DE REPUESTO
        spare-disk     0         # NUMERO DEL DISCO DENTRO DEL RAID
        device         /dev/hdc1 # DISPOSITIVO DE SEGUNDO DISCO DE REPUESTO
        spare-disk     1         # NUMERO DEL DISCO DENTRO DEL RAID
```

Para crear el RAID tendremos que hacer lo siguiente:

```
[root@localhost root]# mkraid /dev/md0
```

esto crearía el RAID y para activarlo:

```
[root@localhost root]# raidstart /dev/md0
```

Para pararlo:

```
[root@localhost root]# raidstop /dev/md0
```

Al arrancar el ordenador automaticamente empezaría a funcionar el RAID. Para montar un dispositivo de este tipo tendremos que hacerlo mediante el dispositivo RAID, es decir:

```
[root@localhost root]# mount -t ext2 /dev/md0 /mnt/raid
```

en lugar de hacerlo:

```
[root@localhost root]# mount -t ext2 /dev/sda1 /mnt/raid
```

Si queremos ver el estado de los dispositivos RAID:

```
[root@localhost root]# cat /proc/mdstat
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 hdb2[2] sda1[1] hdb1[0]
      1693312 blocks [2/2] [UU]

unused devices: <none>
[root@localhost root]#
```

6.7. Creación de sistemas de ficheros

Para poder escribir en un medio físico, ya sea un disquete o una partición en el disco duro hay que darle formato antes, es decir, tenemos que crear el sistema de ficheros.

Hay veces que es recomendable el dividir el espacio disponible en varias "*particiones*", es decir, dividir el disco duro en varios "*discos duros*" más pequeños.

6.7.1. El comando `fdisk`

Existen varias razones para dividir un disco duro:

- Por ejemplo si nuestro sistema operativo no gestiona discos duros mayores de un determinado tamaño. Por ejemplo Windows 95 y los nucleos antiguos de **GNU/Linux** no gestionaban discos duros mayores de 2 GigaBytes. La solución era crear particiones de como mucho 2 Gigas en el disco duro y de esta forma podiamos disponer de todo el disco duro.

- Para no perder espacio. Con el sistema de archivos de Windows 95 en particiones "*grandes*" se perdía mucho espacio en disco debido a las unidades de asignación. Por cada fichero se almancena información, nombre, cuando fue modificado por última vez, posición en el disco. Si se encuentra al final del disco la dirección de su posición será más larga que si se encuentra al principio. Un ejemplo muy escalofriante de esto es que en una partición de 1.5 Gigas en Windows 95 se pueden llegar a perder 700 Megas. Doy fe de ello, desde ese momento no he vuelto a crear particiones para Windows 95 de más de 800 Megas.

- Por ejemplo si en nuestra máquina hay varios sistemas operativos instalados, como por ejemplo **GNU/Linux**, OS/2, Windows, ... Necesitariamos, como mínimo, una partición para cada sistema operativo.

- Por motivos de seguridad, supongamos que todo el sistema está en una sola partición. Si no tenemos establecidas cuotas de usuarios o las tenemos mal establecidas es posible que un usuario llene todo el espacio libre del sistema (excepto el 5 % que está normalmente reservado para el `root`), con lo cual el resto de usuarios no podría utilizar el sistema y el sistema no funcionaría correctamente ya que no dispondría de espacio para escribir datos como los "*logs*" u otra información administrativa.

  Sin embargo si el directorio `/home` estuviera en una partición distinta del resto y algún usuario llenara todo el espacio libre, unicamente llenaría el espacio libre de la partición en la que estuviera el directorio `/home`, pudiendo el resto del sistema funcionar correctamente, ofreciendo páginas web, acceso a base de datos, ...

Con el comando `fdisk` podemos reparticionar discos duros. Obviamente el único usuario en el sistema que debería tener acceso a este comando será el `root`, ya que tener acceso a este comando implica el poder modificar, borrar, particiones.

Para utilizar este comando sobre un disco duro hay que pasarle como argumento el dispositivo que se quiere particionar:

```
[root@localhost root]# fdisk /dev/hdb
Comando (m para obtener ayuda):
```

a partir de aquí podemos modificar las particiones en el dispositivo `/dev/hdb`, esclavo del primer IDE. Hay que tener mucho cuidado al utilizar este comando ya que podemos perder información y no habrá posibilidad de recuperación.

Sería una buena idea que consultaras la página del manual de este comando.

6.7.2. El comando `mkfs`

Una vez que tenemos creadas particiones tenemos que crear el sistema de ficheros (formatearlas), para ello utilizamos el comando `mkfs`:

```
[root@localhost root]#  mkfs -t ext2 /dev/fd0
mke2fs 1.23, 15-Aug-2001 for EXT2 FS 0.5b, 95/08/09
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
184 inodes, 1440 blocks
72 blocks (5.00\%) reserved for the super user
First data block=1
1 block group
8192 blocks per group, 8192 fragments per group
184 inodes per group

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 20 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@localhost root]#
```

Lo que hemos hecho ha sido crear un sistema de ficheros tipo "*ext2*" en un disquete. Para crear otros sistemas de archivos tenemos que cambiar el flag "*ext2*" por:

msdos para crear un sistema de ficheros para MS-DOS.

vfat para crear un sistema de ficheros para Windows 95 (con nombres largos).

minix para crear un sistema de ficheros para Minix.

bfs para crear un sistema de ficheros para SCO.

...

Para poder utilizar este comando es necesario tener privilegios de `root` y que el sistema haya sido previamente desmontado. Es posible "*reformatear*" un sistema de archivos con otro o con el mismo sistema de ficheros.

Muchas veces existen los siguientes comandos para crear sistemas de ficheros:

mkfs.ext2 equivalente a `mkfs -t ext2`.

mkfs.msdos equivalente a `mkfs -t msdos`.

...

Sería una buena idea que consultaras la página del manual de este comando.

Las ventajas de instalar un sistema en varias particiones de disco son:

- Una gestión más sencilla de las copias de seguridad y actualizaciones de software.
- Poder controlar como se monta cada sistema de archivos. Puede ser necesario permitir el uso de permisos SUID en una parte del sistema, pero puede ser peligroso permitirlo en otra parte. Montando ambas partes en particiones separadas se puede permitir el uso de permisos SUID en aquella parte en la que sean necesarios y no permitirlo en el resto.

6.7.3. El superbloque y el comando `tune2fs`

Este comando está reservado al `root` y se utiliza para obtener información de un sistema ext2 y para ajustar algunos de sus parámetros, como después de cuantas operaciones de montaje chequeará el sistema de ficheros.

El superbloque es un bloque especial que contiene la información referente a un sistema de ficheros, para ver esa información:

```
[root@localhost root]# tune2fs -l /dev/hda2
tune2fs 1.23, 15-Aug-2001 for EXT2 FS 0.5b, 95/08/09
Filesystem volume name:   /
Last mounted on:          <not available>
Filesystem UUID:          6c1b1850-0853-11d6-9ef1-9c1c32a8061e
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      filetype sparse_super
Filesystem state:         not clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              166848
Block count:              333648
Reserved block count:     16682
Free blocks:              33524
Free inodes:              87867
First block:              0
Block size:               4096
Fragment size:            4096
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         15168
Inode blocks per group:   474
Last mount time:          Tue Apr  2 21:07:47 2002
Last write time:          Tue Apr  2 23:26:43 2002
Mount count:              25
Maximum mount count:      38
Last checked:             Sun Mar 24 21:02:56 2002
Check interval:           15552000 (6 months)
Next check after:         Fri Sep 20 22:02:56 2002
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               128
[root@localhost root]#
```

6.8. Reparación de sistemas de ficheros

Muchas veces es necesario la reparación de los sistemas de ficheros. Todos los que alguna vez nos hemos tenido que pegar con los sistemas Windows conocemos las herramientas `chkdsk` y `scandisk`. Del uso continuado de un sistema de ficheros o de su incorrecto uso se puede tener problemas en un sistema de ficheros que derive en la perdida de datos.

En esta sección supondremos que no estamos utilizando sistemas de ficheros de journaling[5].

6.8.1. Inconsistencias en el sistema de ficheros

Como ya hemos dicho **GNU/Linux** es un sistema multitarea y multiusuario. Ello implica que en el sistema se esten realizando "*simultaneamente*" varias tareas dandonos la impresión de que todos los recursos del sistema estan siendo utilizados por nosotros.

Cuando se realiza una operación de escritura en realidad no se escribe en el disco, dicha operación es almacenada en un buffer y cuando el sistema determina que ha llegado la hora de escribir los datos a disco los escribe. Por ejemplo, si tenemos montado un disco y copiamos un archivo de 800 kb en él veremos que se hace a una velocidad increiblemente rápida, si a continuación sacamos el disco, sin desmontarlo, y una vez extraido el disco desmontamos la disquetera veremos que si montamos el disco el archivo no fue guardado y, posiblemente, nos de un aviso de `check forced`. Esto es debido a que se produjo una inconsistencia en el sisema de ficheros, no había actualizado toda la información del disco cuando sacamos el disco de la disquetera. Si hubieramos realizado un `umount` antes de sacar el disco hubiera realizado todas las operaciones de escritura pendientes en el disco antes de desmontar la unidad, con lo cual no se tendría ninguna inconsistencia.

Si reseteamos el ordenador sin apagarlo antes mediante las ordenes:

```
shutdown, halt o reboot
```

tendremos el mismo problema, todos los datos que se encuentren en el buffer y no se hayan escrito a disco se perderan y la siguiente vez que arranquemos el ordenador tendremos un `check forced`, con lo cual se chequeará el sistema antes de arrancar y tardará más en arrancar.

[5] Apartado 6.9 en la página 109.

6.8.2. El comando `fsck`

Al montar un sistema de ficheros nos podemos encontrar con un mensaje de aviso de que el sistema no fue limpiamente desmontado la última vez. En estos casos suele haber inconsistencias en el sistema de ficheros, por ejemplo que se quitó un disco de la disquetera sin haber desmontado antes la unidad, se fue la luz con el ordenador encendido o se apagó el ordenador directamente. En estos casos para chequear el sistema de ficheros se utiliza el comando `fsck`:

```
[root@localhost root]# fsck -t ext2 /dev/sda1
```

Con esto se chequearía el sistema de ficheros tipo ext2 que hay en la primera partición del primer dispositivo SCSI.

El único usuario que debería poder chequear los sistemas de ficheros debería ser el `root` y para poder chequear un sistema de ficheros tiene que haber sido previamente desmontado.

Al igual que pasaba con el comando `mkfs` suele haber otros comandos:

fsck.ext2 equivalente a `fsck -t ext2`.

fsck.ext3 equivalente a `fsck -t ext3`.

fsck.vfat equivalente a `fsck -t vfat`.

...

Sería una buena idea que consultaras las página del manual acerca de este comando.

6.8.3. El comando `badblocks`

El uso de este comando está restringido al superusuario o `root` y sirve para encontrar bloques defectuosos en un sistema de ficheros. La forma más fácil de uso es:

```
[root@localhost root]# badblocks /dev/sda1
```

esto buscará bloques defectuosos en el dispositivo `/dev/sda1`, primera partición del primer dispositivo SCSI. Es posible especificar un intervalo de bloques en el dispositivo en el que queremos buscar bloques defectuosos, además podemos escribir la lista de bloques defectuosos en un fichero con el flag "`-o`", así mismo podemos utilizar ese fichero para saltar la comprobarción de esos bloques, que ya sabemos que estan defectuosos, utilizando el flag "`-i`".

Para más información consulta la página del manual.

6.9. Sistemas de ficheros de Journaling

Ultimamente se habla mucho de los sistemas de ficheros de Journaling. Pero ¿que son?. Ya hemos visto los problemas que existen al remover un dispositivo físico sin haberlo desmontado previamente, o que se vaya la luz cuando el ordenador esta encendido. Todo esto puede provocar la perdida de datos.

Los sistemas de Journaling, también conocidos como transaccionales, son sistemas de ficheros diseñados para ser tolerantes a fallos. Es decir, que si se va la luz cuando tenemos el ordenador encendido al volver a arrancarlo no tengamos que esperar tanto tiempo debido a la comprobación de errores[6] y que no perdamos datos.

Todos los ficheros que tenemos en el disco tienen asociados unos datos:

- El contenido del fichero, lo que nosotros escribimos.
- Los datos referentes al fichero, tamaño, fecha, nombre, en que lugar del disco se encuentra.

En los sistemas de ficheros tradicionales cuando guardamos un fichero se modifican directamente tanto el contenido del fichero como los datos referentes al fichero. Si el equipo falla durante ese proceso, un fallo de luz, un fallo del hardware, ... tendremos una inconsistencia en el sistema de ficheros. Podriamos perder los datos del fichero, entonces tendríamos que recurrir a una de las multiples copias de seguridad que, normalmente, no se hacen.

Los sistemas de Journaling para prevenir esto disponen de dos zonas:

1. Una de datos, donde se almacenarán tanto el contenido de los ficheros como los datos referentes a los ficheros.
2. Otra zona de "*Log*" donde el sistema escribe los cambios que va a realizar.

Cuando el sistema actualiza todos los datos de un fichero, borra de la zona de "*Log*" la información correspondiente a esos cambios. Si el sistema cae antes de actualizar el disco, al arrancar lee los datos de la zona de "*Log*" y actualiza el sistema de ficheros sin perdidas de datos.

[6] El famoso `check forced`.

Basicamente así es como funciona un sistema de ficheros de este tipo. Algunos de los sistemas de ficheros de este tipo que existen son:

ext3 que es la versión mejorada, con Journaling, de ext2.

ReiserFS creado por Hans Reiser.

XFS sistema de Journaling de Silicon Graphics para **GNU/Linux**.

JFS sistema de Journaling de IBM para **GNU/Linux**.

6.10. Sistemas criptográficos de ficheros

Ademas de los sistemas de ficheros "*clásicos*", por llamarlos de alguna forma, existen otro "*tipo*" de sistemas de ficheros.

Si llevamos un disco duro de un ordenador a otro, lo conectamos y lo montamos podremos acceder a **TODA** su información. En caso de robo no podremos proteger nuestros datos con los sistemas de ficheros tradicionales.

Los sistemas criptográficos de ficheros solucionan este "*problema*" ya que los datos estan encriptados en el disco y no son accesibles de forma directa. Bueno, en realidad son accesibles, lo que no son es comprensibles.

6.10.1. Cryptographic File System (C.F.S.)

Esta utilidad permite la creacion de ficheros y directorios encriptados dentro del sistema de ficheros que estemos utilizando.

Para poder acceder a estos ficheros se necesitará una contraseña que habrá sido establecida cuando se creó el directorio.

Este software está basado en D.E.S. y está sometido a las leyes de exportación de los Estados Unidos.

6.10.2. Trasparent Cryptographic File System (T.C.F.S.)

Desarrollado por la Universidad de Salerno específicamente para Linux. Su propósito es proporcionar seguridad en sistemas de ficheros NFS. La desencriptación se realiza en el cliente.

Este software funciona a nivel de nucleo, no a nivel de usuario, con lo cual se gana en seguridad.

Esta basado en D.E.S. y para funcionar las máquinas clientes necesitan una versión compilada del nucleo para soporte de T.C.F.S.

Se puede encontrar más información sobre este sistema de ficheros en:

`http://www.tcfs.it`

6.10.3. Self-certifying File System (F.S.)

Se puede encontrar más información sobre este sistema de ficheros en:

`http://www.fs.net`

6.11. Cuotas de usuario

En los sistemas **UNIX** se pueden establecer cuotas de usuario para evitar que los usuarios monopolicen el disco duro o el sistema de ficheros.

La cantidad de espacio disponible es un recurso finito y su agotamiento puede conducir a un malfuncionamiento del sistema, por esta razón es importante una buena gestión de su uso y evitar que un usuario o varios lo agoten. En sistemas multiusuario es muy importante el llevar una buena gestión de los sistemas de ficheros, ya que si dejamos via libre a los usuarios terminarán por agotar el espacio disponible.

Una forma de realizar esto es mediante el establecimiento de las cuotas de disco, lo cual nos permitirá establecer la cantidad de espacio que los usuarios o grupos de usuarios pueden utilizar en cada sistema de ficheros que tenga montado la máquina.

Lo que vamos a ver a continuación es como funciona el sistema de cuotas sobre sistemas de ficheros `ext2`. Sobre sistemas de ficheros con Journaling mira las páginas `man`, especialmente con XFS.

6.11.1. ¿En que sistemas de ficheros se pueden establecer cuotas?

Podemos establecer cuotas en todos los sistemas de ficheros que esten en `/etc/fstab`. Hay que tener en cuenta que hay sitios en el sistema de ficheros en los que el establecimiento de cuotas de disco no tiene sentido.

Es muy frecuente el que un sistema **GNU/Linux** o **UNIX** esté distribuido en varias particiones o discos y estos sean montados al arrancar. Esto facilita la administración y también es bueno desde el punto de vista de la seguridad.

Es posible que el directorio `/usr/` esté es un sistema de ficheros diferente. No tiene sentido el establecimiento de cuotas en este sistema de ficheros ya que los usuarios normales no deben tener permisos de escritura en este directorio. Sin embargo es posible que el directorio `/tmp/` también esté montado en otro sistema de ficheros, y dado que en este directorio si que tienen permiso de escritura todos los usuarios sería conveniente el establecimiento de cuotas de disco para prevenir que uno o varios usuarios monopolicen el sistema de ficheros en el que esta montado. Lo mismo ocurre con el directorio `/home/`.

6.11.2. ¿A que usuarios puedo limitarles el uso de espacio en disco?

Podemos establecer cuotas a todos los usuarios que aparezcan en el fichero `/etc/passwd`, luego podemos establecer cuotas de disco para todos los usuarios del sistema.

6.11.3. ¿A que grupo de usuarios puedo limitarles el uso de espacio en disco?

Podemos establecer coutas a todos los grupos de usuarios que aparezcan en el fichero `/etc/group`, luego podemos establecer coutas de disco para todos los grupos de usuarios del sistema.

6.11.4. Funcionamiento de las cuotas de disco

En cada sistema de ficheros en el que establezcamos cuotas de usuario podemos establecer varios parámetros. Serán estos parámetros los que dictaminarán el comportamiento del sistema ante los abusos en el consumo del disco:

Límite "hard" para usuarios Establece la máxima cantidad de espacio permitido por usuario. Una vez superado el usuario no podrá escribir en el sistema de ficheros.

Límite "hard" para grupos Establece la máxima cantidad de espacio permitido para el grupo. Una vez superado ningún usuario del grupo podrá escribir en el sistema de ficheros. No importa que un usuario no haya superado su límite, si el grupo supera su cuota de espacio nadie en el grupo podrá escribir en el sistema de ficheros.

Límite "soft" para usuarios Este límite se utiliza para indicar a los usuarios que se estan aproximando al límite "hard". Podría decirse que es un mecanismo que sirve para alertar al usuario de que está próximo a agotar su espacio en el sistema de ficheros.

Cuando se supera este límite cada vez que el usuario realiza una operación de escritura se le muestra en la terminal un mensaje de advertencia.

Límite "soft" para grupos Funciona igual que el límite "soft" para usuarios, sólo que afecta a todo el grupo.

Periodo de gracia Cuando se sobrepasa el límite "soft" se entra en el periodo de gracia. Una vez que este periodo pasa no se permite al usuario o grupo de usuarios escribir en el sistema de ficheros hasta que la cantidad de espacio que tienen ocupada sea inferior al límite "soft".

El periodo de gracia puede especificarse en meses, semanas, días, horas, minutos o segundos.

6.11.5. Pasos previos a la activación de las cuotas

Para el uso de cuotas de disco es necesario que el nucleo esté compilado con soporte para ello. Como es de suponer sólo el `root` puede establecer las cuotas de disco.

Supongamos que queremos establecer cuotas de disco en `/dev/sda5`, el cual está montado en `/home`. Tendremos que hacer lo siguiente:

1. En cada sistema de ficheros en el que queramos utilizar cuotas tenemos que añadir en `/etc/fstab` en las opciones de montaje:

   usrquota si queremos utilizar cuotas para los usuarios.

   grpquota si queremos utilizar cuotas para los grupos.

2. Crear en lo alto del sistema de ficheros los ficheros:

   quota.user para establecer las cuotas de los usuarios.

   ```
   [root@localhost root]# touch /home/quota.user
   ```

   quota.group para establecer las cuotas de los grupos

   ```
   [root@localhost root]# touch /home/quota.group
   ```

   A continuación tendremos que establecer los permisos de forma correcta para estos ficheros:

   ```
   [root@localhost root]# chmod 400 /home/quota.*
   ```

   Estos ficheros van a contener datos binarios y no de texto.

3. A continuación tenemos que inicializar las bases de datos que van a almacenar la información referente a las cuotas en los ficheros que hemos creado anteriormente:

   ```
   [root@localhost root]# quotacheck -avug
   quotacheck: Scanning /dev/sda5 [/home] done
   quotacheck: Checked 79 directories and 657 files
   [root@localhost root]#
   ```

Los flags que hemos utilizado sirven para:

a Realiza la comprobación para todos los sistemas de ficheros con cuotas.

v Modo `verbose`, indica todo lo que va haciendo.

u Realiza la comprobación para los usuarios.

g Realiza la comprobación para los grupos.

4. Activamos el sistema de cuotas:

```
[root@localhost root]# quotaon -a
[root@localhost root]#
```

6.11.6. Estableciendo las cuotas

Una vez que tenemos funcionando el sistema de cuota tenemos que establecer las cuotas para los usuarios y grupos.

Las cuotas se establecen por bloques e inodos. Si no se ha jugeteado con las opciones al crear el sistema de ficheros cada bloque equivaldrá, normalmente, a 1 Kb (1.024 bytes).

Para establecer las cuotas se utiliza `edquota`, este comando accede a los ficheros `quota.user` y `quota.group` crea un fichero temporal en `/tmp` y lo edita por defecto con `vi` a menos que en las variables de entorno `EDITOR` o `VISUAL` tengamos especificado otro.

Algunos de los flags que podemos utilizar son:

u Se utiliza para modificar las cuotas de disco de los usuarios. Si se especifica el flag `g` esta opción es ignorada.

g Se utiliza para modificar las cuotas de disco de los grupos.

Para cambiar las cuotas del usuario `jose` debemos ejecutar desde una consola y como `root`:

```
[root@localhost root]# edquota -u jose
```

A continuación se accederá a los ficheros de cuotas y se creará en `/tmp` un fichero con los datos que será editado y aparecerá algo como esto:

```
Disk quotas for user jose (uid 1000):
Filesystem     blocks   soft  hard  inodes  soft  hard
/dev/sda5       10848  15000 25000     732  2000  3500
```

A continuación modificamos a nuestro gustos los campos "*soft*" y "*hard*" y guardamos el fichero, al salir del editor `edquota` almacenará los nuevos datos en los ficheros de cuotas. La información que aparece en "*blocks*" e "*inodes*" hace referencia a los bloques e inodos que tiene el usuario usados en ese momento.

También podemos establecer las cuotas en línea de comando utilizando el comando `setquota`[7].

6.11.7. Estableciendo el periodo de gracia

El periodo de gracia se establece con `edquota -t`. El funcionamiento es igual que para establecer las cuotas de usuarios o grupos. Se crea un fichero en `/tmp` que es editado. Para establecerlo:

```
[root@localhost root]# edquota -t
```

y aparecerán todos los sistemas de ficheros con cuotas. Algo como esto:

```
Grace period before enforcing soft limits for users:
Time units may be: days, hours, minutes, or seconds
Filesystem       Block grace period   Inode grace period
/dev/sda5               7days                7days
```

Como vemos el periodo de gracia se puede establecer para bloques e inodos, lo modificamos a nuestro gusto y al salir guardando los cambios `edquota` actualiza las bases de datos del sistema de cuotas con los nuevos datos.

[7] Prueba man setquota.

6.11.8. Iniciando y parando el sistema de cuotas

Una vez que hemos indicado los sistemas de ficheros en los que estableceremos las cuotas de disco, hemos establecido las cuotas y el periodo de gracia tenemos que inicializar el sistema de cuotas.

Para iniciar el sistema de cuotas se utiliza el comando `quotaon`:

```
[root@localhost root]# quotaon -av
/dev/sda5 [/home]: group quotas turned on
/dev/sda5 [/home]: user quotas turned on
[root@localhost root]#
```

Los flags más habituales son:

a Inicializa todas las cuotas de los sistemas de ficheros que las tienen activadas.

v Modo `verbose`, indica todo lo que va haciendo.

u Unicamente inicializa las cuotas para los usuarios.

g Unicamente inicializa las cuotas para los grupos.

También es posible inicializar las cuotas de un sistema de ficheros en concreto. Para ello es necesario que ya esté montado y que esté especificado en `/etc/fstab`:

```
[root@localhost root]# quotaon -v /home
/dev/sda5 [/home]: group quotas turned on
/dev/sda5 [/home]: user quotas turned on
[root@localhost root]#
```

También es posible inicializar sólo las cuotas de usuario o las de grupo:

```
[root@localhost root]# quotaon -vu /home
/dev/sda5 [/home]: user quotas turned on
[root@localhost root]# quotaon -vg /home
/dev/sda5 [/home]: group quotas turned on
[root@localhost root]#
```

Podemos para el sistema de quotas utilizando el comando `quotaoff`. Los flags de este comando y su funcionamiento son similares a `quotaon`. Si tienes alguna duda `man quotaoff` ;-).

6.11.9. Chequeando el sistema de cuotas

Para chequear los sistemas con cuotas se utiliza el comando `quotacheck`. Este comando se utiliza para chequear y actualizar el uso de espacio en los sistemas de ficheros y para reparar los ficheros de cuotas `quota.user` y `quota.group`.

Por defecto sólo se comprueban las cuotas de usuario, si se desea chequear las cuotas de grupo hay que especificar el flag "**g**". Cuando se comprueben las cuotas de disco es preferible hacerlo con el sistema de cuotas parado.

Es aconsejable que cuando se arranca el sistema se comprueben los sistemas de ficheros con cuotas antes de inicializar el sistema de cuotas, para ello es necesario que estos sistemas de ficheros hayan sido montados previamente.

Los flags más habituales son:

a Comprueba todos los sistemas de ficheros montados con cuotas establecidas.

g Comprueba las cuotas de grupo. No se comprueban a menos que se especifique este flag.

i Trabaja en modo interactivo. Pregunta antes de realizar una acción.

u Comprueba las cuotas de usuario. Acción por defecto.

v Modo verbose, es aconsejable ejecutar `quotacheck` siempre con este flag para ver que es lo que sucede.

6.11.10. Obteniendo informes sobre las cuotas

También es posible obtener informes sobre el estado de las cuotas de disco mediante el comando `repquota`. Los flags más significativos son:

a Informe sobre todos los sistemas con coutas especificados en `/etc/fstab`.

g Informe sobre las cuotas de grupos.

u Informe sobre las cuotas de usuarios.

```
[root@localhost root]# repquota -a
*** Report for user quotas on device /dev/sda5
Block grace time: 7days; Inode grace time: 7days
                Block limits               File limits
User     used    soft    hard  grace  used  soft  hard  grace
----------------------------------------------------------------
root --    20       0       0              4     0     0
jose -- 10848   15000   25000          732  2000  3500
[root@localhost root]#
```

También es posible hacerlo sobre un determinado sistema de ficheros, para ello bastará con indicarle el punto de montaje, `/home`, o el dispositivo físico, `/dev/sda5`.

Capítulo 7

El sistema de ficheros de red NFS

Este sistema de archivos permite compartir recursos (discos duros, CD-Rom's, ...) entre varias máquinas utilizando una red. Con este sistema una máquina puede montar un disco duro que está en otra máquina y acceder a esa información.

Muchas veces estamos trabajando en una máquina y el directorio `/opt` está en otra máquina y nosotros no somos conscientes. Ello es posible ya que al arrancar nuestra máquina hemos montado en un directorio local `/opt` un disco duro o partición de otro equipo via red. Es totalmente transparente al usuario y permite un mejor uso y planificación de recursos, ya que el ordenador en el que realmente reside nuestra información puede encargarse de hacer copias de seguridad o tener montado un sistema RAID y toda esa carga la realizaría otra máquina, dejando a nuestra máquina todos sus recursos para atender a sus usuarios.

7.1. Remote Procedure Call, RPC

El **RPC**[1] lo que hace es que un ordenador, servidor, procesa ordenes de otro ordenador, cliente, y una vez procesadas el servidor le envia al cliente los resultados. Normalmente estos servicios se realizan a traves del puerto 111.

Si un cliente necesita hacer una llamada **RPC** a un servidor se realizaría asi:

1. El cliente realiza una petición **RPC**.
2. El cliente se pone en contacto con el servidor y con el servicio `portmap`.
3. El servicio `portmap` del servidor le comunica al cliente a que puerto del servidor debe dirigirse.
4. El cliente se pone en contacto con el puerto especificado, donde el demonio correspondiente esta esperando conexiones.
5. El servidor realiza la petición, si procede, y devuelve el resultado al cliente.

Este servicio es usado por otros servicios como por ejemplo:

- El servicio `NFS` o sistema de ficheros de red.
- El servicio `NIS` o sistema de información de la red.

El sistema **RPC** es independiente de plataforma ya que utiliza un lenguaje llamado **XDR** o e**X**ternal **D**ata **R**epresentation.

[1] Procedimientos de llamada remota.

7.2. ¿Qué es NFS?

NFS es una tecnología desarrollada por Sun Microsystems en el año 1.984 para la compartición de datos entre varios ordenadores, independientemente del sistema operativo que esten corriendo.

Este servicio funciona a través del protocolo TCP/IP en una arquitectura cliente/servidor utilizando RPC.

GNU/Linux implementa este servicio en el nucleo, gracias a lo cual es transparente para los usuarios y/o programas.

Aunque es posible utilizar NFS entre varios ordenadores independientemente de donde se encuentren, mientras estén comunicados por TCP/IP, muchas veces no es posible utilizar este servicio. Es necesario tener una buena conexión. Normalmente este servicio es utilizado dentro de redes locales.

7.3. ¿Como compartir información con NFS?

Para poder compartir información con NFS es necesario que:

- El nucleo de nuestra máquina este compilado con soporte para NFS.
- La máquina que va a compartir dicha información exporte la información a compartir y nos de permiso para acceder a ella.

  Exportar la información no es más que decirle al sistema que información se va a compartir, a quien se le permite acceder a esa información y como puede acceder.

7.3.1. El fichero `/etc/exports`

Este fichero contiene los sistemas de ficheros, directorios, ... que son accesibles por otras máquinas, además de contener que máquinas pueden acceder a ellos y como pueden acceder. Por ejemplo:

```
#
# /etc/exports
#

/mnt/cdrom        *(ro)
/mnt/raid         *.localdomain(rw)
/mnt/vfat         *.localdomain(ro,no_root_squash)
```

Veamos como se interpreta el fichero `/etc/exports`:

/mnt/cdrom es un directorio que puede ser accedido por cualquier cliente en modo sólo lectura.

/mnt/raid es un directorio que puede ser accedido por cualquier cliente dentro de la red `localdomain` en modo lectura/escritura.

/mnt/vfat es un directorio que puede ser accedido por cualquier cliente dentro de la red `localdomain` en modo sólo lectura y permite al `root` no perder sus privilegios[2].

Después del nombre de cada máquina se especifican las opciones de acceso para los usuarios:

ro montará el sistema de archivos en modo sólo lectura.

rw montará el sistema de archivos en modo lectura y escritura.

root_squash niega el acceso a los ficheros del superusuario aunque dichas peticiones las realice un usuario con UID 0 (superusuario) o UID -2 (nobody). Esta opción es utilizada por defecto.

no_root_squash contrario a la opción anterior.

all_squash cualquier petición es procesada como un usuario anónimo. Todos los usuarios tendrán los mismos privilegios, es util en sistemas de ficheros para servidores de *ftp*.

Para ver con mayor detalle consulta la página del manual del fichero de configuración `exports` en la sección 5.

[2]Ya veremos esto más adelante.

7.3.2. ¿Como ver que directorios tengo exportados?

Para poder ver los directorios que hay exportados en nuestra máquina podemos utilizar el comando `exportfs`, si somos el superusuario:

```
[root@localhost root]# exportfs
/mnt/cdrom      *
/mnt/raid       *.localdomain
/mnt/vfat       *.localdomain
[root@localhost root]#
```

7.3.3. ¿Como montar un directorio via NFS?

Para montar un directorio remoto via NFS lo haremos utilizando el comando `mount`:

```
[root@localhost root]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda2             1.3G  1.2G   63M  95% /
none                   30M     0   30M   0% /dev/shm
/dev/md0              1.9G   33M  1.7G   2% /mnt/raid
/dev/hda6             610M  459M  151M  76% /mnt/vfat
[root@localhost root]# mount -t nfs 192.168.0.1:/mnt/raid /mnt/nfs
[root@localhost root]# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda2             1.3G  1.2G   63M  95% /
none                   30M     0   30M   0% /dev/shm
/dev/md0              1.9G   33M  1.7G   2% /mnt/raid
/dev/hda6             610M  459M  151M  76% /mnt/vfat
192.168.0.1:/mnt/raid 1.9G   33M  1.7G   2% /mnt/nfs
[root@localhost root]#
```

El tipo de archivos que hemos de utilizar es "`nfs`", independientemente de si es un `ext2`, `ext3`, `reiserfs`, ... y como dispositivo hemos de especificar la IP de la máquina que lo tiene exportado seguido de dos puntos y el punto de montaje en la máquina que lo tiene montado y luego especificar donde queremos montarlo.

Se puede ver como en la misma máquina tenemos montados DOS VECES el mismo sistema de archivos. **GNU/Linux** nos permite montar un sistema de archivos por NFS en nuestra propia máquina sin estar conectados a ninguna red. En caso de no tener ninguna tarjeta de red podríamos hacer lo mismo utilizando 127.0.0.1 en lugar de 192.168.0.1.

Los sistemas de ficheros en red se montan como cualquier otro sistema de ficheros, razón por la cual podemos incluir información del sistema en el fichero `/etc/fstab` y de esta forma podriamos montarlo unicamente especificando el punto de montaje.

Para ver todas las opciones que le podemos especificar a `mount` para los sistemas de ficheros en red deberemos consultar la página del manual de `nfs` en la sección 5.

7.3.4. El comando `exportfs`

Ya hemos visto que el comando `exportfs` nos muestra los directorios exportados en nuestra máquina. Pero además tiene otras utilidades.

Este comando se utiliza para mantener la tabla de los sistemas exportados via NFS. Esta lista se almancena, normalmente, en:

```
/var/lib/nfs/xtab
```

Cuando una máquina cliente realiza una petición de montaje de un sistema de ficheros remoto y es concedido se crea una entrada en dicho fichero en la máquina servidora.

Si arrancamos a mano el servicio NFS será conveniente que ejecutemos `exportfs -a` para que actualice las tablas de exportación del nucleo.

Cada vez que modifiquemos el fichero `/etc/exports` será necesario ejecutar `exportfs -r` para que los nuevos cambios tengan efecto.

7.4. Como funciona NFS

Supongamos que tenemos configurado el servicio NFS. Para poder acceder a un disco o una partición via NFS los pasos a seguir serán:

1. El administrador de la máquina cliente tiene que motar el directorio correspondiente:

```
[root@localhost root]# mount -t nfs bender:/mnt/my_shiny_ass /mnt/bender
[root@localhost root]#
```

2. `mount` conectará con el demonio `mountd` en la máquina `bender`[3] mediante RPC.

3. La máquina `bender` comprobará si el cliente esta autorizado a montar el directorio solicitado y en caso afirmativo le devolverá un descriptor que será utilizado en todas las peticiones de ficheros que se realicen sobre el directorio.

4. A continuación ya se puede utilizar el directorio montado remotamente. Cuando se haga un petición a dicho directorio se hará mediante una llamada RPC al demonio `nfsd` en la máquina remota.

7.4.1. Demonios necesarios para el funcionamiento de NFS

Para el funcionamiento de NFS serán necesarios unos cuantos demonios:

rpc.portmap es el demonio mapeador de puertos RPC. El encargado de decir a cada petición RPC a donde debe dirigirse.

rpc.mountd es el encargado de conceder/negar las peticiones de montaje remotas.

rpc.nfsd es el encargado de realizar las peticiones de ficheros, via NFS.

Estos servicios no suelen arrancarse bajo `inetd`, se arrancan en los niveles de ejecución como demonios independientes.

[3] En el dominio local.

7.4.2. ¿Como funcionan los permisos en NFS?

Ya hemos visto que en **GNU/Linux** se realizan los controles de acceso mediante unos permisos que afectan al propietario del fichero, grupo del propietario y resto de usuarios del sistema.

En los sistemas NFS tenemos que los ficheros se tendrán que someter al control en:

- El servidor ya que pertenecen a él.
- En el cliente, ya que estarán montados en él.

Ya hemos visto que cuando pedimos un fichero del directorio montado remotamente se hace mediante RPC al demonio `nfsd` en la máquina remota. Esto implica que se mandarán a la máquina remota el descriptor del directorio montado remotamente, el fichero que se desea y los UID y GID del usuario que lo solicita.

La forma en la que se le concede acceso es:

- Según lo establecido en el fichero `/etc/exports`.
- Y luego con los permisos que tengan en la máquina remota, servidor de ficheros, el usuario UID y el grupo GID.

Para que no haya problemas los usuarios de las dos máquinas deberían ser los mismos, con identicos datos dentro del sistema, el mismo UID y GID en las dos máquinas ya que sus controles de acceso se harán por su UID y GID.

Una forma de solucionar esto sería con **NIS/NYS** que permite tener centralizada en una sóla máquina toda la información referente a las cuentas de los usuarios.

7.4.3. El papel del usuario `nobody`

Muchas veces el superusuario de la máquina cliente no es el mismo que el de la máquina servidora de un sistema de ficheros en red. ¿Qué pasaría si el superusuario de la máquina cliente quisiera borrar todos los ficheros de un sistema exportardo, via NFS, por otra máquina?.

Como los controles de acceso se realizan por UID y GID podría hacerlo siempre y cuando el sistema de ficheros hubíera sido exportado como sistema de ficheros de lectura y escritura.

Para evitar esto en las peticiones del superusuario se hace corresponder al usuario `root`, UID 0, con el usuario `nobody`, UID 65534=-2. Normalmente el usuario `nobody` no posee ningún fichero, entonces sus accesos se harán de acuerdo a los permisos especificados al resto de usuarios.

Aunque esto protege del borrado de todos los ficheros no protege el borrado de los ficheros pertenecientes a los usuarios de la máquina cliente, ya que el superusuario puede acceder al fichero de claves[4] y asumir la identidad de dichos usuarios y con ello tendrá pleno acceso a sus ficheros.

Si en el fichero `/etc/exports` de la máquina servidora especificamos en un sistema de ficheros la opción "*no_root_squash*" permitiremos al superusuario de las máquinas clientes el pleno acceso a los ficheros del sistema montado via NFS (mucho cuidadin con esto!!!).

[4]Puede eliminar las contraseñas o cambiarlas.

Capítulo 8

Proceso de arranque en GNU/Linux

8.1. Los demonios en GNU/Linux

GNU/Linux desde sus inicios ha estado orientado a redes y como servidor. La forma que tiene **GNU/Linux** de ofrecer servicios es mediante los "*demonios*" o "*daemons*".

Los "*demonios*" son unos procesos que tienen asignada una determinada tarea, por ejemplo ofrecer páginas web. Los nombres de los "*demonios*" terminan en "**d**", normalmente, y son arrancados por el superusuario.

Lo normal es que se arranquen los "*demonios*" cuando se inicia el sistema.

8.1.1. ¿Como funciona un demonio?

Una vez arrancado un "*demonio*" no permanece activo todo el rato. Los "*demonios*" permanecen dormidos, inactivos, y cada cierto tiempo despiertan y comprueban si tienen pendiente alguna petición. Si la tienen la atienden y cuando han atendido todas sus peticiones vuelven a dormir.

Cada "*demonio*" tiene especificado un tiempo después del cual despertará y comprobará sus trabajos pendientes, por ejemplo un segundo, 500 milisegundos, ...

En realidad, no todos lo "*demonios*" funcionan de esta manera, ya que existen algunos demonios que se ejecutan bajo el control de un "*demonio*" especial llamado "`inetd`", el cual veremos más adelante.

8.1.2. Algunos demonios

Veamos a continuación algunos de los "*demonios*" de **GNU/Linux**:

atd es el "*demonio*" encargado de realizar los trabajos lanzados con `at`.

crond es el "*demonio*" encargado de realizar los trabajos lanzados con `cron`.

lpd es el "*demonio*" encargado de realizar los trabajos de impresion.

httpd es el "*demonio*" encargado de ofrecer páginas web.

inetd es el "*demonio*" también conocido como el "*super servidor*" o "*super demonio*", lo explicaremos con detalle más adelante.

sendmail demonio encargado del correo electrónico.

snmpd es el "*demonio*" encargado de contestar a las peticiones por `snmp`, el cual es un protocolo para redes.

syslogd demonio encargado de los logs del sistema.

8.2. Como arranca GNU/Linux

Lo primero que quiero decir es que el proceso de arranque que voy a describir es el que utiliza Red Hat Linux, que es la distribución que tengo instalada en este momento. En otras distribuciones puede variar la forma de arrancar, no mucho, lo que si puede cambiar es la localizacion de los *scripts* de arranque.

Cuando encendemos el ordenador, la BIOS[1] se encarga de llevar a cabo los primeros pasos. Una vez realizados y siguiendo las secuencias de arranque preestablecidas en la BIOS se encarga de buscar un sistema operativo en sus unidades de disco.

El encargado de arrancar **GNU/Linux** es "*LILO*", **LI**nux **LO**ader. Cuando la BIOS encuentra a LILO le transfiere el control y es LILO el que se encarga de cargar el nucleo. Una vez cargado el nucleo el control se transfiere al nucleo y este crea el proceso "**init**" que se encarga de arrancar los *scripts* de arranque e iniciar el sistema.

Esto es, de forma resumida, la forma es la que arranca **GNU/Linux**.

[1] **B**asic **I**nput **O**utput **S**ystem.

8.3. El gestor de arranque LILO

LILO es un gestor de arranque, su misión es cargar el nucleo de **GNU/Linux**, existen otros gestores de arranque más avanzados como GRUB. Aquí trataremos LILO por dos razones:

1. Es el gestor clásico de **GNU/Linux**.
2. Es el gestor de arranque con el que me he pegado. ;-)

8.3.1. El fichero /etc/lilo.conf

En este fichero esta la configuración de LILO, un ejemplo de este fichero puede ser:

```
#
# /etc/lilo.conf
#
prompt
timeout=50
default=linux
restricted
password=pepito
boot=/dev/hda2
map=/boot/map
install=/boot/boot.b
message=/boot/message
linear

image=/boot/vmlinuz-2.4.7-10
        label=linux
        initrd=/boot/initrd-2.4.7-10.img
        read-only
        root=/dev/hda2
```

Vamos a explicar los parámetros que aqui aparecen:

prompt indica que debe aparecer el prompt de LILO al arrancar, permite pasarle parámetros la nucleo en el arranque.

timeout segundos que esperará antes de arrancar el nucleo por defecto.

default nucleo por defecto.

restricted si se intenta arrancar **GNU/Linux** pasandole parámetros al nucleo nos pedirá la contraseña que aparece en el campo `password`. Como medida de seguridad.

password password necesaria si se arranca **GNU/Linux** pasandole parámetros al nucleo.

boot partición de arranque.

map indica el fichero "*map*" a utilizar, por defecto utiliza `/boot/map`.

install especifica el fichero que se utiliza como sector de arranque, por defecto es `/boot/boot.b`.

message especifica el fichero en el que se encuentra el mensaje que aparece antes del prompt de lilo.

linear se utiliza para utilizar direcciones de disco que no dependan de la geometría del disco.

imagen nucleo para arrancar.

- **label** etiqueta del nucleo. LILO permite tener varios nucleos y arrancar el que mejor se adapte a nuestras necesidades.
- **initrd** especifica un fichero con datos que contiene los datos del sistema de ficheros.
- **read-only** que arranque en modo sólo lectura.
- **root** la partición raíz del sistema.

LILO además permite arrancar otros pseudo-sistemas operativos como `MS-DOS`, `Windows 95` o `Windows 98`. Sistemas operativos como `Windows N.T.` o `Windows 2.000` no pueden ser arrancados mediante LILO.

Si en nuestro equipo tienen que convivir alguno los sistemas operativos que LILO no puede arrancar tenemos dos opciones:

1. Instalar GRUB, que si lo permite.

2. Utilizar el gestor de arranque de NT para arrancar **GNU/Linux**. Para ello LILO tiene que ser instalado en el primer sector de la partición en la que se encuentre **GNU/Linux**. A continuación copiamos el primer bloque de la partición de **GNU/Linux** a un fichero:

   ```
   [root@localhost root]# dd if=/dev/hda2 of=/mnt/floppy/linuxstart count=1
   1+0 registros leidos
   1+0 registros escritos
   [root@localhost root]#
   ```

   a continuación copiamos el fichero `linuxstart` en el directorio raíz de `Windows N.T.` o `Windows 2.000` y añadimos al fichero `boot.ini` lo siguiente:

   ```
   C:\linuxstart =''LiNUX''
   ```

   Al hacer esto la próxima vez que arranquemos `Windows N.T.` o `Windows 2.000` aparecerá en el gestor de arranque una opción para arrancar **GNU/Linux**, si la elegimos arrancaremos **GNU/Linux**.

Supongamos que en nuestro ordenador tenemos instalado **GNU/Linux** y `Windows 9x` y queremos que LILO sea capaz de arrancarlo. En ese caso LILO debe estar instalado en el MBR del disco, con lo cual en el ejemplo anterior tendremos que:

```
boot=/dev/hda
```

y además después de la información referente a **GNU/Linux** tendremos que añadir:

```
other=/dev/hda1
      label=guindows
```

Cuando modifiquemos el fichero `/etc/lilo.conf`, o cambiemos la imagen del nucleo, tendremos que ejecutar el comando `lilo` para que LILO reconozca los cambios realizados en su configuración[2], es NECESARIO ejecutar LILO cada vez que modifiquemos algo en su configuración:

```
[root@localhost root]# lilo
Added linux *
Added guindows
[root@localhost root]#
```

Con esto cuando arranquemos **GNU/Linux** y nos aparezca el prompt podremos elegir el sistema operativo a arrancar, si pulsamos la tecla TAB veremos las opciones. Tecleamos la opción que deseemos y LILO se encargará de arrancarlo. En caso de no decirle nada a LILO arrancará la imagen que este especificada por defecto después de los segundos especificados en `timeout`.

8.3.2. Seguridad en LILO

LILO permite el paso de opciones en el arranque, para eso en el fichero `/etc/lilo.conf` tiene que estar especificada la opción `prompt`, esto es útil si:

- Disponemos de varias imagenes para arrancar. Cuando compilamos el nucleo es MUY inteligente el no borrar el nucleo con el que estamos funcionando. Se añade el nuevo nucleo a LILO y se intenta arrancar con él. Si hemos hecho algo mal y la máquina se queda colgada, agradeceremos mucho el poder continuar arrancando con el nucleo viejo y volver a compilar, intentando hacerlo mejor que la última vez ;-), el nucleo.
- Si disponemos en nuestra máquina de otros sistemas operativos.
- Ante fallos al configurar hardware. Por ejemplo al configurar una PCMCIA en un portatil, en caso de no haberla configurado bien, o simplemente que se instale en la IRQ que no debe, al arrancar el ordenador se bloquea, entonces podremos arrancar mediante la opción "`single`" y arreglar el problema.

Pero también tiene sus inconvenientes, y es que utilizando la opción "`single`" cualquiera que reinicie la máquina podrá entrar en modo **monousuario**, lo que implica entrar en la máquina como superusuario sin conocer la clave del superusuario ...

[2]Si tenemos un arranque dual con el gestor de arranque de NT deberemos volver a generar un fichero con el sector de arranque utilizando `dd` y actualizarlo.

Para evitar esto tenemos dos opciones:

1. No especificar en el fichero `lilo.conf` la opción `prompt`, con lo cual no se podrán introducir opciones al nucleo de **GNU/Linux** en el arranque.

2. Lo anterior tiene la desventaja de que si tenemos varios sistemas operativos en la máquina o varias imagenes del nucleo unicamente podremos arrancar uno de ellos. Entonces para evitar esto podemos incluir en el fichero `/etc/lilo.conf` las dos lineas siguientes:

   ```
   restricted
   password=pepito
   ```

   con esto cada vez que se intente arrancar el nucleo de **GNU/Linux** pasandole algún argumento nos pedirá la contraseña que aparece en `password`, en este caso `pepito`.

   En este caso es de vital importancia que el fichero `/etc/lilo.conf` no tenga permisos de lectura excepto para el superusuario, o en todo caso para el grupo del superusuario[3].

   Aqui se podría especificar que utilizara la contraseña del superusuario, cotejandola con la del fichero de contraseñas, pero muchas veces dentro del sistema existen usuarios que se dedican a realizar tareas de administración y a los cuales se les permite reiniciar la máquina. Para ello es necesario que puedan hacerlo especificando parámetros al nucleo en el arranque, pero puede ser recomendable el que no tengan acceso a la cuenta del administrador y es por esto que se utiliza una contraseña diferente.

8.4. Los niveles de ejecución

Los niveles de ejecución sirven para arrancar la máquina de una determinada forma. Los procesos que se arrancarán cada vez que se inicie la máquina estarán especificados en el fichero `/etc/inittab`[4].

[3] Si a los miembros de dicho grupo les es permitido arrancar la máquina pasandole parámetros al nucleo.

[4] Apartado 8.5.1 en la página 144.

8.4.1. ¿Cuantos niveles de ejecución existen?

A los niveles de ejecución se les designa por un número y cada nivel de ejecución está pensado para una determinada acción. Estan estándarizados ya que existen una multitud de sistemas `UNIX`, incluidos en ellos las diferentes distribuciones de **GNU/Linux**:

Nivel de ejecución 0 se utiliza para parar el sistema.

Nivel de ejecución 1 se utiliza para arrancar el sistema en modo monousuario y sin servicios de red. Este nivel se utiliza basicamente para la instalación de software o reconfiguración del sistema, ya que no permite conectarse a la máquina. Unicamente podrá el superusuario y tendrá que hacerlo en la propia máquina, no remotamente.

Nivel de ejecución 2 modo multiusuario sin NFS. Ideal para trabajar sin red.

Nivel de ejecución 3 modo multiusuario con NFS. Igual que el nivel 2 pero con funcionalidades de red.

Nivel de ejecución 4 a disposición del administrador.

Nivel de ejecución 5 modo gráfico. En este nivel de ejecución se entra directamente en X-Window. El proceso `login` se realiza de modo gráfico.

Nivel de ejecución 6 se utiliza para reiniciar el sistema.

Los niveles de ejecución 0, 1 y 6 suelen ser los mismos en todas las distribuciones, el resto pueden cambiar de una distribución a otra.

Por ejemplo en Red Hat el nivel de ejecución 3 es el nivel de ejecución en el que entra la máquina por defecto (cuando no entramos en modo gráfico), mientras que en Debian se entra en el nivel de ejecuación 2.

8.4.2. ¿Qué servicios se cargan en cada nivel?

Para cada servicio que deber ser arrancado en los niveles de ejecución existe un *script* que lo inicia. Cuando hay que iniciar un determinado servicio se ejecuta ese *script*.

Todos los *scripts* relacionados con los niveles de ejecución se suelen guardar dentro de un directorio llamado `rc.d`. En Red Hat ese directorio se encuentra dentro de `/etc`. En SuSE Linux dichos *scripts* se encuentran en `/sbin/init.d`.

Suponiendo que la ruta hacía dicho directorio sea `/etc/rc.d` encontraremos los siguientes directorios:

init.d contiene los *scripts* de arranque de cada servicio.

rc0.d contiene enlaces a los servicios, al *script* encargado, que se arrancarán en el nivel de ejecución 0.

rc1.d contiene enlaces a los servicios, al *script* encargado, que se arrancarán en el nivel de ejecución 1.

rc2.d contiene enlaces a los servicios, al *script* encargado, que se arrancarán en el nivel de ejecución 2.

rc3.d contiene enlaces a los servicios, al *script* encargado, que se arrancarán en el nivel de ejecución 3.

rc4.d contiene enlaces a los servicios, al *script* encargado, que se arrancarán en el nivel de ejecución 4.

rc5.d contiene enlaces a los servicios, al *script* encargado, que se arrancarán en el nivel de ejecución 5.

rc6.d contiene enlaces a los servicios, al *script* encargado, que se arrancarán en el nivel de ejecución 6.

8.4.3. ¿Como se arrancan los servicios en cada nivel de ejecución?

Para comprender mejor como se arrancan los servicios en cada nivel de ejecución veamos el contenido de `/etc/rc.d/rc3.d`, donde se encuentran los enlaces correspondientes al nivel de ejecución 3:

```
[root@localhost root]# ls -l /etc/rc.d/rc3.d
total 0
lrwxrwxrwx 1 root  root  19 abr 14 19:14 K01kdcrotate -> ../init.d/kdcrotate
lrwxrwxrwx 1 root  root  15 abr 14 19:14 K03rhnsd -> ../init.d/rhnsd
lrwxrwxrwx 1 root  root  16 abr 14 19:14 K12mysqld -> ../init.d/mysqld
lrwxrwxrwx 1 root  root  15 abr 14 19:14 K15httpd -> ../init.d/httpd
lrwxrwxrwx 1 root  root  15 abr 14 19:14 K20rwhod -> ../init.d/rwhod
lrwxrwxrwx 1 root  root  15 abr 14 19:14 K50snmpd -> ../init.d/snmpd
lrwxrwxrwx 1 root  root  16 abr 14 19:14 K72autofs -> ../init.d/autofs
lrwxrwxrwx 1 root  root  16 abr 14 19:14 K73ypbind -> ../init.d/ypbind
lrwxrwxrwx 1 root  root  14 abr 14 19:14 K74nscd -> ../init.d/nscd
lrwxrwxrwx 1 root  root  14 abr 14 19:14 K74ntpd -> ../init.d/ntpd
lrwxrwxrwx 1 root  root  18 ene 13 19:42 K92iptables -> ../init.d/iptables
lrwxrwxrwx 1 root  root  15 abr 14 19:14 K95kudzu -> ../init.d/kudzu
lrwxrwxrwx 1 root  root  17 abr 14 19:14 S10network -> ../init.d/network
lrwxrwxrwx 1 root  root  16 abr 14 19:14 S12syslog -> ../init.d/syslog
lrwxrwxrwx 1 root  root  17 abr 14 19:14 S13portmap -> ../init.d/portmap
lrwxrwxrwx 1 root  root  17 abr 14 19:14 S14nfslock -> ../init.d/nfslock
lrwxrwxrwx 1 root  root  18 abr 14 19:14 S17keytable -> ../init.d/keytable
lrwxrwxrwx 1 root  root  16 abr 14 19:14 S20random -> ../init.d/random
lrwxrwxrwx 1 root  root  15 abr 14 19:14 S25netfs -> ../init.d/netfs
lrwxrwxrwx 1 root  root  20 abr 14 19:14 S56rawdevices -> ../init.d/rawdevices
lrwxrwxrwx 1 root  root  16 abr 14 19:14 S56xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root  root  13 abr 14 19:14 S60lpd -> ../init.d/lpd
lrwxrwxrwx 1 root  root  13 abr 14 19:14 S60nfs -> ../init.d/nfs
lrwxrwxrwx 1 root  root  18 abr 14 19:14 S80sendmail -> ../init.d/sendmail
lrwxrwxrwx 1 root  root  13 abr 14 19:14 S85gpm -> ../init.d/gpm
lrwxrwxrwx 1 root  root  15 abr 14 19:14 S90crond -> ../init.d/crond
lrwxrwxrwx 1 root  root  13 abr 14 19:14 S90xfs -> ../init.d/xfs
lrwxrwxrwx 1 root  root  17 abr 14 19:14 S95anacron -> ../init.d/anacron
lrwxrwxrwx 1 root  root  13 abr 14 19:14 S95atd -> ../init.d/atd
lrwxrwxrwx 1 root  root  11 ene 13 19:40 S99local -> ../rc.local
[root@localhost root]#
```

Podemos ver que los nombres de los enlaces son de una de las dos siguientes formas:

SXXnombre donde "`nombre`" indica el nombre de un *script* en `/etc/rc.d/init.d`.

KXXnombre donde "`nombre`" indica el nombre de un *script* en `/etc/rc.d/init.d`.

Cuando se entra en el nivel de ejecución correspondiente, en este caso el tres, se ejecutan aquellos *scripts* referenciados por los enlaces que empiezan con "**S**", de start. Y el orden en el que se ejecutan lo da "**XX**". Se empieza por `S00`, luego `S01`, `S02`, ... En caso de que no exista, por ejemplo `S03`, prueba con `S04`, `S05`, ...

También podemos ver enlaces que empiezan con "**K**". Cuando salimos de un nivel de ejecución hemos de parar todos los servicios que hay iniciados, esto se hace mediante los enlaces que empiezan con "**K**" y el orden nos lo da "`XX`".

8.4.4. Manejo de servicios manualmente

Hay veces que es necesario manejar servicios de forma manual. Por ejemplo:

- Si se cambió algo en la configuración del servicio es necesario rearrancar el demonio que lo maneja para que la nueva configuración tenga efecto.
- Hay veces que es necesario arrancar un servicio que no se arrancó. Por ejemplo si nuestra máquina no va sobrada de potencia hay veces que para un buen rendimiento del sistema conviene tener parado el servidor de páginas web y arrancarlo unicamente cuando lo necesitemos.
- Hay veces que es necesario parar algún servicio. Por ejemplo si hemos arrancado el servidor de páginas web y ya no lo necesitamos, puede que sea conveniente pararlo ya que nuestro equipo no va sobrado.

Supongamos que hemos cambiado la configuración de la tarjeta de red, entonces vamos a directorio `/etc/rc.d/init.d`:

```
[root@localhost root]# cd /etc/rc.d/init.d
[root@localhost init.d]# ./network
Uso: ./network {start|stop|restart|reload|status|probe}
[root@localhost init.d]#
```

Ejecutando el *script* correspondiente nos da los parámetros que podemos pasarle. Estos parámetros suelen ser estándar y su significado es:

start inicia el servicio.

stop para el servicio.

restart reinicia el servicio.

reload carga los nuevos parámetros de la red.

status nos da el estado del servicio.

probe probar el servicio.

Si hemos cambiado la configuración seguramente nos valga con recargar los nuevos parámetros:

```
[root@localhost init.d]# ./network reload
Interrupcion de la interfaz eth0:                          [  OK  ]
Interrupcion de la interfaz eth1:                          [  OK  ]
Configurando parametros de red:                            [  OK  ]
Activando interfaz lo:                                     [  OK  ]
Activando interfaz eth0:                                   [  OK  ]
Activando interfaz eth1:                                   [  OK  ]
[root@localhost init.d]# ./network status
Dispositivos configurados:
lo eth0 eth1
Dispositivos activos en el momento:
eth0 eth1 lo
[root@localhost init.d]#
```

8.4.5. El comando `runlevel`

Este comando nos permite ver el nivel de ejecución anterior y el actual:

```
[root@icarus root]# runlevel
N 3
[root@icarus root]#
```

El primer número nos indica el nivel de ejecución anterior, en este caso la "`N`" nos indica que ninguno (se inició la máquina en el nivel de ejecución actual). El siguiente número nos indica el nivel de ejecución actual.

Este comando, normalmente, sólo es ejecutable por el superusuario.

8.4.6. Cambio del nivel de ejecución, el comando `init`

Mediante este comando el superusuario, o usuarios privilegiados dedicados a la administración de la máquina, pueden cambiar de un nivel de ejecución a otro. Basta con ejecutar `init` seguido del nivel de ejecución deseado.

Al cambiar de nivel de ejecución se ejecutarán los *scripts* referenciados en `/etc/rc.d/rcN.d`, donde `N` nos indica el nivel de ejecución actual, y que empiecen por "**K**" para matar los servicios iniciados y seguidamente se ejecutarán los *scripts* referenciados en `/etc/rc.d/rcM.d`, donde `M` nos indica el nivel de ejecución en el que vamos a entrar, y que empiecen por "**S**" para iniciarlos.

8.5. El proceso `init`

Una vez que arranca el nucleo crea el proceso `init`, este es el padre del resto de procesos que se crearán en el sistema, y, es el encargado de continuar con el arranque y arrancar los servicios que queramos que esten disponibles en nuestra máquina.

```
[jose@localhost jose]$ ps -p 1
  PID TTY          TIME CMD
    1 ?        00:00:04 init
[jose@localhost jose]$
```

8.5.1. El fichero `/etc/inittab`

Cuando el proceso `init` es arrancado utiliza este fichero para activar los servicios que estarán presentes en la máquina.

La sintaxis de este fichero es muy sencilla, cada linea describe una acción y es de la siguiente forma:

```
id:runlevels:action:process
```

donde:

id identificador, cuatro caracteres máximo, de cada entrada en el fichero.

runlevels indica el nivel de ejecución en el que se realizará esta acción.

action acción a realizar.

process indica el proceso a realizar.

Para una mejor comprensión de este fichero consultar la página del manual.

Entre otras cosas en este fichero se especifica el nivel de ejecución en el que entrará la máquina por defecto:

```
id:3:initdefault:
```

Le indica a `init` el nivel de ejecución que ha de utilizar por defecto.

Si queremos permitir a los usuarios que reinicien el sistema pulsando `Ctrl + Alt + Sup`, en el teclado conectado físicamente al sistema:

```
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

8.5.2. El fichero `/etc/initscript`

Cuando este *shell script* está presente el proceso `init` lo utiliza para ejecutar los comandos que encuentre en `/etc/inittab`.

Capítulo 9

El demonio `inetd` y los servicios de red

`inetd` es un demonio especial que es conocido como "*super-demonio*" o también como "*super-servidor*". Es, sin lugar a dudas, el más importante de todos los demonios.

9.1. ¿Qué tiene de especial `inetd`?

Como ya hemos dicho también se le conoce con los nombres de "*super-servidor*" o "*super-demonio*".

Para todos los servicios que necesitemos prestar en nuestra máquina necesitaremos un demonio que se encarge de escuchar en un determinado puerto. Esto implica el tener muchos demonios ejecutandose a la vez, escuchando en diferentes puertos a la espera de peticiones, con lo cual se reducen los recursos de la máquina.

Para evitar este desperdicio se recurrió a utilizar un demonio "*especial*", `inetd`, encargado de gestionar otros demonios.

9.1.1. ¿Maneja `inetd` todos los demonios?

`inetd` puede manejar todos los demonios del sistema, pero normalmente no lo hace. Este demonio se utiliza para manejar aquellos servicios que menos se usan.

9.2. Funcionamiento de `inetd`

`inetd` controla a aquellos demonios encargados de los servicios que menos se utilizan. El tener arrancados los demonios que controlan aquellos servicios que menos se utilizan es un desperdicio de recursos.

Para evitar esto lo que se hace es lanzar esos demonios bajo el control de `inetd`. `inetd` se encarga de vigilar los puertos y cuando recibe una petición se encarga de arrancar el demonio correspondiente.

Aunque parezca una buena idea el lanzar todos los demonios del sistema bajo el control de `inetd`, no lo es.

Iniciar un proceso en una tarea costosa dentro de un sistema, ya que tiene que acceder a datos dentro del disco duro (las velocidades de acceso y lectura en discos son lentas, comparadas con la memoria física) además debe reservar espacio en memoria, posiblemente descargar alguna página de memoria al disco duro, ... Si lanzaramos un servicio que reciba peticiones a menudo, como un servidor web, dentro de `inetd` nuestra máquina tendría una carga bastante alta y sería muy probable que se produjera una denegación de servicio[1].

Por ello los servicios que no reciben peticiones de forma continuada son arrancados por `inetd`, como por ejemplo:

fingerd el demonio del servicio `finger`. Un servicio que nos ofrece información sobre los usuarios del sistema. No es muy recomendable el utilizar este servicio ya que suele ser utilizado para recoger información sobre un sistema y luego atacarlo.

in.telnetd el demonio del servicio `telnet`. Nos permite crear conexiones remotas desde cualquier máquina utilizado el programa `telnet`.

in.ftpd el demonio del servicio `ftp`. Nos permite trasferir ficheros de un sistema a otro.

[1] Uno o todos los servicios de la máquina dejarían de responder debido a la sobrecarga del sistema.

9.2.1. El fichero /etc/services

Este fichero lo utiliza `inetd` para saber que puerto y protocolo utiliza cada servicio. Los puertos para los servicios más comunes estan estandarizados, por ejemplo:

finger utiliza el puerto 79.

telnet utiliza el puerto 23.

ftp utiliza el puerto 21.

http utiliza[2] el puerto 80.

En este fichero se encuentran todos los servicios y sus puertos, los controle `inetd` o no. Cada línea del fichero determina un servicio. Por ejemplo:

```
telnet 23/tcp
telnet 23/udp
```

Estas líneas nos dicen que el servicio `telnet` usará el puerto 23 y como protocolo de transporte `tcp` y `udp`. Además es posible añadir otra columna especificando un nombre alternativo para el servicio (alias).

9.2.2. El fichero /etc/protocols

En este fichero se encuentran los protocolos del sistema y su número de protocolo, que es el que entiende el protocolo TCP/IP.

[2]Normalmente no es manejado por `inetd`.

9.2.3. El fichero /etc/inetd.conf

Este fichero contiene la lista de servicios que tiene que arrancar el demonio `inetd`. Cada línea representa a un servicio y son de la forma:

```
telnet stream tcp nowait root /usr/sbin/telnetd in.telnetd
```

donde cada campo significa:

telnet nombre del servicio. Mediante el fichero `/etc/services` `inetd` obtiene su puerto.

stream indica el tipo de socket. Puede tener dos valores:

- **stream** para protocolos orientados a conexión (servicios basados en TCP siempre *stream*).
- **dgram** para protocolos no orientados a conexión (servicios basados en UDP siempre *dgram*).

tcp indica el protocolo. Debe estar listado en `/etc/protocols`

nowait se utiliza para sockets UDP y puede tomar dos valores:

- **wait** `inetd` ejecuta un sólo servidor para el puerto especificado.
- **nowait** `inetd` ejecuta varios servidores. Los sockets de tipo *stream* deben ser de este tipo.

root usuario bajo el que se ejecutará.

/usr/sbin/telnetd ruta completa al servidor encargado del servicio.

in.telnetd comando a ejecutar.

9.2.4. El demonio tcpd

Este demonio se utiliza para negar/conceder el acceso a nuestros servicios TCP a todos aquellos que los soliciten.

Cuando arrancamos un demonio con `tcpd` este demonio se encarga de verificar si el cliente que realiza la petición esta autorizado para ese servicio.

La forma de comprobar si esta autorizado para el acceso es comprobando los ficheros:

`/etc/hosts.allow` y `/etc/hosts.deny`

y lo hará de la siguiente forma:

1. Si en `/etc/hosts.allow` esta concedido el acceso para la máquina que solicita el servicio `tcpd` arrancará el demonio correspondiente.
2. Si en `/etc/hosts.deny` esta denegado el acceso a la máquina que solicita el servicio `tcpd` no lo arrancará.
3. Si no existe ninguno de los dos ficheros anteriores se concederá el acceso.

`tcpd` también informa al demonio `syslog` de las peticiones de servicios.

Por motivos de seguridad todos los servicios que controle `inetd` deberán ser lanzados con este demonio. Por ejemplo:

```
telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd
```

9.3. Implementación de un pequeño sistema de DNS

El servicio de D.N.S.[3] es el servicio que se encarga de traducir direcciones del tipo:

`www.augcyl.org`

a formato numérico que es el que entienden los ordenadores. Muchas veces tenemos en casa montada una pequeña red local, (LAN), y tenemos que acceder a todos los equipos mediante direcciones del tipo:

`192.168.0.1`

lo cual es bastante incomodo. Podemos montar un pequeño sistema, que no servidor, de DNS en nuestro **GNU/Linux**. Con esto desde nuestro **GNU/Linux** podremos acceder al resto de ordenadores mediante nombres simbólicos como:

`bender.futurama.net` o `lila.futurama.net`

en lugar de acceder mediante sus direcciones IP:

`192.168.0.1` o `192.168.0.2`

Este tipo de acceso no es recíproco, es decir, desde `bender.futurama.net` no podrán acceder a nosotros utilizando el nombre simbólico de nuestra máquina.

[3] **D**omain **N**ame **S**erver o servidor de nombres de dominio.

9.3.1. El archivo /etc/hosts

Mediante este archivo podemos tener un pequeño sistema de D.N.S. No confundir con un servidor de D.N.S. ya que este archivo sólo lo puede utilizar nuestra máquina para traducir direcciones simbólicas a direcciones IP.

Si quisieramos que otras máquinas **GNU/Linux** en nuestra red pudieran utilizarlo tendríamos que copiar este fichero en su directorio /etc/.

Si disponemos de una pequeña red local de máquinas, podemos listar todas las máquinas con sus direcciones IP y nombres simbólicos en este fichero, independientemente del sistema operativo que utilicen, y podremos acceder a ellas con su nombre simbólico. Esto es factible si nuestra red es pequeña, ya que hay que tener duplicado este archivo en todos y cada uno de los equipos **GNU/Linux**, cada vez que se quite o añada un nuevo equipo a la red tendremos que modificar el equipo.

Un ejemplo de este archivo sería:

```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               localhost localhost.localdomain localhost
192.168.0.1             bender bender.futurama.net bender
192.168.0.2             lila lila.futurama.net lila
```

```
[jose@localhost jose]$ ping localhost
PING localhost (127.0.0.1) from 127.0.0.1 : 56(84) bytes of data.
Warning: time of day goes back, taking countermeasures.
64 bytes from localhost (127.0.0.1): icmp_seq=0 ttl=255 time=2.765 msec
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=255 time=268 usec
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=255 time=276 usec

--- localhost ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.268/1.103/2.765/1.175 ms
[jose@localhost jose]$
```

de esta forma podemos acceder a los equipos utilizando su nombre simbólico.

Si queremos forzar a todas las aplicaciones a utilizar /etc/hosts para resolver los nombres entonces deberemos editar el fichero /etc/host.conf y añadir:

```
order hosts
```

9.3.2. El fichero `/etc/networks`

Este fichero es equivalente al anterior, pero en lugar de ser utilizado para nombres de máquinas es utilizado para almacenar nombres y IP's de redes.

9.4. Control de accesos con TCP Wrappers

El control de accesos normalmente se suele hacer a traves de un "*cortafuegos*" o "*firewall*". Hay veces que no es posible el uso de un "*cortafuegos*".

Para ello se utiliza el demonio `tcpd`[4]. Ya hemos visto su uso en `inetd`.

Mediante este demonio antes de lanzar un servidor para que se haga cargo de una petición comprueba que la máquina que ha realizado la petición tiene permisos para utilizar ese servicio.

9.4.1. ¿Como funciona TCP Wrappers?

Cuando `tcpd` tiene que comprobar una petición lo hace de la siguiente forma:

1. Si en `/etc/hosts.allow` esta concedido el acceso para la máquina que solicita el servicio `tcpd` arrancará el demonio correspondiente.
2. Si en `/etc/hosts.deny` esta denegado el acceso a la máquina que solicita el servicio `tcpd` no lo arrancará.
3. Si no existe ninguno de los dos ficheros anteriores se concederá el acceso.

Por cada petición que gestiona `tcpd` informa al demonio de *logs* del sistema `syslog`.

9.4.2. El lenguaje `hosts_options`

TCP Wrappers fue escrito por Wietse Venema y para controlar los accesos mediante los ficheros `/etc/hosts.allow` y `/etc/hosts.deny` se utiliza en lenguaje especial llamado `hosts_options`. No explicaremos aquí sus fundamentos. Veremos unos ejemplos de su uso. Si deseas más información:

```
man hosts_options
```

[4] Apartado 9.2.4 en la página 148.

9.4.3. El fichero /etc/hosts.allow

En este fichero se encuentran los servicios que son accesibles desde el exterior de la máquina, lanzados mediante `tcpd`, y por quien son accesibles.

La sintaxis es muy sencilla:

```
demonio : lista de clientes autorizados
```

Por ejemplo:

```
in.telnetd : .futurama.net EXCEPT bender.futurama.net
```

Permitiría el acceso por `telnet` a todas las máquinas de la red `futurama.net` con la excepción de la máquina `bender.futurama.net`.

Podemos autorizar a cualquier máquina a todos nuestros servicios mediante:

```
ALL : ALL
```

Para una mayor información:

```
man hosts_options
```

ya que se pueden hacer muchas más combinaciones de las que hemos visto.

9.4.4. El fichero /etc/hosts.deny

En este fichero se encuentran los servicios que no son accesibles desde el exterior de la máquina, lanzados mediante `tcpd`, y a quien se le niega el acceso.

La sintaxis es muy sencilla:

```
demonio : lista de clientes no autorizados
```

Es una práctica habitual el negar todas las conexiones a todos los servicios y luego permitir las conexiones necesarias en `/etc/hosts.allow`. Esto se hace:

```
ALL : ALL
```

Si quisieramos negar todos nuestros servicios a todo el mundo, pero permitir a una máquina determinada el acceso por `telnet`:

```
ALL EXCEPT in.telned: bender.futurama.net
```

Para una mayor información:

```
man hosts_options
```

ya que se pueden hacer muchas más combinaciones de las que hemos visto.

9.4.5. El comando `tcpdchk`

Este comando se utiliza para comprobar si la sintaxis de los ficheros `/etc/hosts.allow` y `/etc/hosts.deny` es la correcta.

9.4.6. El comando `tcpdmatch`

Este comando se utiliza para comprobar como se comportaran los TCP Wrappers ante la petición de una determinada máquina. Nos permite ver si hemos configurado bien los ficheros `/etc/hosts.allow` y `/etc/hosts.deny` para permitir/negar conexiones a determinadas máquinas.

Capítulo 10

Automatización de tareas

En todo sistema informático existen tareas que hay que realizar de forma periódica, como copias de seguridad (back-ups), eliminación de ficheros temporales, eliminación de ficheros `core`, busqueda de ficheros SUID, ...

GNU/Linux permite la automatización de estas tareas realizandolas el sistema. En este capítulo veremos como podemos planificar todos estos trabajos.

10.1. El comando `run-parts`

Este comando lo que hace es ejecutar "TODOS" los scripts contenidos dentro del directorio especificado como parámetro. Es buena idea el indicar la ruta completa del directorio desde el raíz.

La utilidad de este comando la veremos cuando veamos `cron` y `anacron`.

10.2. Ejecución de tareas con `batch`

El comando `batch` lee un comando desde la entrada estándar y lo ejecuta cuando el sistema no tiene mucha carga. Este comando es ideal para la ejecución de ciertas tareas no urgentes, ya que las tareas lanzadas con este comando no se realizarán hasta que la carga del sistema sea mínima, de esta forma no se verá perjudicado el rendimiento del sistema.

Normalmente, según la página del manual, los comandos son lanzados cuando la carga del sistema está por debajo del 0'8 %.

Un ejemplo del uso de este comando sería:

```
[jose@localhost jose]$ batch
warning: commands will be executed using (in order)
         a) $SHELL b) login shell c) /bin/sh
at> find /home/jose -name core -exec rm {} \;
at> <EOT>
job 2 at 2002-04-08 13:36
[jose@localhost jose]$
```

Con esto lo que haríamos sería buscar en nuestro directorio de trabajo todos los archivos `core` y borrarlos. Una vez que hemos hemos introducido los comandos en el prompt `at>` para finalizar deberos pulsar `Control+D`, que indica `EOF` o fin de fichero. Entonces cuando la carga del sistema lo permita se lanzará el comando `find` o los comandos que le hayamos dicho.

También podemos ejecutar comandos desde un fichero, por ejemplo para lanzar con `batch` el comando anterior crearemos un fichero, `batch.dat`, cuyo contenido sea:

```
find /home/jose -name core -exec rm {} \;
```

y a continuación lo lanzamos con `batch` redireccionado la entrada estándar al fichero:

```
[jose@localhost jose]$ batch < batch.dat
warning: commands will be executed using (in order)
         a) $SHELL b) login shell c) /bin/sh
job 5 at 2002-04-08 13:50
[jose@localhost jose]$
```

`batch` en realidad es un "*shell script*" que lanza el comando `at` con una serie de opciones.

10.2.1. La cola de trabajos de `batch`

Todos los trabajos lanzados con `batch` son almacenados en una cola de trabajos, por defecto esta cola reside en `/var/spool/at` y por defecto es un fichero que recibe por nombre `b`. Si editas el script `/usr/bin/batch` podrás verlo y cambiarlo a tu gusto, si eres `root`.

10.3. El comando at

Mediante este comando podemos lanzar una serie de comandos a una hora en concreto, sin necesidad de estar conectados para ello.

10.3.1. El demonio atd

Este demonio se encarga de llevar a cabo los trabajos encolados con los comandos `at` y `batch`. Se arranca con el sistema y periodicamente mira la cola de trabajos pendientes y los va realizando de acuerdo con las especificaciones de tiempo que se le dieron.

10.3.2. ¿Quién puede utilizar at y batch?

Puede ser peligroso el permitir a todos los usuarios estos comandos, ya que podrían dedicarse a lanzar comandos a diestro y siniestro y colapsar el sistema.

Para evitar esto existen dos ficheros `/etc/at.allow` y `/etc/at.deny` que determinan quien puede utilizar los comandos `at` y `batch`. Cuando se ejecuta uno de estos dos comandos se realizan las siguientes comprobaciones para ver si el usuario puede utilizarlos:

1. Se comprueba si existe `/etc/at.allow` si existe y el usuario está listado en dicho fichero se le permite el uso de `at` o `batch`.

2. En caso de que no exista `/etc/at.allow` se comprueba si existe `/etc/at.deny` y si existe y el usuario esta listado en el se le deniega el uso de `at` y `batch`.

3. En caso de no existir ninguno de los dos sólo el `root` podrá utilizar `at` y `batch`.

10.3.3. El fichero /etc/at.allow

Si este archivo esta presente en el sistema, unicamente los usuarios que esten listados en el podrán lanzar trabajos con los comandos `at` y `batch`. Por ejemplo:

```
[bender@localhost bender]$ at
You do not have permission to use at.
[bender@localhost bender]$
```

El contenido del fichero `/etc/at.allow` sería:

```
#
# at.allow
#

jose
lila
```

Entonces sólo podrían utilizar los comandos `at` y `batch` los usuarios `jose` y `lila`. Aunque el usuario `root` no aparezca listado siempre podrá utilizar `at` y `batch`.

10.3.4. El fichero /etc/at.deny

Si este archivo esta presente en el sistema y no existe el archivo `/etc/at.allow`, los usuarios que esten listados en el no podrán lanzar trabajos con los comandos `at` y `batch`. Por ejemplo:

```
[bender@localhost bender]$ at
You do not have permission to use at.
[bender@localhost bender]$
```

El contenido del fichero `/etc/at.deny` sería:

```
#
# at.deny
#

bender
```

Entonces `bender` no podría utilizar los comandos `at` y `batch`, el resto de usuarios si podrían.

Si quieres que todo el mundo en el sistema pueda utilizar at y batch basta con que crees el fichero /etc/at.deny vacio:

```
[root@localhost root]# touch /etc/at.deny
[root@localhost root]#
```

10.3.5. La cola de trabajos de at, /var/spool/at

Cada vez que se lanza un trabajo con at se almacena en una cola que gestiona el demonio atd. Por defecto esas colas se almacenan en el directorio /var/spool/at. Cualquier trabajo encolado se encola, por defecto, en la cola a:

```
[root@localhost root]# ls -l /var/spool/at
total 12
-rwx------    1 jose     users        1661 abr  8 16:43 a000060102fa98
-rwx------    1 root     root         1426 abr  8 16:49 a000070102fa98
drwx------    2 daemon   daemon       4096 abr  8 13:50 spool
[root@localhost root]#
```

aqui podemos ver como tenemos dos trabajos encolados en la cola a. Podemos ver los trabajos que hay en la cola de la siguiente forma:

```
[root@localhost root]# at -l
6       2002-04-09 12:00 a jose
7       2002-04-09 12:00 a root
[root@localhost root]#
```

10.3.6. Uso de at

Como ya hemos dicho con el comando at podemos dejar trabajos que se ejecutaran a una hora determinada. Todos los trabajos lanzados con at se almacenan en una cola y el demonio atd se encarga de realizarlos cuando llega el momento.

Por defecto la cola de trabajos predeterminada recibe el nombre de a, pudiendose encolar trabajos en otras colas si estuvieran predefinidas, en caso de duda pregunte a su administrador. Por supuesto para poder encolar trabajos usted debe estar autorizado a usar at y batch.

10.3.7. Como ver los trabajos encolados

Ya hemos visto como se hace, si el usuario `root` lo solicita verá todos los trabajos pendientes, mientras que si un usuario particular lo solicita sólo verá sus trabajos.

```
[root@localhost root]# at -l
6       2002-04-09 12:00 a jose
7       2002-04-09 12:00 a root
8       2002-04-09 12:00 c jose
[root@localhost root]#
```

Veamos que significa cada una de esas columnas:

- La primera columna nos muestra el número del trabajo.
- La segunda columna nos muestra la fecha en la que se efectará el trabajo.
- La tercera columna nos muestra la hora a la que se efectuará el trabajo.
- La cuarta columna nos muestra la cola en la que esta encolado el trabajo.
- La quinta columna nos muestra el propietario del trabajo.

10.3.8. Como eliminar trabajos, el comando `atrm`

Podemos eliminar nuestros trabajos encolados, o si somos el superusuario todos los trabajos utilizando el comando `atrm`. Su uso es muy sencillo, basta con especificar el número del trabajo que queremos eliminar:

```
[jose@localhost jose]$ atrm 6
[jose@localhost jose]$
```

esto eliminará el trabajo número 6. Si intentáramos eliminar un trabajo que no es nuestro, por ejemplo el siete, ya que nuestros trabajos eran el 6 y el 8 debería haber un trabajo con el número 7 que no sabemos a quien pertenece:

```
[jose@localhost jose]$ atrm 7
7: Not owner
[jose@localhost jose]$
```

Vemos que no nos deja ya que no somos el propietario.

10.3.9. Como encolar trabajos

Podemos encolar trabajos desde la línea de comandos o desde un archivo. Es necesario especificar una hora, en caso de no especificar una fecha el trabajo se realizará en las próximas veinticuatro horas.

Desde la línea de comandos

La forma de hacerlo es igual a como se hacia con el comando `batch`.

```
[jose@localhost jose]$ at 18:55 -m
warning: commands will be executed using (in order)
         a) $SHELL b) login shell c) /bin/sh
at> find /home/jose -name core | xargs rm
at> <EOT>
job 9 at 2002-04-08 17:20
[jose@localhost jose]$
```

Hemos lanzado un trabajo que se ejecutará a las 18:55 y cuando se realice mandará un aviso por correo, flag "`-m`", al propietario. Una vez que hemos terminado de introducir los comandos que queremos que se ejecuten tendremos que pulsar `Control+D`.

Desde un fichero

Para hacerlo desde un fichero deberemos escribir las ordenes a ejecutar en un fichero y podemos hacerlo de dos formas:

1. Redirigiendo la entrada estándar:

   ```
   [jose@localhost jose]$ at 18:55 < at.dat
   warning: commands will be executed using (in order)
            a) $SHELL b) login shell c) /bin/sh
   job 10 at 2002-04-08 17:23
   [jose@localhost jose]$
   ```

   Lo cual ejecutaría los comandos contenidos en el fichero `at.dat` a la hora especificada.

2. Utilizando el flag "`-f`" seguido del nombre del fichero:

```
[jose@localhost jose]$ at 18:55 -f at.dat
warning: commands will be executed using (in order)
         a) $SHELL b) login shell c) /bin/sh
job 11 at 2002-04-08 17:26
[jose@localhost jose]$
```

   Lo cual ejecutaría los comandos contenidos en el fichero `at.dat` a la hora especificada.

10.3.10. Creación de nuevas colas

Para crear nuevas colas sólo hay que especificar la nueva cola con el parametro "`-q`":

```
[jose@localhost jose]$ at 18:55 -q b < at.dat
warning: commands will be executed using (in order)
         a) $SHELL b) login shell c) /bin/sh
job 12 at 2002-04-09 12:00
[jose@localhost jose]$
```

donde `b` es la cola en la que se encolará el trabajo a realizar especificado en el fichero `at.dat`.

10.3.11. Especificación de las fechas

Si quieres ver la especificación de las fechas suele estar en el fichero `timespec`, que normalmente estarían en uno de estos dos directorios:

`/usr/doc/at-x.y.z` o `/usr/share/doc/at-x.y.z`

Con el comando `at` se pueden utilizar una serie de palabras:

midnight 12:00 a.m. o 0:00.

noon 12:00 p.m.

now en este instante.

today hoy.

tomorrow mañana.

También podemos especificar incrementos de tiempo:

minutes minutos.

hours horas.

days días.

weeks semanas.

months meses.

years años.

Por ejemplo:

```
[jose@localhost jose]$ at now +3 hours < at.dat
warning: commands will be executed using (in order)
         a) $SHELL b) login shell c) /bin/sh
job 18 at 2002-04-08 21:52
[jose@localhost jose]$
```

ejecutará el contenido de `at.dat` en tres horas.

10.4. El comando `cron`

Este comando es similar a `at`, pero se diferencia en que este comando se utiliza para la realización periódica de tareas. Con `at` podiamos lanzar un comando y una vez ejecutado ya no se volverá a ejecutar. Mediante `cron` podemos realizar trabajos que se han de realizar con una periodicidad determinada, por ejemplo semanalmente, diariamente, mensualmente, anualmente.

Este comando es imprescindible para una buena administración de un sistema, existen versiones de este comando para otros sistemas no `UNIX` como Windows.

10.4.1. El demonio `crond`

Este demonio se arranca con el sistema y se encarga de buscar en sus ficheros de configuración y colas los trabajos a realizar y realizarlos.

10.4.2. ¿Quién puede utilizar cron?

Puede ser peligroso el permitir a todos los usuarios el uso de `cron`, ya que podrían dedicarse a lanzar comandos a diestro y siniestro y colapsar el sistema.

Para evitar esto existen dos ficheros `/etc/cron.allow` y `/etc/cron.deny` que determinan quien puede utilizar `cron`. Cuando se pretende utilizar `cron` se realizan las siguientes comprobaciones para ver si el usuario puede utilizarlo:

1. Se comprueba si existe `/etc/cron.allow` si existe y el usuario está listado en dicho fichero se le permite el uso de `cron`.
2. En caso de que no exista `/etc/cron.allow` se comprueba si existe `/etc/cron.deny` y si existe y el usuario esta listado en el se le deniega el uso de `cron`.
3. En caso de no existir ninguno de los dos sólo el `root` podrá utilizar `cron`.

10.4.3. El fichero /etc/cron.allow

Si este archivo esta presente en el sistema, unicamente los usuarios que esten listados en él podrán utilizar `cron`.

Si el contenido del fichero `/etc/cron.allow` es:

```
#
# cron.allow
#

jose
lila
```

Entonces sólo podrían utilizar `cron` los usuarios `jose` y `lila`. Aunque el usuario `root` no aparezca listado siempre podrá utilizar `cron`.

10.4.4. El fichero /etc/cron.deny

Si este archivo esta presente en el sistema y no existe el archivo `/etc/cron.allow`, los usuarios que esten listados en el no podrán lanzar utilizar `cron`.

Si el contenido del fichero `/etc/cron.deny` es:

```
#
# cron.deny
#

bender
```

Entonces `bender` no podría utilizar `cron`, el resto de usuarios si podrían.

10.4.5. El fichero /etc/crontab

Este fichero controla los trabajos a relizar por `crond`. Las instrucciones que hay en este fichero le indican a `crond` que comando debe ejecutar y cuando tiene que hacerlo.

Este fichero sólo es editable por el administrador del sistema, con lo cual nuestras tareas no podremos especificarlas utilizando este fichero.

Este fichero esta compuesto de varias filas, cada fila indica un trabajo. Además todas las filas estan divididas en siete columnas:

- La primera columna indica los minutos. Puede tomar los valores enteros comprendidos entre 0 y 59.
- La segunda columna indica las horas. Puede tomar los valores enteros comprendidos entre 0 y 23.
- La tercera columna indica el día del mes. Puede tomar los valores enteros comprendidos entre 1 y 31.
- La cuarta columna indica el mes. Puede tomar los valores enteros comprendidos entre 1 y 12 o también los nombres de los meses[1].
- La quinta columna indica el dia de la semana. Puede tomar los valores enteros comprendidos entre 1 y 7 o también nombres[2].

[1] Ver `man 5 crontab`.
[2] Ver `man 5 crontab`.

- La sexta columna nos indica el usuario que va a ejecutar el comando especificado en el campo siguiente.
- La septima columna indica el comando a ejecutar.

En cada columna se pueden:

- Emplear rangos. Por ejemplo si en la columna de las horas tenemos "`2-4`" significa que ejecutará el comando a las horas "`2`", "`3`" y "`4`".
- Emplear listas. Por ejemplo si en la columna de los minutos tenemos:
  - "`1,3,4,5,8`" significa que habrá ejecución en los minutos "`1`", "`3`", "`4`", "`5`" y "`8`".
  - Analogamente con "`0-4,8-12`".
- Podemos emplear intervalos con los rangos. Por ejemplo si en la columna de los meses tenemos "`1-12/2`" significa que habrá ejecución del primer mes al último cada dos meses.

Si en un campo aparece "`*`" significa que habrá ejecución todos los minutos, si esta en la columna de los minutos.

La mejor forma de comprenderlo es con un ejemplo:

```
# A las 0:30 y a las 12:30 de cada dia buscara fichero SUID y se
# lo notificara al root
30 */12 * *   * root find / -perm +4000 | mail root
# A las horas en punto durante los meses del 4 al 8 borrara todos
# los archivos core que encuentre en el sistema
0  *    * 4-8 * root find / -name core | xargs rm
```

Los comentarios deben estar en una línea aparte de los trabajos.

Una buena idea si deseamos ejecutar varios scripts a la vez es incluirlos todos en un directorio y ejecutarlos con el comando `run-parts`. Con esto haremos que el fichero `crontab` sea más pequeño:

```
# A las 0:30 y a las 12:30 de cada dia ejecutara todos los
# scripts del directorio /root/backup
30 */12 * *   * root run-parts /root/backup
```

también reduciremos el uso de memoria ya que este fichero se carga en memoria al arrancar el demonio `crond`. Además cada vez que el demonio `crond` "*despierta*" comprueba el fichero para ver si ha cambiado, y si lo ha hecho lo carga en memoria.

10.4.6. Cron y los usuarios normales, el comando crontab

Como ya hemos dicho antes el fichero /etc/crontab sólo es editable por el administrador del sistema, o al menos debería serlo. Para que los usuarios normales puedan utilizar cron se dispone del comando crontab.

Para ejecutar comandos periodicamente bajo cron tendremos que crear un fichero con la misma sintaxis que /etc/crontab y ejecutarlo con el comando crontab. Supongamos que nuestro fichero es mistareas:

```
[jose@localhost jose]$ crontab mistareas
You (jose) are not allowed to use this program (crontab)
See crontab(1) for more information
[jose@localhost jose]$
```

Esto sucede ya que el usuario jose no tiene privilegios para el uso de cron. Cuando ejecutamos crontab se realizan las siguiente comprobaciones:

1. crontab comprueba si existe el fichero /etc/cron.allow, si existe los únicos usuarios en el sistema que podrán utilizar crontab serán los listados en él.

2. En caso de no existir comprobará si existe el fichero /etc/cron.deny y si existe los usuarios listados en él no podrán utilizar crontab.

3. Si ninguno de estos dos ficheros existen, lo más normal es que, nadie pueda utilizar crontab, con la excepción del superusuario.

10.4.7. La cola de trabajos de cron, /var/spool/cron

Cada vez que un usuario ejecuta crontab este se almacena en la cola de trabajos de cron. Dicha cola se almacena en /var/spool/cron.

Cuando un usuario ejecuta `crontab` se crea en este directorio un fichero con el nombre del usuario y conteniendo la información del fichero que le paso como argumento a `crontab`. Por ejemplo:

```
[root@localhost root]# ls -l /var/spool/cron
-rw-------    1 root     users         326 abr 12 18:27 jose
[root@localhost root]# cat /var/spool/cron/jose
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (cron.test2 installed on Fri Apr 12 18:27:24 2002)
# (Cron version -- $Id:!Adm-Basica.ps,v!1.1!2003/03/15!00:22:37!rrey!Exp!$)
# Cada 30min buscara y eliminara los ficheros .o de mi cuenta
30 */2 * *   * find /home/jose -name *.o | xargs rm
[root@localhost root]#
```

Hay que tener cuidado:

```
[jose@localhost jose]$ crontab tareas1
[jose@localhost jose]$ crontab tareas2
```

Al hacer esto la segunda invocación de `crontab` creará un fichero llamado `jose` en el directorio `/var/spool/cron` que sobreescribirá el existente de la invocación `crontab tareas1`.

10.5. El comando anacron

Este comando se utiliza para realizar acciones periodicamente, igual que `cron`. La diferencia con `cron` es que `cron` está pensado y diseñado para su uso en máquinas que estan encendidas las 24 horas del día.

Los servidores normalmente estan encendidos las 24 horas del día, pero si utilizamos nuestro ordenador con **GNU/Linux** como estación de trabajo seguramente no esté encendido 24 horas al día. En estos casos puede resultar aconsejable el utilizar `anacron` en lugar `cron`.

10.5.1. El fichero /etc/anacrontab

En este fichero se guardan los trabajos a realizar con anacron. Este fichero tiene cuatro campos:

1. El primer campo que especifica cada cuantos días se ha de ejecutar el trabajo.
2. El segundo campo indica la demora en minutos.
3. El tercer campo es una cadena de caracteres para identificar el trabajo.
4. El cuarto campo es el trabajo a realizar.

Un ejemplo de este fichero podría ser:

```
# /etc/anacrontab: configuration file for anacron
# See anacron(8) and anacrontab(5) for details.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# These entries are useful for a Red Hat Linux system.
1       5       cron.daily      run-parts /etc/cron.daily
7       10      cron.weekly     run-parts /etc/cron.weekly
30      15      cron.monthly    run-parts /etc/cron.monthly
```

Este fichero se encarga de realizar las tareas de cron. En el apartado siguiente se explica el funcionamiento de anacron.

En este ejemplo podemos ver el uso del comando run-parts, con el cual reducimos el tamaño de dicho fichero.

10.5.2. ¿Como funciona anacron?

Cuando es ejecutado anacron lee la lista de trabajos del fichero /etc/anacron y realiza los trabajos que sean necesarios.

Normalmente anacron es arrancado en el inicio del sistema, comprueba si tiene que realizar algún trabajo, en caso afirmativo lo realiza y luego termina su ejecución.

Cada vez que el sistema se arranca `anacron` comprueba si se han realizado los trabajos que tiene especificados. Para ello comprueba si cada comando se ha ejecutado en los últimos n días, donde n es un parámetro especificado en el fichero `anacrontab`. En caso de que no haya sido ejecutado durante ese periodo de tiempo `anacron` espera m minutos, donde m es la demora especificada en `anacrontab`, para realizarlo.

Una vez realizado guarda la fecha en la que se ejecutó por última vez en un fichero especial en el directorio `/var/spool/anacron`. Para esto último sólo utiliza la fecha, nunca la hora.

10.5.3. La cola de trabajos de `anacron`, `/var/spool/anacron`

En este directorio `anacron` guarda cuando fue la última vez que se ejecutaron los trabajos de los que es responsable. De esta forma cada vez que arranca el sistema y se arranca `anacron` puede comprobar si ya se ha realizado cada trabajo, en caso contrario lo realiza y actualiza el fichero correspondiente.

```
[root@localhost root]# ls -l /var/spool/anacron
total 12
-rw-------    1 root     root            9 abr 12 13:39 cron.daily
-rw-------    1 root     root            9 mar 15 21:55 cron.monthly
-rw-------    1 root     root            9 abr 12 13:48 cron.weekly
[root@localhost root]#
```

10.5.4. ¿Quién puede utilizar `anacron`?

`anacron` sólo puede ser utilizado por el superusuario, ya que el fichero en el que se establecen los trabajos, `/etc/anacrontab`, sólo puede ser modificado por el superusuario.

Es posible hacer que otros usuarios utilicen `anacron` dandoles permisos de ejecución, y acceso al directorio `/sbin` y permisos de escritura en el fichero `/etc/anacron`. Esto no es muy aconsejable salvo para usuarios que realicen tareas de administración. Si algún usuario requiere la realización de tareas periodicas puede utilizar `batch`, `at` o `cron`.

Capítulo 11

Auditorías y Logs del sistema

En este capítulo veremos algunas cosas referentes a las auditorías dentro del sistema y a los ficheros de Log. Algo muy importante desde el punto de vista del Administrador del sistema.

11.1. ¿Quién esta en el sistema?

Algo que siempre debe conocer un administrador es quién está en el sistema en un momento determinado. En **GNU/Linux** existen varios comandos al respecto.

11.1.1. El comando `who`

Este comando nos da información sobre quien está conectado al sistema en el momento de su ejecución:

```
[root@localhost root]# who
jose     tty1     Apr  7 09:17
lila     tty2     Apr  7 09:54
bender   tty3     Apr  7 09:54
root     tty4     Apr  7 09:54
jose     pts/0    Apr  7 09:18
jose     pts/1    Apr  7 09:27
[root@localhost root]#
```

podemos ver quién está conectado y desde que terminal. Existen varios flags que resultan utiles:

- -H, imprime las cabeceras de las columnas.

```
[root@localhost root]# who -H
USUARIO  LINEA    HORA DE CONEXION DESDE
jose     tty1     Apr  7 09:17
lila     tty2     Apr  7 09:54
bender   tty3     Apr  7 09:54
root     tty4     Apr  7 09:54
jose     pts/0    Apr  7 09:18
jose     pts/1    Apr  7 09:27
[root@localhost root]#
```

- -u, muestra el tiempo que estuvo inactivo(idle) el terminal.

```
[root@localhost root]# who -Hu
USUARIO  LINEA    HORA DE CONEXION INACTIVO  DESDE
jose     tty1     Apr  7 09:17 00:43
lila     tty2     Apr  7 09:54 00:06
bender   tty3     Apr  7 09:54 00:06
root     tty4     Apr  7 09:54 00:06
jose     pts/0    Apr  7 09:18 00:42
jose     pts/1    Apr  7 09:27   .
[root@localhost root]#
```

- -q, lista unicamente el nombre de los usuarios y el número total de usuarios conectados al sistema.

```
[root@localhost root]# who -q
jose lila bender root jose jose
N de usuarios=6
[root@localhost root]#
```

11.1.2. El comando `w`

Este comando nos indica qué está haciendo cada usuario:

```
[root@localhost root]# w
 10:09am  up  1:03,  6 users,  load average: 0.07, 0.27, 0.24
USER     TTY      FROM      LOGIN@   IDLE   JCPU   PCPU  WHAT
jose     tty1     -         9:17am 52:21   2.10s  0.05s  /usr/X11R6/bin/startx
lila     tty2     -         9:54am 15:26   0.29s  0.20s  -bash
bender   tty3     -         9:54am 15:21   0.30s  0.17s  -bash
root     tty4     -         9:54am 15:00   0.27s  0.17s  -bash
jose     pts/0    -         9:18am 51:26   0.05s  0.05s  /bin/cat
jose     pts/1    -         9:27am  0.00s 60.70s  0.08s  w
[root@localhost root]#
```

La primera línea nos da la siguiente información:

> Hora actual, tiempo que lleva arrancada la máquina, número de usuarios conectados a la máquina, y una estimación de la carga del sistema.

Veamos el significado de cada una de las columnas:

USER usuario.

TTY terminal desde la que se esta conectado.

FROM desde donde esta conectado.

LOGIN@ hora en la que empezó la actual conexión.

IDLE tiempo que ha permanecido inactivo.

JCPU tiempo total de CPU usado por todos los procesos en el terminal.

PCPU tiempo total de CPU para todos los procesos activos en el terminal.

WHAT comando que esta siendo ejecutado en el terminal.

También podemos ver lo que está haciendo un determinado usuario:

```
[root@localhost root]# w jose
USER     TTY      FROM     LOGIN@   IDLE   JCPU   PCPU  WHAT
jose     tty1     -        9:17am  1:03m  2.11s  0.05s  /usr/X11R6/bin/startx
jose     pts/0    -        9:18am  1:02m  0.05s  0.05s  /bin/cat
jose     pts/1    -        9:27am  0.00s 52.31s  0.11s  w jose
[root@localhost root]#
```

11.1.3. El comando `users`

Este comando se utiliza para ver que usuarios estan presentes en el sistema:

```
[jose@localhost jose]$ users
jose jose jose jose root
[jose@localhost jose]$
```

Podemos ver que el usuario `jose` tiene abiertas varias sesiones.

11.1.4. El fichero `/var/run/utmp`

En este fichero se encuentra la información de los usuarios presentes en el sistema. Los comandos `who`, `w`, `users`, `finger` y `write` lo utilizan para saber que usuarios estan presentes en el sistema.

11.2. ¿Quién estuvo en el sistema?

GNU/Linux permite almacenar los inicios de sesión y los intentos fallidos de conexión. Esto permite realizar estadísticas de uso del sistema y permite conocer potenciales intentos de intrusión.

11.2.1. El fichero `/var/log/wtmp`

En este fichero se almacenan las conexiones, mediante `login`, realizadas con éxito. Si intentamos leer este fichero con un editor normal de texto obtendremos basura en la terminal, ello es debido a que su formato es binario y tenemos que utilizar el comando `last`[1] para poder ver los datos de forma legible.

En este archivo se almacenan todas las conexiones realizadas con éxito, además cada vez que se reinicia el sistema se produce una entrada en este fichero. La entrada se produce bajo el usuario `reboot`, el cual no existe en el sistema.

[1] Apartado 11.2.2 en la página 175.

Cada vez que un usuario entra en el sistema se produce una entrada en este fichero, luego el tamaño del fichero va aumentando con el tiempo. Para evitar problemas de espacio si este fichero no se encuentra presente no se produce ninguna entrada. Luego para llevar una contabilidad de los accesos es necesario crear este fichero:

```
[root@localhost root]# touch /var/log/wtmp
```

Normalmente este fichero puede ser leido por todos los usuarios del sistema, si queremos evitarlo tendremos que eliminar los permisos de lectura para los usuarios no privilegiados.

11.2.2. El comando `last`

Este comando permite ver las conexiones realizadas con éxito a nuestra máquina, mediante `login`, ello lo hace consultando el fichero `/var/log/wtmp`. Si ejecutamos el comando obtendremos por la salida estándar todos los `login` realizados con éxito, mostrandos el usuario, la terminal desde la que se conectó, la fecha en la que se conectó, hora de inicio, hora final y tiempo total.

Podemos ver cuando se ha conectado un determinado usuario de la siguiente forma:

```
[root@localhost root]# last bender
bender   tty2         Sun Apr  7 17:29 - 17:30  (00:01)
bender   tty3         Sun Apr  7 09:54 - 10:53  (00:59)
bender   tty3         Mon Apr  1 12:06 - 12:45  (00:38)

wtmp begins Mon Apr  1 11:51:02 2002
[root@localhost root]#
```

También podemos ver los accesos en función de los terminales en lugar de por usuario. Los terminales de texto en **GNU/Linux** son "`tty0`", "`tty1`", ...

```
[root@localhost root]# last 3
root     tty3         Sat Apr  6 10:34 - down   (05:29)
jose     tty3         Thu Apr  4 20:18 - 22:55  (02:37)
jose     tty3         Thu Apr  4 18:23 - 19:12  (00:49)
jose     tty3         Wed Apr  3 15:59 - 16:00  (00:00)
jose     tty3         Tue Apr  2 15:53 - 15:54  (00:01)
root     tty3         Mon Apr  1 17:25 - down   (00:11)
root     tty3         Mon Apr  1 15:05 - down   (00:00)
bender   tty3         Mon Apr  1 12:06 - 12:45  (00:38)

wtmp begins Mon Apr  1 11:51:02 2002
[root@localhost root]#
```

Podemos ver las conexiones al terminal `pts/3`.

```
[root@localhost root]# last pts/3
jose     pts/3        Wed Apr  3 15:44 - down   (00:16)

wtmp begins Mon Apr  1 11:51:02 2002
[root@localhost root]#
```

Para aquellos accesos que se hagan de forma remota también se almacena la IP, y se puede comprobar utilizando los flags "`-a`" y "`-d`".

Si ejecutamos el comando y obtenemos:

```
[root@localhost root]# last
last: /var/log/wtmp: No such file or directory
Perhaps this file was removed by the operator to prevent logging last info.
[root@localhost root]#
```

eso significa que no existe el fichero `/var/log/wtmp`, con lo cual no se lleva contabilidad de los accesos. Si deseamos llevarla bastaría con crear un fichero vacio:

```
[root@localhost root]# touch /var/log/wtmp
[root@localhost root]#
```

y a partir de ese momento se empezarían a registrar los accesos al sistema.

11.2.3. El fichero /var/log/btmp

Este fichero es análogo al fichero `/var/log/wtmp`, sólo que registra los intentos fallidos de conexión. Su formato es binario luego editandolo con un editor normal no veremos nada legible. Para consultar este fichero se utiliza el comando `lastb`[2].

En muchos sistemas este fichero no se crea, luego los intentos fallidos no se registran. Si queremos tener registrados estos intentos debemos crear el fichero:

```
[root@localhost root]# touch /var/log/btmp
```

Normalmente este fichero puede ser leido por todos los usuarios del sistema, si queremos evitarlo tendremos que eliminar los permisos de lectura para los usuarios no privilegiados.

11.2.4. El comando lastb

Este comando permite ver los intentos fallidos de conexión a nuestra máquina, mediante `login`, ello lo hace consultando el fichero `/var/log/btmp`. Si ejecutamos el comando obtendremos por la salida estándar todos los `login` realizados sin éxito, mostrandonos el usuario, la terminal desde la cual se intentó el acceso, máquina desde la cual se intentó, hora de inicio, hora final y tiempo total.

```
[root@localhost root]# lastb
bender   pts/2   localhost   Sun Apr  7 18:29 - 18:29  (00:00)
lila     pts/2   localhost   Sun Apr  7 18:29 - 18:29  (00:00)
root     pts/2   localhost   Sun Apr  7 18:29 - 18:29  (00:00)
icarus   pts/2   localhost   Sun Apr  7 18:28 - 18:28  (00:00)

btmp begins Sun Apr  7 18:28:59 2002
[root@localhost root]#
```

Al igual que con el comando `last` podemos ver los intentos fallidos de conexión a usuarios en concreto, a terminales, ...Si ejecutamos el comando y obtenemos:

```
[root@localhost root]# lastb
last: /var/log/btmp: No such file or directory
Perhaps this file was removed by the operator to prevent logging lastb info.
[root@localhost root]#
```

[2] Apartado 11.2.4 en la página 177.

eso significa que no existe el fichero `/var/log/btmp`, con lo cual no se lleva contabilidad de los accesos fallidos. Si deseamos llevarla bastaría con crear un fichero vacio:

```
[root@localhost root]# touch /var/log/btmp
[root@localhost root]#
```

y a partir de ese momento se empezarían a registrar los accesos fallidos al sistema.

11.2.5. El fichero `/var/log/lastlog`

En este fichero se almacena la última vez que cada usuario accedió al sistema. Este fichero tiene formato binario, luego es necesario acceder a el mediante el uso de alguna herramienta.

11.2.6. El comando `lastlog`

Este comando imprime por la salida estándar la última vez que cada usuario accedió al sistema. Lo hace consultando la información contenida en el fichero `/var/log/lastlog`.

Una salida típica de este comando:

```
Username         Port     From             Latest
root             tty2                      mar abr 16 10:25:54 +0200 2002
nobody                                     **Never logged in**
icarus           pts/3    icarus           mar abr 16 13:40:06 +0200 2002
```

11.3. Permisos SUID, GUID

Los ficheros que tengan permisos SUID[3] y GUID[4] son muy peligrosos ya que se ejecutan con los privilegios del propietario o de su grupo y si alguno de estos tiene privilegios especiales dentro del sistema se puede llegar a comprometer la seguridad del sistema.

Es muy importante el tener localizados y controlados estos ficheros.

[3] Apartado 1.5 en la página 25.
[4] Apartado 1.6 en la página 28.

11.3.1. Peligros con estos permisos

Si logramos ejecutar un comando mediante un fichero que tenga el permiso SUID o SGID activado y pertenezca al `root` entonces estaremos ejecutando ese comando como `root`, imaginar que pasaría si ese comando fuera:

```
rm / -Rf
```

Pues habríamos borrado todo el sistema de ficheros.

Muchos exploits utilizan estos ficheros para acceder y tomar el control del sistema. Supongamos que el usuario `bender` en su casa crea el siguiente programita en C:

```
#include <stdio.h>

#define SIZE 2000

int main (void) {

  FILE *ptFichero;
  char chrBuffer[SIZE];
  int intLeidos;

  /* ABRIMOS EL FICHERO EN SOLO LECTURA*/
  ptFichero=fopen("/etc/shadow","r");

  /* LEEMOS 2000 CARACTERES */
  intLeidos = fread(chrBuffer, sizeof(char), SIZE, ptFichero);

  /* SACAMOS POR LA SALIDA ESTANDAR LOS CARACTERES LEIDOS */
  fwrite(chrBuffer, sizeof(char), intLeidos, stdout);

  /* CERRAMOS EL FICHERO */
  fclose(ptFichero);

  return 0;
}
```

A continuación lo compila y como en su casa es el `root`, entonces hace lo siguiente:

```
[root@localhost root]# chmod 4755 exploit
[root@localhost root]# ls -l exploit
-rwsr-xr-x    1 root     root        14177 abr  4 18:37 exploit
[root@localhost root]# mv exploit /mnt/floppy
[root@localhost root]# ls -l /mnt/floppy
total 27
drwxr-xr-x    2 root     root        12288 abr  4 18:21 lost+found
-rwsr-xr-x    1 root     root        14177 abr  4 18:37 exploit
[root@localhost root]#
```

lo que ha hecho `bender` ha sido activar el permiso SUID del fichero `exploit` y copiarlo a un disquete. Si `bender` se lleva ese disco a un **GNU/Linux** en el que tiene privilegios para montar la disquetera y el sistema de archivos de la disquetera fue montado con permisos de ejecución de SUID, entonces `bender` verá en pantalla el contenido del fichero `/etc/shadow`.

Si en lugar de hacer un ataque pasivo, unicamente curiosear un poquito, decide hacer un ataque activo, como por ejemplo cambiar el password del `root` o cualquier otra cosa podrá hacerlo ya que dicho programa se ejecuta con los privilegios del `root` y si quiere modificar el fichero `/etc/shadow` podrá hacerlo.

11.3.2. Como evitar la ejecución de ficheros con estos permisos

Es posible evitar le ejecución de ficheros con los permisos SUID o SGID. La forma de hacerlo es especificar en el fichero `/etc/fstab` la opcion "**nosuid**" en los sistemas de ficheros en los que no queramos que se puedan ejecutar ficheros con permisos SUID o SGID. Normalmente no se conceden permisos de ejecución SUID o SGID a menos que en las opciones del sistema de ficheros se especifique "**defaults**", lo cual no es muy buena idea. Por ejemplo en el fichero `/etc/fstab` no debería aparecer la opción **defaults** en la disquetera:

```
LABEL=/         /              ext2    defaults         1 1
none            /proc          proc    defaults         0 0
/dev/hda5       swap           swap    defaults         0 0
/dev/hda6       /mnt/vfat      vfat    noauto,user      0 0
/dev/cdrom      /mnt/cdrom     iso9660 noauto,user,ro   0 0
/dev/fd0        /mnt/floppy    auto    defaults         0 0
/dev/md0        /mnt/raid      ext2    auto             0 0
```

una opción más segura para la disquetera sería:

```
/dev/fd0        /mnt/floppy    auto    user,nosuid,exec 0 0
```

En este caso la disquetera la podría montar cualquier usuario, en el sistema de ficheros de la disquetera se podrían ejecutar ficheros binários, pero no se podrían ejecutar aquellos con permisos SUID o SGID.

11.3.3. Como encontrar estos ficheros

Como ya hemos visto es de vital importancia tener localizados y controlados los ficheros con estos permisos. El tener que ir comprobando uno a uno todos los ficheros que puede haber en los sistemas de ficheros que hay montados en una máquina es una tarea de locos. Hay una forma más fácil de hacerlo, utilizando el comando `find`[5].

Recordemos un par de cosas:

SUID los ficheros con este permiso activado tienen permisos "4???".

SGID los ficheros con este permiso activado tienen permisos "2???".

Sticky bit los ficheros con este permiso activado tienen permisos "1???".

[5] Ver apartado 2.4.2 en la página 36.

Supongamos que queremos buscar en todos los sistemas de ficheros montados los ficheros con permisos de ejecución SUID y además queremos listarlos para ver todos sus permisos:

```
[root@localhost root]# find / -perm +4000 -exec ls -l {} \;
```

Expliquemos brevemente los argumentos de `find`:

/ indica a partir de donde se realizará la busqueda.

-perm indica que la busqueda la hará por los permisos de los archivos.

+4000 indica que estamos buscando todos los ficheros que tengan, por lo menos, activado el permiso "`4000`" (SUID). Si hubieramos puesto "`4000`" sin el signo "`+`" unicamente buscaría los ficheros cuyos permisos fueran "`4000`".

-exec ls -l {} indica que ejecute el comando "`ls -l {}`". Cada vez que encuentre un fichero que coincida con los parámetros de la busqueda ejecutará el comando sustituyendo las llaves por la ruta hacía el archivo. Esto tiene un problema y es que cada vez que encuentra un fichero tiene que lanzar un proceso, lo cual es una tarea que sobrecarga bastante la máquina, razón por la cual muchas veces se ejecutan los comandos con el comando `xargs` y un pipe:

```
[root@localhost root]# find / -name core | xargs rm
```

Esto buscaría en todos los sistemas de ficheros montados los ficheros core y los borraría.

11.4. El uso de firmas digitales

Podemos utilizar firmas digitales para asegurar partes del sistema que sean criticas, como por ejemplo los archivos de configuración del sistema.

11.4.1. ¿Que es una firma digital?

No vamos a entrar en detalles de que es una firma digital, ni de todos sus fundamentos, unicamente nos limitaremos a decir que es y para que vale sin entrar en detalles. Si no estas familiarizado con las firmas digitales sería buena idea que te documentaras un poco más sobre Criptografía y sus aplicaciones.

Una firma digital es una técnica criptográfica que nos asegura la procedencia de un documento. Es decir podemos saber "*quien*[6]" nos ha enviado ese documento. Además nos permite verificar que el documento no ha sido modificado por una tercera persona.

11.4.2. ¿Como utilizar una firma digital?

Dentro del sistema existen archivos de configuración que es necesario proteger de cambios no autorizados. Pudiera ser que algún usuario local o no local logrará ganar privilegios dentro del sistema y cambiará la configuración de alguna parte del sistema para favorecer sus intereses. Es en esta parte donde las firmas digitales entran en acción.

Por ejemplo un archivo que puede interesar modificar sería el archivo `/etc/hosts.allow`. Es una buena política de seguridad el que el archivo `/etc/hosts.deny` contenga:

```
ALL: ALL
```

lo cual niega el acceso a todo el mundo a todos nuestros servicios.

[6]He entrecomillado esta palabra ya que para asegurarnos la procedencia necesitamos de la intervención de una tercera parte de confianza, Autoridad de Certificación, pero en el caso que vamos a tratar aquí no será necesaria.

Luego habría que permitir los accesos que nosotros queramos en el fichero `/etc/hosts.allow`. Lo cual puede llegar a ser un poco tedioso. Un atacante estaría interesado en permitir a su máquina el acceso a un determinado puerto, por ejemplo al puerto 22 (login mediante SSH). Entonces tendría que modificar el archivo `/etc/hosts.allow` para permitir el acceso.

Con las firmas digitales no podemos evitar que alguien modifique algo dentro del sistema, pero si nos permiten conocer que algo ha cambiado dentro del sistema.

Si firmamos digitalmente los archivos importantes del sistema como:

- `/etc/hosts.allow`
- `/etc/hosts.deny`
- `/etc/passwd`
- `/etc/shadow`
- ...

podremos comprobar mediante la firma digital si han sido cambiados, y en caso afirmativo actuar en consecuencia.

Es recomendable no almacenar las firmas digitales dentro del sistema, ya que si se ve comprometida la seguridad del sistema un intruso podría llegar a cambiarlas para hacerlas coincidir con las modificaciones que haya hecho.

Una buena opción es imprimirlas en papel o grabarlas en un CD-Rom y ponerlas a buen recaudo. La ventaja que pudiera tener el almacenarlas en el sistema, o en otro ordenador accesible por red, sería el poder comprobar las firmas digitales mediante las herramientas que proporciona el sistema como `at`[7] o `cron`[8], lo cual conlleva sus riesgos y es que dichas herramientas necesitarían conocer nuestras claves, con lo cual si alguien consigue privilegios de `root` podría cambiar todas las firmas digitales para hacerlas coincidir con sus cambios y no nos dariamos cuenta de los cambios.

[7] Apartado 10.3 en la página 157.
[8] Apartado 10.4 en la página 163.

11.5. El demonio syslogd

syslogd permite que cualquier demonio pueda realizar registros en el sistema.

11.5.1. Las "facilidades" de syslogd

Las "*facilidades*" describen "*quien*" origina el mensaje y son:

auth mensajes de seguridad y autenticación. En desuso.

authpriv igual que el anterior.

cron mensajes originados por crond.

daemon mensajes originados por otros demonios del sistema.

kern mensajes originados por el nucleo del sistema.

lpr mensajes originados por el demonio de impresión.

mail mensajes originados por el demonio del correo.

news mensajes originados por el demonio de news.

security igual que "*privauth*". En desuso.

syslog mensajes generados por el demonio syslogd.

user mensajes genéricos de usuario.

uucp mensajes generados por el demonio uucpd.

local0, ...,local7 reservados.

11.5.2. Los "tipos" de `syslogd`

Nos indican los tipos de cada mensaje:

none no envía ningún mensaje.

debug mensajes de depuración.

info mensajes de información.

notice mensajes que necesitan una atención especial.

warning mensajes de aviso.

warn mensajes de aviso. En desuso.

err mensajes de error.

error mensajes de error. En desuso.

crit mensajes críticos, fallo de hardware.

alert mensajes urgente. Algo tiene que ser reparado inmediatamente.

emerg mensajes de emergencia. El sistema no está disponible debido a un fallo grave.

panic mensajes de emergencia. En desuso.

11.5.3. El fichero /etc/syslog.conf

Este fichero contiene la configuración del demonio `syslogd` y le dice que mensajes tiene que almacenar y donde hacerlo.

En cada línea del fichero se especificará como tratar a los mensajes. Lo más normal será indicar una facilidad seguida de un punto y un tipo. Es posible utilizar el asterisco para hacer referencia a todas las facilidades o a todos los tipos.

Veamos algunos ejemplos:

```
*.info;mail.none;authpriv.none;cron.none                /var/log/messages
```

Esto hace que:

- Todos los mensajes del tipo info.
- Ningún mensaje del demonio de correo.
- Ningún mensaje de seguridad o autenticación.
- Ningún mensaje del demonio `cron`.

se almacenen en el fichero `/var/log/messages`. La separación entre las dos columnas se debe hacer mediante tabuladores.

```
authpriv.*                                              /var/log/secure
```

hace que todos los mensajes de autenticación o seguridad se almancenen en el fichero `/var/log/secure`.

```
*.emerg                                                 *
```

hace que cualquier mensaje del tipo "*emerg*" sea notificado con un mensaje *broadcast* a todos los usuarios en la red.

```
uucp,news.crit                                          /var/log/spooler
```

hace que los mensajes del tipo "*crit*" de los demonios `uucp` y `news` se almacenen en el fichero `/var/log/spooler`.

```
mail.*;mail.!=info                                      /var/log/mail
```

hace que todos los mensajes del demonio del correo, exceptuando los del tipo "*info*", se almacenen en el fichero `/var/log/mail`.