

Daniel Ezquerro

Linux Internals

A stylized illustration of Tux, the Linux mascot, a black penguin. The penguin is shown from the chest up, facing forward. Its body is black with a large, irregular white shape on its chest. Inside this white shape are several interlocking grey gears of different sizes, symbolizing the internal workings of the Linux operating system. The penguin's eyes are simple black dots, and it has a small black beak. The background is a solid light blue.

Como funciona Linux

Linux Internals: Como funciona Linux

Daniel Ezquerra

Índice

[Prefacio](#)

[1. Los sistemas Linux](#)

[1.1 Breve historia de los sistemas Linux](#)

[1.2 Una visión general de los sistemas Linux](#)

[1.3 Unix Shell o Terminal, un primer vistazo](#)

[1.3.1 Errores comunes](#)

[2. Ficheros y Jerarquía de directorios](#)

[2.1 Ficheros y directorios](#)

[2.2 Jerarquía de directorios](#)

[2.3 Permisos y propiedades de ficheros](#)

[2.3.1 Umask](#)

[2.3.2 Permisos adicionales](#)

[2.3.4 De vuelta a las propiedades](#)

[2.4 Links simbólicos](#)

[2.4.1 Creando links simbólicos](#)

[3. Dispositivos](#)

[3.1 Bash, salida estándar y otros](#)

[3.1.2 Pipes o Tuberías](#)

[3.2 Extrayendo información de los dispositivos](#)

[3.3 Algunos dispositivos de los sistemas Linux](#)

[3.3.1 EL fichero /dev/null](#)

[3.3.2 El fichero /dev/zero](#)

[3.3.3 Los ficheros /dev/random y /dev/urandom](#)

[3.3.4 Discos duros /dev/sd*](#)

[3.3.5 CD y DVD /dev/sr*](#)

[3.3.6 Terminales /dev/tty* /dev/pts/*](#)

[3.3.7 Puertos Serie /dev/ttyS*](#)

[3.4 Crear ficheros de tipo dispositivo en Linux](#)

[3.5 Udev](#)

[3.5.1 Como funciona Udev](#)

[3.5.2 Las reglas de Udev](#)

[3.5.3 Substitución de cadenas](#)

[4. Discos y sistemas de ficheros](#)

[4.1 Tablas de particiones y particiones](#)

[4.1.1 MBR](#)

[4.1.2 GPT](#)

[4.2 La herramienta dd](#)

[4.3 La partición Swap o espacio de swap](#)

[4.3.1 Usar o no espacio de swap](#)

[4.3.2 Usar ficheros o particiones como espacio de swap](#)

[4.3.2.1 Particiones](#)

[4.3.2.2 Ficheros](#)

[4.4 Creando particiones y una tabla de particiones](#)

[4.5 Sistemas de ficheros](#)

[4.6 Inodos y Ficheros](#)

[4.7 Journaling](#)

[4.8 Creando un sistema de ficheros](#)

[4.8.1 Comprobar un sistema de ficheros](#)

[4.9 Montando un sistema de ficheros](#)

[4.9.1 El fichero /etc/fstab](#)

[5. Bash, profundizando](#)

[5.1 History](#)

[5.2 Variables especiales del terminal](#)

[5.3 Path](#)

[5.4 Autocompletar](#)

[5.5 Algunos comandos básicos](#)

6. Procesos y threads

[6.1 Listando los procesos en nuestro sistema](#)

[6.2 Señales](#)

[6.3 Viendo los procesos en tiempo real](#)

[6.3.1 Interactuando con el comando top](#)

[6.4 Planificación de tareas](#)

[6.4.1 at](#)

[6.4.2 cron](#)

[6.4.2.1 Como funciona cron](#)

7. Recursos y monitorización

[7.1 Visión general de los recursos](#)

[7.1.1 cpu](#)

[7.1.2 Memoria](#)

[7.1.3 Espacio de disco](#)

[7.2 Memoria y fallos de página](#)

[7.3 Monitorización de ficheros](#)

[7.4 Monitorización de entrada/salida](#)

8. Usuarios

[8.1 El fichero /etc/passwd](#)

[8.2 El fichero /etc/shadow](#)

[8.3 Grupos de usuarios](#)

[8.3.1 Trabajando con los grupos](#)

[8.3.1.1 Crear un grupo](#)

[8.3.1.2 Eliminar un grupo](#)

[8.3.1.3 Modificar un grupo](#)

[8.3.1.4 Grupos y contraseñas](#)

[8.4 Creando y borrando usuarios](#)

[8.4.1Borrar usuarios en sistemas Linux](#)

[8.5 Ficheros de inicialización](#)

[8.5.1 El fichero /etc/default/useradd](#)

[8.5.2 El fichero /etc/login.defs](#)

[8.6 Root, sudo y el fichero /etc/sudoers](#)

[8.7 UIDs y cambios de usuario](#)

[8.8 Identificación y Autenticación de usuarios](#)

[8.9 PAM \(Pluggable Authentication Modules\)](#)

[8.9.1 Limitando recursos con PAM](#)

[9. Arranque del sistema](#)

[9.1 Los mensajes de inicio](#)

[9.2 El bootloader](#)

[9.3 Parámetros del kernel](#)

[9.4 Arranques MBR y UEFI](#)

[9.5 GRUB, un primer vistazo](#)

[9.6 El terminal GRUB](#)

[9.7 GRUB, configuración y gestión](#)

[9.8 El sistema de ficheros inicial en RAM](#)

[9.9 El programa init](#)

[9.10 Run levels](#)

[9.11 System V Init](#)

[9.11.1 Cambiar de run level](#)

[9.11.2 Interactuar con los servicios y demonios](#)

[9.12 Systemd](#)

[9.12.1 Systemd y las dependencias](#)

[9.12.2 Configurar Systemd](#)

[9.12.3 Systemctl y journalctl](#)

[9.12.3.1 journalctl](#)

[9.12.4 La polémica tras systemd](#)

[9.13 Upstart](#)

[9.13.1 Configuración de Upstart](#)

[9.13.2 initctl](#)

[9.14 Apagado del sistema](#)

[10. Red](#)

[10.1 Breve introducción a las redes de computadores](#)

[10.2 Un símil de las comunicaciones entre máquinas](#)

[10.3 La capa de red o acceso a la red](#)

[10.4 Direcciones IP](#)

[10.4.1 Subredes](#)

[10.4.2 Convertir una netmask a notación CIDR](#)

[10.4.3 Subredes Privadas](#)

[10.4.4 Cambiando o asignando una IP a una interfaz](#)

[10.4.5 Resolución de nombres de Dominio \(DNS\)](#)

[10.4.5.1 Como funciona la resolución de nombres de dominio](#)

[10.4.5.2 El fichero /etc/nsswitch.conf](#)

[10.4.5.3 El fichero /etc/hosts](#)

[10.4.5.4 El fichero /etc/resolv.conf](#)

[10.5 Rutas y tablas de rutas](#)

[10.5.1 Ruta por defecto](#)

[10.6 Configurar un sistema Linux como router](#)

[10.7 DHCP](#)

[10.8 La capa de transporte](#)

[10.8.1 TCP](#)

[10.8.2 Puertos](#)

[10.9 Ethernet inalámbrico](#)

[10.9.1 La herramienta iw](#)

[Epílogo](#)

[Bibliografía](#)

Prefacio

Hoy en día somos muchos los que trabajamos con ordenadores y no parece que esta tendencia vaya a ir a la baja. Es cierto que muchos de los que trabajan con ordenadores lo hacen en su entorno familiar o con programas de ofimática, sin embargo, hay también un gran número de persona que necesitan sacar mayor provecho de las máquinas.

Hay gente que se dedica a hacer que los ordenadores funcionen 24/7 (24 horas al día, 7 días a la semana) o algunos que se dedican a hacer programas para terceros. En general, hay usuarios para todos los gustos.

Muchas veces se recurre a programas ya hechos o pequeñas utilidades (scripts) que nos solucionan la vida y que, para que negarlo, son de lo más útil. Sea cual sea el caso, conocer como funciona aquello con lo que trabajamos nos puede hacer más sencillo el trabajo, e incluso a veces nos puede permitir hacerlo. Siempre, claro está, dentro de las posibilidades.

Ese es el objetivo de Linux Internals: dar unas nociones de como funcionan las cosas en los sistemas operativos Linux. Y con esta última frase, me gustaría hacer referencia a como funciona el sistema operativo además de como funcionamos nosotros con él. En ocasiones los detalles cambian entre distintas distribuciones de Linux, pero en cualquier caso, los conceptos son los mismos incluso entre distribuciones.

La idea de este texto es dar una visión de como trabajan los sistemas Linux más allá de los comandos y las instrucciones y sobre

como podemos modificar el comportamiento del sistema en las ocasiones en las que nos es conveniente.

1. Los sistemas Linux

Antes de ver como funciona un sistema Linux deberemos entender que es, de que habla la gente cuando habla de Linux. Debemos entender por que a veces es Linux y a veces Ubuntu u otro sinfín de preguntas que seguro que surgen cada vez que vemos uno de los sistemas operativos del pingüino. Una breve historia y una visión global deberán ser suficiente para entender donde nos estamos metiendo y poder comprender como funciona un sistema operativo Linux.

1.1 Breve historia de los sistemas Linux

Un poco de historia sobre Linux nos ayudara a ponernos en contexto. Si la historia de este sistema operativo no os interesa podéis pasar directamente a la siguiente sección.

A finales de 1960, los laboratorios Bell junto con AT&T y otros trabajaban con un sistema llamado Multics (Multiplexed Information and Computer Service). Este sistema estaba diseñado de forma modular de tal manera que era posible desactivar algunos de los módulos sin que los otros se viesan afectados por ello. Esta característica se mantiene todavía en los sistemas actuales. De la misma manera, Multics era un sistema para mantener información con diferentes niveles de confidencialidad en la misma máquina, y por tanto con distintos niveles de permisos. Característica que también se mantiene hoy en día.

A partir de Multics, uno de los trabajadores de la AT&T, Ken Thompson, diseñó un juego llamado aventura espacial. Este juego era muy costoso (monetariamente de jugar). Esto lleva a Ken Thomson, junto con Dennis Ritchie a desarrollar, al principio sin ningún apoyo económico, el sistema operativo UNIX.

Unix se convirtió rápidamente en el sueño de los desarrolladores puesto que estaba compuesto por pequeños programas con funciones muy concretas que se juntaban para realizar tareas más complejas. Al incorporar a un sistema multitarea y con un sistema de ficheros potente como UNIX soporte para el procesamiento de textos, este proyecto empezó a recibir apoyo económico por parte de los Bell Labs. Es a partir de entonces que empieza la verdadera evolución de Unix.

Al principio de los años 70 se opta por reescribir el código de UNIX en el lenguaje C. Esto permite que el sistema se convierta en portable, por la gran facilidad con la que se puede modificar el código para llevarlo a otras máquinas.

Veinte años más tarde, en 1991, Linus Torvalds en Helsinki empezó a desarrollar un sistema operativo que sería compatible con UNIX y que trabajaría sobre la base de este. El sistema estaba pensado para cualquier ordenador con una arquitectura i386 y empezó, según palabras del propio Linus Torvalds: "Como un hobby que realizaba en su tiempo libre y que nunca llegaría a ser portable o tan grande como el proyecto GNU".

El código de Linux estaba programado en C, hoy en día todavía lo está, y se distribuyó bajo una GNU Public License desarrollada años atrás por Richard Stallman. El compilador GNU C Compiler (gcc) fue la opción de Torvalds para compilar el sistema

operativo Linux en sus inicios (entonces llamado Freax en lugar de Linux, nombre que adoptaría más tarde) y sigue siendo hoy en día la opción principal para los sistemas Linux.

Internet y la licencia GNU ayudó a que el sistema operativo tuviese una gran aceptación como un “clon de código libre de UNIX”, que estuvo durante muchos años enfangado en pleitos legales sobre derechos y propiedad.

A partir de la base diseñada por Torvalds, conocida como el Kernel, aparecen diversas distribuciones que no son más que una agrupación de programas, utilidades y ficheros preparados para su instalación. Aparecen también gran cantidad de desarrolladores que empiezan a colaborar en mejorar el kernel de Linux.

Así pues, cuando hablemos de sistemas Linux nos estaremos refiriendo a todos aquellos sistemas operativos que usan como kernel el sistema creado por Torvalds. Siempre que hablemos de Unix nos referiremos a un conjunto todavía mayor de sistemas operativos que han evolucionado desde este. Nos referimos en este caso a Linux, MAC OS, BSD,...

1.2 Una visión general de los sistemas Linux

Antes de entrar en detalles sobre como funcionan las cosas en Linux y como trabajar con él, tenemos que ser capaces de tener una visión general de como funciona y de que estamos hablando al mencionar un sistema Linux.

Los sistemas operativos, como es el caso que nos ocupa, tienen multitud de tareas de las que encargarse y, por tanto, multitud de

partes que los componen y que realizan unas u otras tareas. Intentar aprenderlo todo resulta abrumador, y muchas veces innecesario, puesto que el SO realizará tareas de las que no debemos preocuparnos.

En este punto, puede ser válido el símil del televisor. Al usar un televisor con un mando distancia que permite cambiar de canal, no conocemos realmente todos los detalles que hacen que al apretar un botón el canal del televisor cambie. Sin embargo, si que somos capaces de separar los problemas, cuando los hay, en problemas del mando o problemas del televisor. Es decir, estamos creando dos niveles de abstracción sobre los detalles técnicos, el nivel del mando a distancia y el del televisor.

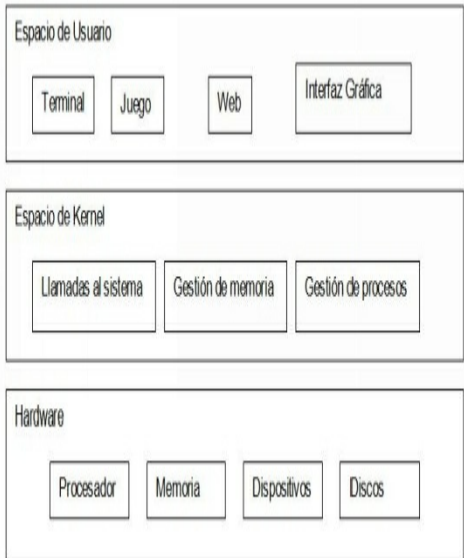
Con los sistemas operativos haremos algo similar. Como es una tarea ardua y casi imposible llegar a comprender todos los detalles que conforman el sistema operativo, intentaremos abordarlo usando determinados niveles de abstracción que nos permitirán tener una visión global. Más tarde seremos capaces de profundizar en aquellos detalles que más nos interesen.

En el caso de los sistemas Linux, que son los que nos ocupan, podemos definir tres niveles de abstracción:

- Hardware
- Kernel
- Usuario

En esta clasificación que presentamos, cada capa es un conjunto de funciones agrupadas según se sitúen más cerca del hardware o del usuario. Así por ejemplo, los juegos, servidores web, etc. se situarían en la capa superior: la Capa de Usuario. Aquello que corresponde a los 0's y 1's se situaría en la capa más baja, es decir en

la de Hardware. El espacio de Kernel, la capa intermedia, comprendería, más o menos, todo lo que corresponde al sistema operativo.



El trabajo de Kernel o del espacio de kernel, consiste en gestionar los distintos procesos que se ejecutarán en nuestra máquina, la memoria de que disponemos o de gestionar los *drivers* de los

distintos dispositivos. Cosas, todas ellas, que veremos más adelante.

Anivel de memoria, el Kernel se encarga de “particionar” la memoria para los distintos procesos, crear áreas de memoria compartida, gestionar los accesos a memoria, etc. Anivel de procesos el Kernel realiza también un papel muy importante puesto que es el encargado de arrancar y parar procesos, conocer el estado en el que se encuentran, darle tiempo de computo a cada proceso y una multitud más de tareas. Todos los sistemas operativos de una u otra manera deben realizar estas tareas. Si queréis saber más acerca de como trabaja un sistema operativo con los distintos procesos podéis buscar información sobre el planificador del sistema.

Aclaración

Más adelante dedicaremos tiempo a comprender los procesos y como funcionan. Por ahora, para comprender que es lo que hace el Kernel de Linux, nos basta con saber que, a grandes rasgos, un proceso es un programa en ejecución.

El Kernel cuenta, además, con una funcionalidad muy importante: las system call o llamadas al sistema. Las llamadas al sistema son operaciones específicas que los programas que ejecuta un usuario son incapaces de realizar, por ejemplo abrir un fichero, reservar memoria, etc.

Para realizar estas operaciones, de las que se encarga el sistema operativo, los programas usan las llamadas al sistema. Para aquellos que alguna vez han programado, las llamadas al sistema son funciones que suelen estar en las librerías estándar del lenguaje de programación usado y que tienen una equivalencia directa con una

llamada hexadecimal a una instrucción conocida por el kernel.

Además del kernel, como hemos dicho, los sistemas Linux cuentan con un espacio de usuario. En el espacio de usuario es donde se ejecutan la mayoría de tareas. Hay que pensar que aunque para los usuarios cada programa realiza tareas distintas, para el kernel cualquier proceso es básicamente lo mismo. El conjunto de instrucciones o de operaciones que un programa puede realizar en el espacio de usuario es limitado. Por este motivo, cuando necesita realizar operaciones que solo se pueden ejecutar a nivel de kernel, realizará una llamada al sistema. La llamada al sistema se encargará de cambiar el contexto en el que trabaja el programa para poder realizar la tarea en el espacio de kernel y poder devolver más tarde el resultado al espacio de usuario. Es decir, cuando un programa necesita realizar una tarea que únicamente puede realizar el kernel, le solicita a este que ejecute la tarea y le devuelva el resultado cuando termine. La petición al kernel se conoce con el nombre de llamada al sistema.

Para acabar con esta visión rápida a los sistemas Linux, faltaría añadir un último componente: los usuarios. Aunque más adelante veremos en profundidad que suponen los usuarios para un sistema Linux y como trabajar con ellos, es importante tener una pincelada general de este concepto para entender a grandes rasgos los sistemas Linux.

Un usuario en un sistema Unix no es la persona que se sienta delante del terminal a aporrear las teclas, en los sistemas Unix un usuario es cualquier entidad capaz de ejecutar procesos y poseer ficheros. Más adelante, entraremos también a discutir el concepto de fichero. Así pues, cualquier sistema Linux deberá contar siempre, al

menos con un usuario. Vemos también que, efectivamente, todos los sistemas Unix son sistemas multiusuario y multitarea.

1.3 Unix Shell o Terminal, un primer vistazo

El shell de Unix, shell o terminal, es un intérprete de comandos que nos permite interactuar con el sistema a través de una serie de órdenes sencillas. Existen cantidad de shells o terminales distintos y podemos instalar uno u otro según nos convenga (bash, sh,...) Sea como sea, cualquier distribución de Linux que instalemos pondrá a nuestra disposición una shell (línea de comandos) para que podamos interactuar con el sistema.

Es habitual en los sistemas actuales que los SO cuenten con diversos terminales que arrancan junto con el sistema. Si tenemos una distribución de Linux que posee una interfaz gráfica podremos abrir una shell desde dicha interfaz. En otro caso, el SO arrancará directamente en un terminal.

De hecho al arrancar, el SO carga diversos terminales y en uno de ellos ejecuta la interfaz gráfica mientras que mantiene los otros en modo texto. Habitualmente podremos cambiar entre los diversos terminales con las combinaciones de teclas:

Alt+f1

Alt+f2

...

Alt+f6

Es decir, la tecla Alt seguida de la tecla 'f' con el número correspondiente al terminal al que queremos ir. Normalmente no

necesitaremos cambiar el terminal, pero en algunas ocasiones puede llegar a ser útil. Por ejemplo, en el proceso de instalación de una distribución Debian, donde la instalación se hace a través de una interfaz de fondo azul, el terminal 4 suele dar información escrita de todos los pasos que está realizando el instalador así como de los errores que puedan surgir.

Según lo visto hasta ahora, siempre que queramos ejecutar un comando deberemos recurrir a una shell o a un terminal.

Siempre que nos encontremos en una shell o terminal veremos algo como lo siguiente:

```
root@vps18547:/etc#>
```

El nombre delante de la '@' nos indica con que usuario hemos entrado al sistema. En el capítulo 8. Usuarios veremos más sobre esto. Lo que encontramos después de la '@' es el hostname o nombre del PC. Finalmente, después de los ':' encontramos el path actual del directorio en el que trabajamos. Veremos más sobre esto en 'Ficheros y Directorios'. Esta línea la veremos después de ejecutar cada comando y nada más abrir un terminal y debería leerse de una forma similar a:

```
root at hostname working on /etc #Si leemos en inglés
```

```
root en hostname trabajando en /etc #Si leemos en español
```

Ahora ya podemos introducir comandos y esperar que la consola nos dé un resultado o un error. En caso de que cualquier comando nos dé un error, en los sistemas Linux resulta más o menos

sencillo entender cual ha sido. A diferencia de otros sistemas operativos, no obtendremos un número incomprensible o un texto larguísimo, en el caso de los sistemas Linux todos los errores suelen seguir la misma estructura.

```
root@vps18547:~#> cd /asd
```

```
-bash: cd: /asd: No existe el fichero o directorio
```

En este caso lo primero que nos muestra es el programa que ha dado el error: `cd`.

A continuación, tenemos una línea que nos muestra la ruta (path en inglés), el lugar donde se ha producido el error. Esto puede ser un directorio como en este caso, puede ser el número de línea de un fichero o cualquier lugar donde se haya localizado un error. Sea como sea, se trata de una información muy útil para ayudar a localizar el error.

Finalmente, obtenemos un mensaje informándonos de cual ha sido el error. En este caso que no existe el fichero o directorio. Esto podría significar que realmente no existe en el sistema o bien que no existe allí donde lo estamos buscando.

Hemos puesto como ejemplo un comando sencillo (veremos su uso más adelante) pero cualquier error de un programa o comando, seguirá, como norma general, la misma estructura.

1.3.1 Errores comunes

Aunque la formulación de los errores es siempre bastante similar en los sistemas Linux vamos a presentar algunos de los errores más comunes para empezarnos a familiar con los tipos de error que podemos encontrar y estar prevenidos.

- **No existe el fichero o directorio.** Este ya lo hemos visto, nos indica que el fichero o directorio con el que estamos intentando trabajar no existe o no se puede encontrar en la ruta especificada. Listar los ficheros de la ruta inmediatamente superior nos puede dar una idea bastante acertada del problema.
- **El fichero ya existe.** Este error suele aparecer cuando intentamos crear un fichero ya existente o cuando intentamos crear un directorio usando como nombre el de un fichero que ya existe. En los Sistemas Operativos (SO) Linux, como en tantos otros, pueden existir ficheros con el mismo nombre siempre que no se encuentren en la misma ruta.
- **No es un directorio/Es un directorio.** Un error muy común, aparece cuando tratamos un fichero como si fuese un directorio o a la inversa.
- **Permiso denegado.** Un error muy común también. Suele estar relacionado con problemas de permisos. En la sección 2.3 Permisos y propiedades de ficheros veremos más detalles. También aparece cuando tratamos de hacer tareas que únicamente el súper usuario puede hacer. Más adelante hablaremos del súper usuario cuando tratemos con los usuarios.
- **No hay espacio en el dispositivo.** Un error claro y que suele incomodar bastante. No queda espacio para almacenar más datos. Tenemos dos opciones, empezar a borrar ficheros o comprar más disco. Eso cuando se trata de disco duro. Si se

trata de un pendrive, un cd, etc quizás no haya muchas opciones.

2. Ficheros y Jerarquía de directorios

Una de las cosas con las que más vamos a trabajar en los sistemas Linux es con los ficheros, bien sea porque tenemos que configurar un servicio, crear un nuevo usuario, trabajar con los usuarios que ya existen o simplemente configurar una nueva red. Al final los ficheros conforman uno de los elementos más importante de los sistemas Linux, es por tanto un buen punto de partida para empezar a comprender como funcionan estos sistemas.

2.1 Ficheros y directorios

Antes de comprender como se organizan los ficheros o directorios en los sistemas Linux deberemos tratar de comprender que son.

Un directorio, es el equivalente a las carpetas de Windows, aunque nunca utilizaremos ese nombre puesto que en sistemas Linux no tiene sentido esta denominación. Un directorio es el lugar donde se guardan los ficheros. En última instancia, un directorio es también un fichero con unas características concretas. Es decir, un directorio es un fichero que organiza otros ficheros. De hecho, podemos decir que en un sistema Linux todo son ficheros.

Un fichero es la estructura que usa el sistema operativo para almacenar información. Como decíamos, un directorio no deja de ser un fichero que almacena la información de donde se pueden encontrar los ficheros que contiene. Los sistemas Linux reconocen 3 tipos distintos de ficheros:

- **Ficheros ordinarios:** permiten almacenar información en un dispositivo físico. Los documentos de texto, binarios ejecutables, páginas web, etc... se incluyen en esta categoría. Estos ficheros deben tener un nombre de hasta 256 caracteres (esto puede variar según el kernel y el SO) y pueden incluir cualquier carácter en el nombre a excepción de '/'.

Advertencia

Las letras minúsculas y mayúsculas se consideran distintas, es decir, los sistemas Linux son case sensitive. Mucho cuidado también con los nombres de los ficheros puesto que aquellos que empiezan por el carácter '.' son considerados ficheros ocultos por el sistema.

- **Directorios:** Son ficheros especiales que agrupan otros ficheros de forma estructurada. Por supuesto, pueden contener otros directorios.
- **Ficheros especiales:** Son quizás de los ficheros más importantes de Linux, representan todos los dispositivos conectados al ordenador: impresoras, discos duros, discos usb, etc.

Hemos visto que los ficheros son la manera como trata Linux la información que necesita. Volveremos a profundizar en el tema más adelante, sin embargo, antes de seguir vamos a ver algunos comandos que nos serán útiles para trabajar con lo que ya sabemos. En general los comandos de Linux son muy fáciles de recordar si sabemos algo de inglés y pensamos en lo que queremos hacer.

```
root@vps18547:~#> pwd  
/home/internal
```

Este comando nos indica en que directorio estamos. (**print working directory**, escribe el directorio en el que trabajas en inglés).

```
ls
```

(list --> listar en inglés) nos devuelve una lista de los ficheros que se encuentran en el directorio actual. Este comando admite varios parámetros que ofrecen más información. Permite también especificar la ruta del directorio cuyo contenido queremos listar.

```
cd
```

(change directory, cambiar directorio en inglés) Es bastante explicativo, permite cambiar a un directorio especificado a continuación del comando. Hay que tener en cuenta que en los sistemas Unix, en Linux por tanto también, el '.' representa el directorio actual mientras que '..' representa el directorio inmediatamente superior o directorio padre. Luego veremos más acerca de estos detalles.

```
cat
```

(de concat, concatenar en inglés) Es un programa que permite mostrar ficheros o concatenarlos. En general los estamos concatenando siempre puesto que lo que hace este comando es concatenar el contenido de los ficheros a la salida estándar. Veremos más sobre esto llegado el momento.

Juntando conceptos: Si ejecutamos el comando ls en un directorio en el que tenemos un fichero llamado holamundo.txt, ¿Cuántos

ficheros nos listará ls -a? La a es de all, todo en inglés.

Nos listará 3 ficheros:

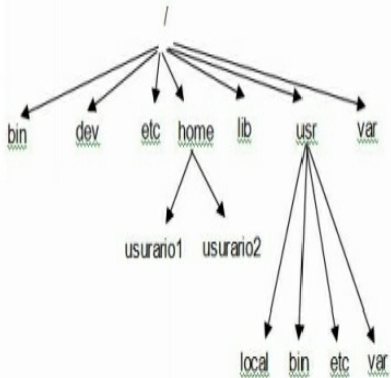
holamundo.txt # nuestro fichero de texto

. # El directorio actual

.. # El directorio padre o inmediatamente superior

2.2 Jerarquía de directorios

En los SO modernos, los directorios y ficheros se organizan de forma jerárquica. En los sistemas Unix no es diferente, la estructura jerárquica de Unix se presenta (más o menos) en la imagen a continuación.



El directorio `/` se conoce como el directorio raíz del sistema. De él cuelgan todos los demás directorios. Esto tiene un par de implicaciones importantes. Por un lado, nos está diciendo que jamás tendremos un directorio al mismo nivel que `/`. Es decir en el directorio `/` no tendremos directorio superior, por tanto al hacer un `ls` no se listará `..`.

La segunda de las implicaciones que presenta es que al expresar el path (camino o ruta de un directorio), si este empieza por `'/'` estaremos especificando una ruta absoluta al sistema, mientras que si el directorio no empieza por `'/'` estaremos especificando una ruta relativa al lugar donde nos encontramos.

Los directorios de primer nivel, inmediatamente debajo de `/`,

más relevantes y normalmente comunes a todos los sistemas Unix son los siguientes:

- **/bin** Contiene programas listos para ejecutarse o ejecutables. Todos los programas básicos como ls o cp se encuentran en este directorio. Los programas en este directorio pueden invocarse desde cualquier directorio en el sistema sin necesidad de especificar la ruta.
- **/dev** Contiene los ficheros de los distintos dispositivos. Ya hemos visto en la sección 2.1 Ficheros y directorios la existencia de estos ficheros. Además veremos más sobre dispositivos en el capítulo 3. Dispositivos.
- **/etc** Contiene los ficheros de configuración del sistema. Algunos de los ficheros que encontramos aquí son relativos al hardware de la máquina. En general están representadas la mayoría de configuraciones de nuestro sistema, desde las contraseñas de los usuarios, los ficheros de configuración de servicios y software, configuración de red e incluso de la tarjeta gráfica.
- **/home** Contiene los directorios personales de los usuarios estándar. Este directorio no existe en todas las variantes de los sistemas Unix, pero suele estar presente en la mayoría.
- **/lib** Contiene las librerías que pueden ser usadas por el resto de los programas presentes en el sistema. Habitualmente existen dos tipos de librería, estáticas o dinámicas. Esencialmente son lo mismo. A grandes rasgos, la diferencia está en como las cargan y usan los programas. En este directorio se encuentran únicamente las librerías dinámicas. No

siempre es así, pero se cumple en la mayoría de los casos. En otros directorios como `/usr/lib` si que es más usual encontrar ambos tipos de librería.

- **/proc** Suele ser el directorio más distinto entre las distribuciones, aunque su existencia es común. Provee estadísticas sobre el sistema, tanto software como hardware. Todo a partir de un conjunto de ficheros que se pueden acceder de las maneras que ya se han visto.
- **/sys** Sigue la misma filosofía que el directorio anterior, con la diferencia que muestra información sobre los distintos dispositivos.
- **/sbin** Ofrece la misma funcionalidad que `/bin`. La diferencia radica en que contiene ejecutables destinados a la gestión o mantenimiento del sistema, por lo que los usuarios regulares no pueden ejecutar comandos de este directorio en ninguna ruta dentro del sistema, ni tan siquiera en `/sbin`
- **/tmp** Suele ser un pequeño espacio para los ficheros temporales de tamaño reducido de los que no nos preocupamos mucho, ni nosotros como usuario ni el sistema. Muchas distribuciones eliminan el contenido de `/tmp` en el proceso de boot o arranque.
- **/var** El directorio donde se guarda la información temporal de los sistemas. Los programas guardan aquí sus logs (registros de actividad) y otra información cambiante que necesitan para su correcto funcionamiento como información sobre identificadores de procesos, caché ...

Advertencia

Aunque `/var` tiene un subdirectorio `tmp` (`/var/tmp`) el SO no vacía este directorio al arrancar como sucede con `/tmp`

- **/usr** Pronunciado “user”, usuario en inglés. Parece que debiese contener los archivos de los usuarios, sin embargo nada más lejos de la realidad. Los ficheros de los usuarios se encuentran habitualmente en `/home/usuario`. Este directorio suele contener una gran cantidad de directorios en su interior organizados de forma jerárquica. Es quizás uno de los directorios más importante puesto que dentro de `/usr` encontramos las librerías, manuales, programas, etc que contiene nuestro SO. La separación entre `/usr` y `/` se debe a motivos históricos sobre la distribución de espacio para la raíz del sistema. Hoy suele ser habitual encontrar archivos que pertenecen de forma específica a la distribución en este directorio.
- **/boot** Contiene los ficheros para el arranque del Kernel. Los sistemas actuales ya no permiten la inclusión en este directorio de nuestros propios ficheros. Si queremos que algún programa se ejecute al inicio tenemos otras posibilidades que veremos más adelante.
- **/media** Este directorio es el punto de entrada para todos los dispositivos “removibles” como pendrives o similares.

Juntando conceptos:

Recordemos 2.1 *Ficheros y directorios* y supongamos que al ejecutar el comando `pwd` nos devuelve lo siguiente:

```
/home/usuario/software
```

que pasará ahora si ejecutamos los siguientes comandos?

```
cd /bin
```

```
cd bin
```

Si volvemos a ejecutar `pwd` después de los comandos anteriores obtendremos:

En el primero de los casos: `/bin`

En el segundo de los casos: `/home/usuario/software/bin`

¿Que ha pasado? En el primer caso hemos especificado un directorio con el carácter `'/'` delante, es decir absoluto o relativo a la raíz del sistema. En el segundo caso hemos especificado una ruta relativa al directorio en el que trabajábamos.

2.3 Permisos y propiedades de ficheros

Encontrar el lugar para hablar de permisos en los sistemas Linux es complicado. Es difícil saber si para hablar de permisos hay que hablar antes de usuarios y grupos o si con conocer como se gestionan los ficheros es suficiente. En este caso vamos a tratar los permisos junto con los ficheros. Basta con saber, que como hemos mencionado en algún momento anterior los sistemas Linux son

sistemas multiusuario. Esto significa que un mismo SO puede tener diversos usuarios. No solo eso si no que además los usuarios de los sistemas Linux se dividen en grupos.

La seguridad de los documentos en Linux sigue el sistema militar. Cada documento tiene una “clasificación” de seguridad que se plasma en 3 tipos de permisos: aquellos que afectan al usuario que es propietario del fichero, aquellos que afectan al grupo que es propietario del fichero y aquellos que afectan al mundo entero, a todos los que no cumplan los requisitos anteriores.

Así pues, lo primero que deducimos hasta ahora es que todos los ficheros en los sistemas UNIX y por tanto Linux, son poseídos por un usuario y un grupo existentes en el sistema.

Podremos cambiar estos parámetros para que se ajusten a las necesidades.

```
chown usuario1.grupo1 <fichero>
```

Cambiará los propietarios de <fichero> al usuario *usuario1* y al grupo *grupo1*. Tanto el usuario como el grupo deben ser conocidos por el sistema., en caso de que sean desconocidos obtendremos un error. Si queremos aplicar el mismo comando a todos los ficheros dentro de un directorio modificaremos el comando chown(change owner en inglés).

```
chown -R user1.group1 <fichero>
```

Sabemos cambiar los propietarios de los ficheros pero, ¿como podemos saber el propietario actual de un fichero?. Sencillo, recurriremos a un comando que ya hemos visto anteriormente.

```
ls -l
```

drwxr-xr-x 2 rootroot 4096 jul 31 04:34 backups

Lo que vemos en gris claro, el primer root, muestra a que usuario pertenece el fichero. Lo que vemos en gris oscuro, el segundo root, muestra a que grupo pertenece. Si seguimos observando, vemos que la línea que nos está mostrando el comando `ls` empieza con una ristra de letras. Esas letras son los permisos. Vamos a ver que significan:

- **d**: Nos indica que backups es un directorio.
- **rwx**: Los permisos del propietario del fichero.
- **r-x**: Los permisos para el grupo que posee el fichero.
- **r-x**: Los permisos para todos los demás.

Los permisos son grupos de 3 bits que se representan de la siguiente manera:

- **r**: Permiso de lectura.
- **w**: Permiso de escritura.
- **x**: Permiso de ejecución.

Si en la posición de la 'w' encontramos un `-` significa que el grupo o quien corresponda no tiene permiso de escritura, lo mismo sucede con las otras posiciones. Para modificar los permisos disponemos también de un comando.

`chmod g+w backups`

Dará permisos de escritura al grupo que posee el fichero backups. Así según nos convenga podremos asignar permisos con

cualquiera de las siguientes combinaciones.

```
chmod [u|g|o] [+|-|=] [r|w|x] <fichero>
```

#Da los permisos rw para el grupo al fichero independientemente de los que tuviese antes

```
chmod g=rw <fichero>
```

#Añade permisos de ejecución al fichero para los otros (ni el grupo ni el propietario) a los permisos que ya tuviesen otros.

```
chmod o+x <fichero>
```

#Quita permisos de lectura al propietario y al grupo propietario del fichero. Mantiene los otros permisos que tuviesen.

```
chmod ug-w <fichero>
```

Existen, también, otras maneras de expresar los permisos. Estos se pueden expresar numéricamente. Puede parecer más complicado, pero acaba siendo más cómodo a largo plazo acostumbrarse a cambiar los permisos numéricamente. Vamos a ver como: Hemos comentado que los permisos son 3 grupos de 3 bits. Esto representado en binario equivale a:

000 000 000

El primer grupo de 0's representa los permisos del propietario, el segundo grupo representa los permisos del grupo, finalmente el tercero representa los permisos de todos. Estos bits solo pueden tomar los valores 0 o 1 (estamos trabajando con base binaria), el 1 significa que el permiso está activado el 0 representa un permiso

desactivado. Así pues

111 101 101

Significa que el propietario tiene permisos de lectura, escritura y ejecución (los bits van en este orden) mientras que el grupo y el resto de los usuarios tienen permisos de lectura y ejecución pero no de escritura. ¿Como usamos entonces estos valores para cambiar los permisos? Muy sencillo, lo primero que vamos hacer es convertir cada grupo a la notación decimal:

7 5 5

chmod 755 backups

A continuación recurrimos al comando chmod que dará los permisos que hemos descrito según la notación binaria al fichero backups. Cualquier combinación que nos permitan estos 3 bits será válida.

Advertencia

El mayor valor que se puede representar con 3 bits en notación binaria es el 7 en notación decimal. Eso significa que en ningún caso se podrán asignar permisos mayores a 7.

Decimal	Binario	Lectura (r)	Escritura (w)	Ejecución(x)
0	000	No	No	No
1	001	No	No	Sí

2	010	No	Sí	No
3	011	No	Sí	Sí
4	100	Sí	No	No
5	101	Sí	No	Sí
6	110	Sí	Sí	No
7	111	Sí	Sí	Sí

Advertencia

Para poder listar el contenido de un directorio es necesario tener permisos de ejecución para dicho directorio.

2.3.1 Umask

¿Que es lo que sucede con los permisos cuando se crea un nuevo fichero? ¿Como podemos controlar con que permisos se va a crear? Para ello están las umask o mascarar de usuario. Una mascara de usuario hace exactamente eso, define con que permisos se va a crear un fichero o directorio. Así pues si ejecutamos el comando umask

```
umask
```

```
022
```

Nos devolverá la máscara actual, algo similar a 022. Los número en el caso de umask funcionan exactamente igual que a nivel de permisos. Pero al crear un fichero vemos que realmente no se está creando con los permisos 022. ¿Que estas sucediendo?

La máscara de usuario establece valores negativos sobre los permisos originales. En la mayoría de sistemas UNIX los ficheros

regulares (normales) se crean con los permisos 666 mientras que los directorios se crean con los permisos 777. A esto hay que aplicar-le la máscara. Esta máscara consiste en aplicar una operación NAND. Veamos como funciona:

umask:	000 010 010
Permisos originales:	111 111 111
Permisos finales:	111 101 101

La operación NAND hace lo siguiente: allí donde la máscara tiene un 1, si los permisos originales tienen también un 1 el valor de la máscara original se pone a 0, en caso contrario la máscara original se deja tal y como está.

Pese a todo lo explicado, todavía tenemos un escollo. El comando umask únicamente nos permite cambiar la máscara de usuario hasta que se cierra la sesión. ¿Que debemos hacer entonces para que el cambio sea permanente? Si nos remitimos a la sección 2.2 Jerarquía de Directorios, veremos que en el directorio /etc se encuentran los ficheros de configuración. Como lo que queremos hacer es modificar un parámetro de la configuración parece claro que tendremos que recurrir a ese directorio. En este caso necesitaremos editar el fichero

/etc/profile

ó

/etc/bash.bashrc

para añadir la modificación de umask. La añadiremos como si escribiésemos el comando correspondiente en un terminal. Estos

ficheros a modificar pueden ser distintos dependiendo de la distribución que tengamos instalada. En cualquier caso, la jerarquía de directorios es común para todos los sistemas Linux y por tanto siempre deberemos ir a `/etc` a buscar esta configuración.

Importante

Es posible encontrar un fichero `.bashrc` en el *home* de cada usuario, esto es debido a que ese fichero contiene únicamente modificaciones que afectan a ese usuario. En caso de que las modificaciones sean comunes a todos los usuarios jamás encontraremos el fichero de configuración en el *home* de dicho usuario.

2.3.2 Permisos adicionales

Aunque con lo que hemos visto hasta ahora tenemos cubierto todo lo básico acerca de los permisos, hay situaciones que aún no se han considerado. Por ejemplo, supongamos que tenemos un directorio en que el usuario *user1* es el propietario y por tanto goza de todos los permisos. El grupo por contra solo tiene permisos de escritura. ¿Que sucedería si un miembro del grupo crea un nuevo fichero en el directorio?

El nuevo fichero sería propiedad del usuario que lo ha creado, por tanto el usuario *user1* que debía tener todos los permisos sobre los documentos del directorio, sólo tendrá permisos de escritura (pues pertenece al mismo grupo que el usuario que creo el fichero). Mientras tanto el usuario que creo el fichero podrá hacer lo que le venga en gana con él. Pero no es ese el comportamiento que deseábamos. Linux pone a nuestra disposición una serie de bits que

nos permiten arreglar esta situación.

- **SUID:** cuando un fichero con este bit activado se ejecuta asumirá el identificador de usuario del propietario del fichero. Esto significa que si un fichero es propiedad del súper usuario (root) y tiene el SUID activado. Cuando user1 lo ejecute, este fichero se ejecutará como si realmente lo estuviese haciendo root y no user1.
- **GUID:** tiene la misma función que el anterior, la identidad que se toma sin embargo es la del grupo propietario y no la del usuario propietario del fichero.

Hemos hablado de ejecutables y ejecuciones. Lo que sucede, si se aplican los bits anteriores a directorios es que los ficheros bajo aquel directorio sufrirán la herencia de estos bits, es decir los nuevos ficheros creados se crearán con el identificador de usuario o de grupo correspondiente a la propiedad del directorio.

Existe un tercer bit conocido como **stiky bit**. Este bit permite especificar al SO que al terminar de ejecutar un programa no lo elimine de la memoria. Hoy en día ya nos e le da este uso pero nos ofrece una funcionalidad muy interesante. Activar este bit en un directorio permite indicar al SO que evite que los usuarios renombren, muevan o borren los ficheros del directorio. Únicamente el propietario y el súper usuario no se ven afectados por esta restricción.

Para añadir estos comandos adicionales usaremos igualmente el comando *chmod*. Seguro que nos hemos dado cuenta que conforman de nuevo un grupo de 3 bits, por lo que podremos hacer algo como:

chmod 2777 <directorio>

para añadir a <directorio> el bit GUID. En este caso los bits se ordenan: suid, guid, sticky. Lo que significa que los permisos quedan así:

Decimal	Binario	suid	guid	sticky
0	000	No	No	No
1	001	No	No	Sí
2	010	No	Sí	No
3	011	No	Sí	Sí
4	100	Sí	No	No
5	101	Sí	No	Sí
6	110	Sí	Sí	No
7	111	Sí	Sí	Sí

Este grupo se coloca siempre delante del grupo de usuario y es opcional. Sus valores se pueden especificar también según letras, tal como se haría con los permisos normales. Estos permisos a diferencia de los permisos normales afectan a las 3 clases (usuario, grupo y otros) y no únicamente a una de ellas.

chmod +t <directorio># agrega permisos de sticky bit

chmod g+s <directorio> # agrega el setgid al grupo

chmod u+s <directorio> # agrega el setuid usuario

2.3.4 De vuelta a las propiedades

Después de este paseo los permisos, nos fijamos en que lo que nos devuelve el comando `ls -l` todavía contiene más información:

```
ls -l
```

```
drwxr-xr-x 2 rootroot 4096 jul 31 04:34 backups
```

Se observa fácilmente que delante del usuario, recordemos que está en gris claro y es el primer `root`, aparece un número, después del grupo, en gris oscuro o segundo `root`, aparece otro número y a continuación una fecha. El primer número, el que está delante del propietario, nos indica que el directorio que estamos listando contiene dos directorios en su interior. En el caso de que lo que listamos sea un fichero este número será siempre un 1.

En cuanto al número que encontramos detrás del grupo, nos indica el tamaño del fichero que estamos listando. El tamaño mostrado es en bytes. Si queremos un tamaño más adecuado para su lectura podemos especificar la opción `-h` en el comando `ls`. La `h` es por *human readable, legible por humanos* en inglés. El tamaño de un directorio no es el de la suma de sus elementos, por lo que es habitual ver que todos los directorios tienen el mismo tamaño.

Finalmente, vemos una fecha, esta fecha indica la última modificación del fichero o directorio, esto puede llegar a ser muy útil para comprobar cuando se realizó el último cambio a un fichero o directorio.

Juntando conceptos:

En caso que listemos un directorio vacío, el número de ficheros que nos mostrará será de 2. ¿Recordamos por que? Si recordamos lo explicado anteriormente, veremos que siempre que hacemos un `ls`

–a este nos devuelve los valores ‘.’ y ‘..’ que representan el propio directorio y el directorio inmediatamente superior. Hay una excepción a esta regla: el directorio raíz, puesto que no tiene directorio superior.

2.4 Links simbólicos

Un link simbólico no es más que un puntero a otro fichero o directorio. Es decir, lo que en un sistema Windows sería un enlace directo o un alias. Es una manera de acceder rápidamente a rutas (paths) que de otra manera resultarían muy complicadas o tediosas de escribir.

Juntando conceptos

Al ejecutar el comando `ls -l` sobre un directorio que contiene un link simbólico veremos algo como lo siguiente:

```
lrwxr-xr-x  2 root root  4096 jul 31 04:34 backups ->
/home/user/web/backups
```

Si nos fijamos en las propiedades del fichero veremos que la primera letra es una *l*. Esto nos está indicando que *backups* se trata de un link simbólico que apunta a */home/user/web/backups* esto quiere decir que cualquier cosa que hagamos sobre *backups* la estaremos haciendo realmente sobre el path al que apunta.

Con este mismo ejemplo podemos ver que los links simbólicos se tratan en realidad de un arma de doble filo. Aunque en muchas ocasiones pueden llegar a ser muy útiles, en otras pueden ser confusos. Si nos fijamos bien en las propiedades que nos han

aparecido en pantalla vemos que en el link simbólico no tenemos manera de distinguir si apunta a un fichero regular, a un directorio o a un dispositivo. Tampoco tenemos manera de saber si el fichero al que apunta (sea del tipo que sea) existe o no. Es por eso que puede resultar confuso trabajar con los links simbólicos.

2.4.1 Creando links simbólicos

Para crear links tenemos el comando *ln*. Este comando nos creará un link, aunque este no será simbólico. Para crear un link simbólico necesitaremos una opción extra, haremos lo siguiente

`ln -s <objetivo del link> <nombre del link>`

Donde nombre es como queremos acceder y el objetivo del link es el path completo que queremos evitar. ¿Porque la opción *-s*? Esta opción nos permite especificar que se trata de un link simbólico y no de un link común o *hard link*. ¿Que diferencia hay entre los dos? El link simbólico apunta al nombre del fichero del que se crea el link, el no simbólico por contra apunta directamente al contenido del fichero y es un nombre de fichero real, contrario al simbólico que se trata únicamente de un alias. Todo esto puede parecer complejo, pero más adelante entraremos en profundidad a tratar los sistemas de ficheros y entenderemos mucho más sobre los links. En este caso basta con recordad que sin la opción *-s* todo se vuelve en exceso confuso a no ser que se tenga un gran conocimiento de lo que se está haciendo.

3. Dispositivos

Aunque los dispositivos no son otra cosa que ficheros de un tipo especial, la complejidad e importancia que suponen para cualquier sistema operativo, sea o no Linux, hace que merezcan un capítulo completo.

Quizás antes de entrar a pelearse con los dispositivos es un buen momento para aprender algo de Usuarios y grupos, pero dejaré esa elección al sentir del lector. Si se decide leer sobre usuarios puede recurrirse al capítulo 8. Usuarios.

En Linux es posible interactuar con la mayoría de los dispositivos como se interactuaría con un fichero corriente. No es el espacio de usuario el que gestionará la interacción con los dispositivos si no que será el kernel el que decidirá que hacer con dichos dispositivos.

Si hacemos memoria seguro recordaremos que los ficheros de tipo dispositivo se encontraban en /dev. Así pues si echamos un vistazo con el comando `ls -l` en el directorio /dev deberíamos ver los dispositivos de los que disponemos en el sistema. Veremos que tenemos varios, algo que puede resultar incluso sorprendente. Además el resultado será algo como lo siguiente:

```
crw-r----- 1 root kmem 1, 2 mar 4 2011 kmem
srw-rw-rw- 1 root root  0 ago 14 1:16 log
brw-rw---- 1 root disk 7, 0 mar 4 2011 loop0
```

Si nos fijamos vemos que la primera letra de las propiedades no es ninguna letra a la que estemos acostumbrados. Esta primera letra nos está mostrando el tipo de fichero, igual que sucede cuando al

encontrar una 'd' sabemos que se trata de un directorio. En este caso los valores c, s, b nos indican el tipo de dispositivo, existe además otro que no se encuentra en la salida mostrada: el tipo p.

- **c Character Device (Dispositivo por caracteres)** : Como su nombre indica los dispositivos marcados con esta letra nos indican que el kernel del sistema operativo va a trabajar con ellos byte a byte o carácter a carácter, esto se conoce usualmente como un *stream* de datos. Un ejemplo de este tipo de dispositivos sería una impresora.
- **b Block Devices (Dispositivos por bloques)**: En este caso el sistema operativo trabajará con estos dispositivos leyendo o escribiendo bloques de datos. Un ejemplo perfecto son los Discos duros o los usb en los que es habitual trabajar con bloques de 512 bytes.
- **s Socket Devices (Dispositivos socket)**: Este tipos de dispositivos son algo más complejos, y requerirían un estudio en profundidad. En cualquier caso se trata de una interfaz de comunicación entre procesos. Este tipo de dispositivos pueden encontrarse habitualmente fuera del directorio /dev
- **p Pipe Devices (Dispositivos Tubería)**: Aunque el nombre no es muy indicativo, este tipo de ficheros trabajan de la misma manera que los que son de tipo c. La diferencia estriba en que los de tipos c comunican directamente con un dispositivo o con el driver del Kernel que controla el dispositivo mientras que los dispositivos tubería se comunican con procesos y no con dispositivos. Son entonces, igual que los sockets, una interfaz más que un

dispositivo físico.

3.1 Bash, salida estándar y otros

Llegados a este punto, nos detenemos un momento para hacer una segunda incursión en los terminales Unix, en este caso y de ahora en adelante, nos centraremos en bash. Bash es un terminal de Unix que fue programado para el proyecto GNU y que es una evolución de otro terminal muy conocido: sh.

Esto no significa que todos los comandos no valgan para otros terminales, si hacemos memoria, todos los sistemas de los que hablamos están basados en Unix y son sistemas Linux. Esto significa que los conceptos y la mayoría comandos serán los mismos, pero siempre que hagamos referencia a un terminal nos referiremos a un terminal Bash.

Aclarado esto, nos podemos centrar en algunos asuntos más interesantes como son la salida estándar, entrada estándar y otros tipos de salida o entrada.

En todo sistema Linux, cuando trabajamos con un intérprete de comandos o un terminal tenemos 3 fuentes principales para interactuar con la máquina:

- **Entrada estándar (stdin):** Toma como valor por defecto el teclado.
- **Salida estándar (stdout):** Toma como valor por defecto la pantalla, nuestro terminal.
- **Error estándar (stderr):** Igual que la anterior toma como valor por defecto la pantalla. Es decir, nuestro terminal.

Todos los comandos que nos permiten trabajar con nuestro sistema operativo son, a fin de cuentas, programas. Los programas, sin entrar en detalles puesto que daría para largas discusiones, se asemejan a cajas negras de las que no vemos el contenido. Sin embargo, si les proporcionamos unas entradas estos nos darán unas salidas que dependerán de las entradas suministradas. Así pues, los 3 canales mostrados más arriba son los que nos permiten proporcionar las entradas y leer las salidas.

La entrada estándar nos permite especificar que parámetros o valores deben procesar los programas. Así por ejemplo cuando ejecutamos:

```
passwd
```

el comando nos solicita que le entremos una contraseña y espera que se la suministremos por la entrada estándar, es decir, el teclado.

De la misma manera sucede con la salida estándar y el error estándar. Cada vez que un programa se ejecuta nos dará un resultado. Si todo ha ido bien este resultado se mostrará por la salida estándar, en caso contrario el terminal nos ofrecerá un error como los que comentábamos en la sección 1.3 Unix Shell o Terminal, un primer vistazo.

Veámos más arriba que valores toman por defecto esos canales. El hecho de que tome unos valores por defecto ya nos indica que podremos cambiarlo si así nos parece. Por ejemplo, nos puede resultar interesante enviar la salida de error a un fichero, no ver los errores por pantalla o incluso leer la entrada estándar de un fichero en lugar de desde el teclado. Todo esto podremos hacerlo gracias a la re-

dirección.

Lo primero que debemos saber es que cada uno de los canales posee su propio descriptor de fichero, siendo:

- 0 Entrada estándar.
- 1 Salida estándar.
- 2 Error estándar.

Como hemos dicho podremos re-direccionar la salida o entrada de cualquier programa. Para hacerlo tenemos los caracteres '>', '<', y '>>'.

'>' y '>>' nos permiten redirigir la salida estándar. El primero (>) nos permite enviar la salida a un fichero. Si el fichero existe se vaciará su contenido y se rellenará con la salida del programa. Si el fichero no existe se creará antes de llenar el contenido. Por contra, '>>' añade el contenido de la salida al final del fichero especificado sin borrar el contenido existente. Si el fichero no existe lo crea. Veamos un ejemplo:

```
root@PC:~/test# ls #No devuelve nada porque la carpeta está vacía
root@PC:~/test# ls-a > test #Redirecciona la salida de ls -a al fichero
test
root@PC:~/test# cat test # Muestra el contenido del fichero test
.
..
test
root@PC:~/test# ls -a >> test #Redireccionamos la salida de ls -a al
fichero test, pero añadimos el resultado al final
root@PC:~/test# cat test
```

.

..

test

.

..

test

root@PC:~/test# echo 1 > test #El comando echo imprime lo que vaya a continuación por pantalla. En este caso lo redireccionamos al fichero test con un solo > lo que sobrescribirá todo el contenido del fichero.

root@PC:~/test# cat test

1

¿Que sucede si lo que queremos re-direccionar es la entrada? No hay problema, todo lo que debemos hacer es usar '<'. Por ejemplo, el comando *wc* (*word count*) cuenta el número de palabras que recibe por la entrada estándar. Sin embargo, a nosotros podría interesarnos contar las palabras de un fichero. Usaríamos para ello:

wc < fichero.txt

Ahora ya tenemos todo lo que necesitamos saber sobre las re-direcciones de las entradas y salidas estándar. No hemos dicho nada sobre el error estándar. Recordemos que el error estándar tiene como identificador de fichero el número 2. Así pues podemos hacer lo siguiente:

echo 1 2> test

Este comando le está diciendo a nuestro terminal que queremos imprimir un 1 en pantalla y en caso de que diese algún error, queremos redirigir el error al fichero test. Es muy importante que el

identificador de fichero NO este separado por un espacio de '>' o no se comportará de la manera esperada.

Juntando conceptos

Conociendo como funcionan los dispositivos y que podemos redirigir los 3 canales mencionados podemos hacer gran variedad de cosas. Por ejemplo si tuviésemos unos dispositivos de tipo c (como una impresora) podríamos redirigir la salida estándar de un comando a la impresora. ¿Que sucedería en este caso? Estaríamos imprimiendo el resultado del comando por nuestra impresora en lugar de por la pantalla.

3.1.2 Pipes o Tuberías

Hay dispositivos que permiten enviar la información a otro programa en lugar de enviarlo a un dispositivo físico. La salida o entrada de los programas pueden tener más o menos complejidad, pero podemos usar las tuberías de una manera sencilla y rápida en el terminal para enviar la salida de un comando a otro. ¿Para que es útil esto? Cada programa en los sistemas Linux suele tener una funcionalidad muy concreta, la concatenación de programas nos puede ser muy útil en un determinado momento. Por ejemplo, supongamos que quiero contar el número de ficheros en un directorio. Se que el comando `ls` me permite listar los ficheros de un directorio. Necesito, sin embargo, una manera de contarlos una vez listados. Se además que hay un comando, `wc`, que me permite contar el número de líneas o palabras de aquello que le entre como parámetro. Así pues, haré uso del pipe para obtener el resultado deseado. Expresaremos los pipes o tuberías en el terminal mediante el símbolo '|'.

ls -l | wc -l

Devolverá 2 en lugar de listar los ficheros del directorio, puesto que le hemos enviado los resultados de ls -l al programa wc como entrada.

3.2 Extrayendo información de los dispositivos

Ahora que ya sabemos donde se encuentran los dispositivos en nuestros sistemas y como podemos interactuar con ellos, nos puede resultar interesante saber distinguir que dispositivos tenemos conectados en nuestro ordenador. Para caracterizar nuestras máquinas los sistemas Linux nos ofrecen una serie de herramientas que es interesante conocer.

- **lscpu:** Nos da información sobre la arquitectura, número de procesadores, información sobre los procesadores de la máquina, memoria caché y en general todo lo relacionado con las unidades de procesamiento.
- **lshw(list hardware):** Es una herramienta de uso general y no reporta información muy específica pero puede ser útil porque nos lista el hardware que tiene nuestra máquina
- **hwinfo:** Esta herramienta aporta información sobre el hardware que tenemos instalado en nuestra máquina.
- **lspci:** Lista todos los buses PCI y nos da información

sobre los dispositivos conectados a estos buses. Muy útil, por ejemplo, para obtener información sobre la tarjeta gráfica conectada en nuestro ordenador.

- **lsusb:** Igual que en el caso anterior pero este comando nos reporta información sobre los buses usb y los dispositivos conectados.
- **df:** Nos da el espacio de disco para todos los dispositivos conectados. Muy útil tanto para conocer el espacio restante como para saber que dispositivos tenemos conectados y que nombre tienen.
- **hdparm:** Nos da información sobre dispositivos SATA y discos duros. Muy útil para obtener la información de estos últimos.
- **mount:** Este comando está pensado para asociar un sistema de ficheros a un dispositivo. Hablaremos de los sistemas de ficheros en el capítulo 4. *Discos y Sistemas de ficheros*. Si invocamos al comando sin parámetros nos devolverá una lista de los dispositivos que ya tienen asociado un sistema de ficheros junto con otra información útil.

Acabamos de presentar una serie de herramientas de una forma muy breve. La mayoría de ellas podemos encontrarlas instaladas en el sistema, mientras que otras deberemos instalarlas. En los sistemas Linux hay varias maneras de instalar los programas, unas más fáciles y otras más difíciles. En cualquier caso, en Internet es relativamente sencillo encontrar información de cualquier cosa que se

nos pase por la cabeza instalar.

Pese a que se insiste en la importancia de los ficheros del directorio /dev puesto que son muy importantes para poder interactuar con los dispositivos de una forma relativamente compleja, hay que tener en cuenta que el SO va a trabajar con los dispositivos a medida que los encuentre. Es por eso que el nombre de los ficheros en /dev puede cambiar en cada reinicio del sistema.

Puesto que los ficheros son susceptibles de cambio en cada reinicio del sistema es muy importante ser capaces de poder trabajar con estas herramientas u otras de las que se hablará más adelante para poder extraer información y trabajar adecuadamente con los dispositivos.

Aclaración

En este artículo aparecen muchos conceptos técnicos relacionados con el hardware, no es el objetivo de este texto tratar esos temas, sin embargo es muy sencillo obtener las descripciones de los elementos mencionados en Internet.

Aclaración

Se han listado los comandos en su versión más sencilla, sin parámetros. La mayoría de ellos tienen una plétora de parámetros bastante amplia. Conociendo la funcionalidad general del parámetro es posible imaginar algunos de los parámetros. Por ejemplo, el comando *df* nos da el espacio en disco que tenemos en

cada dispositivo. Los números que nos muestra sin embargo, son difíciles de comprender. Es por tanto plausible que existan distintas opciones para mostrar esos números según convenga. En cualquier caso siempre es posible recurrir al comando `man` (de manual) para obtener información detallada del comando y sus parámetros. La invocación sería algo similar a lo siguiente:

```
man ls
```

3.3 Algunos dispositivos de los sistemas Linux

Aunque en `/dev` el número de ficheros puede ser muy grande hay algunos ficheros o más bien algunas convenciones de nombres que nos resultará muy interesante conocer si vamos a trabajar con sistemas Linux.

3.3.1 EL fichero `/dev/null`

Se trata de un dispositivo por caracteres. Toda la información que se manda a este dispositivo queda descartada, es decir no se usa para nada. No se imprime tampoco por pantalla ni se hace nada con ella. No se puede tampoco leer ninguna información de este dispositivo. ¿Para que es útil entonces? Supongamos que queremos lanzar un programa en segundo plano y no queremos que muestre nada por pantalla. Tan solo tenemos que re-direccionar su salida a este dispositivo y ya tenemos el comportamiento que buscábamos.

3.3.2 El fichero `/dev/zero`

Este dispositivo se usa para leer información de él. Es un

fichero que funciona por caracteres igual que el anterior. Siempre que leamos de este dispositivo nos va a devolver 0's. Tantos 0's como sea necesario. Puede ser muy útil para borrar ficheros u otros dispositivos.

3.3.3 Los ficheros /dev/random y /dev/urandom

Estos ficheros proveen de la interfaz para interactuar con el generador de números aleatorios del sistema. Leer de ellos asegura la obtención de un número aleatorio de muy buena calidad. ¿Que significa de muy buena calidad? Los números aleatorios generados por ordenador no son aleatorios al 100%, puesto que se generan con un sistema matemático y son, por tanto, predecibles después de un número de repeticiones muy elevado. Los dos dispositivos que presentamos ahora usan el “ruido” (pulsaciones de teclas, lecturas a disco,...) de otros dispositivos para generar números aleatorios muy difíciles, o casi imposibles, de predecir. En el caso de /dev/random el número aleatorio será realmente aleatorio, es decir no generado por un algoritmo matemático, pero podríamos no obtener un valor si no se ha recogido suficiente “ruido”. En caso de /dev/urandom si el kernel no puede generar un valor aleatorio real se generara uno matemáticamente de muy buena calidad.

3.3.4 Discos duros /dev/sd*

La mayoría de discos duros y dispositivos de almacenamiento usb conectados a nuestro sistema se corresponden con archivos que empiezan por *sd*. Estas siglas provienen de SCSI disk (Small Computer System Interface). El sistema SCSI se desarrollo originalmente para permitir la comunicación entre dispositivos como discos duros y otros periféricos. No es necesario recordar todo esto, basta con recordar

que los dispositivos con estas siglas corresponden a discos duros o unidades de almacenamiento usb. Llegados a este punto, podemos añadir un nuevo comando a la lista de la que ya disponíamos:

```
ls SCSI
[0:0:0:0]      disk      ATA      WDC WD3200AAJS-2
01.0          /dev/sda
```

Este comando, como se muestra en el ejemplo, lista todos los dispositivos scsi conectados a nuestra máquina. No solo eso, si no que además nos indica con que fichero `/dev` están vinculados.

Los números entre corchetes nos dicen la dirección donde el dispositivo está conectado. No es necesario que profundicemos más en la dirección de los dispositivos. La segunda columna nos dice que tipo de dispositivo es, en este caso un disco. A continuación tenemos información del disco y del proveedor y finalmente la última columna nos indica el fichero de correspondencia en `/dev`.

Como convención cada disco físico se representa mediante una letra, por ejemplo:

```
/dev/sda
/dev/sdb
```

Si nos fijamos en el directorio `/dev`, veremos que además de contar con el dispositivo `/dev/sda`, contamos también con `/dev/sda1`. Esto es porque el kernel identifica cada partición dentro del disco duro con un número distinto siguiendo al identificador del dispositivo. Así `sda1` nos indica la partición 1 del disco a. Esto puede sonar confuso ahora pero en el capítulo 4. Discos y Sistemas de ficheros entraremos

mucho más en detalle en estos conceptos.

Se debe recordar que la identificación de los dispositivos puede cambiar entre reinicios del sistema. Es un detalle que resulta muy importante tener en cuenta a la hora de trabajar con los discos que tenemos conectados en nuestra máquina, aunque los kernels modernos solucionan esta problemática.

3.3.5 CD y DVD /dev/sr*

Igual que los discos duros se identifican mediante el prefijo sd, los cd y dvd lo hacen mediante el prefijo sr. Los dispositivos así identificados son de solo lectura, no podemos escribir información en ellos. Si tenemos un CD o DVD con capacidad para escribirlo o reescribirlo, hay CDs regrabables, entonces es muy posible que el kernel lo identifique como un dispositivo SCSI genérico, esto es /dev/sdg0, por ejemplo.

3.3.6 Terminales /dev/tty* /dev/pts/*

Los terminales son dispositivos que permiten mover caracteres desde un proceso de usuario hacia una interfaz de entrada o salida. Quizás sea un buen momento para recordar la sección 3.1 Bash, salida estándar y otros. Los dispositivos de pseudo terminales, identificados por pts son dispositivos que en lugar de comunicarse directamente con un dispositivo hardware como la pantalla, presentan una ventana de comandos u otra interfaz, que permite al usuario interactuar con los dispositivos de entrada y salida. También en la sección 1.3 Unix Shell o Terminal, un primer vistazo, hablábamos de poder cambiar entre diversos terminales mediante la combinación Alt+f1 y así sucesivamente. Cada uno de los terminales en los que podemos trabajar es un terminal virtual, un /dev/tty y por

tanto se trata de un dispositivo dentro del sistema. Si no disponemos de la combinación de teclas Alt+f1 o la tecla fx tenemos otra opción para decirle al sistema que queremos cambiar de un terminal a otro

chvt 1

chvt (change virtual terminal) recibe como único parámetro el número del terminal al que queremos cambiar.

Los terminales en Linux suelen tener dos modos de funcionamiento: modo texto o modo gráfico. Uno solo de los terminales virtuales trabaja en modo gráfico mientras que los otros lo hacen en modo texto. En cualquier sistema Linux, nos será difícil encontrar el terminal que trabaja en modo gráfico en el terminal 1 puesto que el modo gráfico suele iniciar tarde en el sistema y coge el primer terminal libre. Normalmente los primeros terminales se han usado o se están usando para mostrar los mensajes del arranque del sistema en el momento en que se ejecuta la interfaz gráfica.

Si nos encontramos en un terminal virtual que está trabajando en modo gráfico es posible que necesitemos pulsar la combinación de teclas Ctrl+Alt+f1 en lugar de Alt+f1 para cambiar de terminal.

3.3.7 Puertos Serie /dev/ttyS*

Aunque el puerto serie es un puerto bastante antiguo y hoy en día su uso ha caído bastante, pocos portátiles cuentan ya con puerto serie. Sigue siendo útil en determinadas situaciones, sobre todo para trabajar con algunos sensores o dispositivos concretos, por ejemplo la configuración de algunos routers. Configurar la interacción con un puerto serie por consola es extremadamente complejo porque hay que tener muchos parámetros en cuenta, si nos vemos forzados a trabajar con ellos, deberemos buscar estos identificadores.

Hoy en día contamos también con los adaptadores USB a serie para poder conectar dispositivos serie a un puerto USB de nuestra máquina. Probablemente encontremos estos dispositivos vinculados a archivos `/dev/ttyUSB*`.

3.4 Crear ficheros de tipo dispositivo en Linux

En los SO actuales no es necesario que el usuario cree los ficheros que se corresponden con los dispositivos puesto que se encarga de ello Udev. Sin embargo, existen los mecanismos para crear estos ficheros y ver como funcionan puede ayudar a comprender mejor como trabaja el sistema. Es posible saltarse este apartado sin que suponga dejar de entender como trabajan los dispositivos, pero se incluye porque puede aportar algo de valor, aunque sea como curiosidad.

`mknod` es el comando que nos permitirá crear un fichero de tipo dispositivo. Para ello deberemos usar una llamada como la siguiente:

```
mknod /dev/ttyS0 c 4 64
```

El primer parámetro que recibe `mknod` es el dispositivo que queremos crear. En este caso `/dev/ttyS0`. El segundo parámetro nos indica de qué tipo de dispositivo se trata, si tenemos dudas podemos volver al inicio del capítulo para refrescar la memoria. En este caso estamos intentando crear un dispositivo de tipo carácter. Finalmente, debemos especificar el mayor number y minor number. Estos son los números que utiliza el kernel para identificar de forma única los

dispositivos. Antes de poner un número al azar deberemos conocer como identifica el kernel a los dispositivos, puerto serie en este caso, para poder asignarle un mayor number, después necesitaremos conocer cuantos dispositivos del mismo tipo, que comparten mayor number, se han conectado previamente para poder asignarle un minor number.

Incluso después de conseguir asignar mayor y minor number, deberíamos cambiar los propietarios del fichero recién creado y darle los permisos adecuados para que el dispositivo se pueda usar. Finalmente podríamos trabajar con él como si fuese el propio sistema el que lo hubiese reconocido.

```
mknod /dev/ttyS0 c 4 64  
chown root.dialout /dev/ttyS0  
chmod 0644 /dev/ttyS0
```

En los SO modernos este sistema debería usarse para poco o para nada más que para crear un pipe con nombre para interconectar procesos. No es, sin embargo, el objetivo de este texto entrar a las diferencias entre pipes y pipes con nombre.

3.5 Udev

Dado que esta parte es algo compleja, debe ser decisión del lector profundizar un poco más en ella o pasarla de largo. Aunque se

espera que la información dada ayude a comprender mejor como funcionan los sistemas Linux, Udev es un tema complejo y es muy difícil dar información 100% completa. Aún así, la información que viene a continuación es útil para entender como funcionan los dispositivos, como funciona Udev y por extensión los sistemas Linux.

Udev es el gestor de dispositivos de las versiones modernas del kernel de Linux, lo encontramos a partir de la versión 2.6 del Kernel. Este sistema sustituye a sistemas anteriores como *hotplug* o *devfs*. Udev soluciona diversas problemáticas importantes como el cambio de nombres entre reinicios o el hecho de que las operaciones entre dispositivos se realicen únicamente en espacio de Kernel en tanto que Udev permite algunas operaciones en espacio de usuario. Esto puede parecer una mejora menor, pero permite hacer cosas como asignar un nombre inteligible a un dispositivo USB sin tener que recurrir al espacio de Kernel. Podemos ver la sección 1.2 Una visión general de los sistemas Linux para refrescar que son el espacio de kernel y usuario.

Udev es una parte compleja del Kernel que se encarga de gestionar el directorio `/dev` y todos los dispositivos conectados al sistema, tanto en el arranque como los que se conectan a posteriori. Aunque entraremos a ver como funciona Udev, antes de nada es interesante conocer otro de los directorios que encontraremos en nuestros sistemas Linux.

Quando vemos los dispositivos en el directorio `/dev` vemos cantidad de ficheros y no tenemos manera de conocer los detalles relativos a cada dispositivo. No podemos saber que firmware carga u otros detalles que pueden resultar muy interesantes de conocer. Podemos encontrar muchísima

información de cada uno de los dispositivos en el directorio /sys donde está información se estructura para cada uno de los distintos dispositivos. La ruta de acceso será mucho más larga que a través de /dev pero la información que obtendremos será mucho mayor. En /sys/block encontraremos, por ejemplo, toda la información relativa a los dispositivos que se manejan por bloques. Aunque los ficheros de este directorio no suelen ser comprensibles por humanos los nombres de los ficheros permiten hacerse una idea del contenido.

```
cd /sys/devices/pci0000:00/0000:00:1d.1/usb2/2-2/2-2:1.0/block/sdd
ls
alignment_offset holders power stat uevent
dev inflight size subsystem ve_device_add
discard_alignment partition start trace
```

¿Por que hablar ahora de este directorio? Udev trabaja junto con sysfs que es un sistema de ficheros que ayuda le ayuda a lidiar con los dispositivos. Pero, ¿como puede un sistema de ficheros ayudar a Udev? En el momento de boot o arranque del sistema, Udev debe conocer todo el hardware que está conectado para poder trabajar con él. Sin embargo, es posible que no todo el hardware este disponible al mismo tiempo. Puede suceder, como se comentaba en la sección 3.3 Algunos dispositivos de los sistemas Linux, que los dispositivos tengan un nombre de dispositivo en cada reinicio de la máquina. ¿Pero no hemos quedado en que Udev soluciona ese problema? Efectivamente, Udev trabaja por solucionar ese problema. El sistema de ficheros sysfs ofrecido por el kernel, crea un árbol de

directorios bajo el directorio `/sys` en que incluye toda la información de la que dispone el kernel sobre el hardware conectado a nuestro sistema. A partir de esta información suministrada por el Kernel Udev es capaz de solucionar los problemas de cambios de nombres en `/dev` y de obtener toda la información que necesita sobre los dispositivos conectados. Es decir, Udev trabaja con el directorio `/sys` y con el sistema de ficheros `sysfs` muy estrechamente.

3.5.1 Como funciona Udev

Udev funciona como un conjunto de reglas sobre los dispositivos. Estas reglas se pueden definir en ficheros con la extensión `".rules"`. Estos ficheros no son otra cosa que ficheros de configuración y como fichero de configuración podremos encontrarlos en el directorio `/etc`. En este caso concreto, debemos colocar esos ficheros en `/etc/Udev/rules.d/` . El nombre del fichero es irrelevante siempre que tenga la extensión adecuada.

Aunque el nombre no es relevante, hay una consideración que si que lo es. Udev lee las reglas según el orden lexicográfico de estas. ¿Que significa esto? Algo tan sencillo como que una regla dentro de un fichero que empiece con `a` se procesará antes que una regla de un fichero que empiece con `b`. Por ejemplo, el archivo `asd.rules` será leído antes que el archivo `sdf.rules`. Esto significa que si empezamos los ficheros de reglas con un numero de por ejemplo 2 cifras nos será mucho más fácil definir el orden en que nuestras reglas van a ser procesadas. Intentaremos entonces dar un formato del estilo: `12-cardreader.rules`

En caso de conflicto, la regla menos prioritaria gana! Esto significa que el orden de nuestras reglas es importante. Los nombres de los ficheros son evaluados según la siguiente prioridad.

1. Número de menor a mayor.
2. Letras mayúsculas en orden alfabético. OJO! Según el alfabeto inglés.
3. Letras minúsculas en orden alfabético.

Sabemos que Udev funciona con reglas, sabemos donde crear las reglas y que nombre debemos darle a los ficheros donde guardamos las reglas. Todavía no sabemos, sin embargo, que aspecto tiene una regla de Udev ni como trabajar con ella. Vamos a averiguarlo.

Lo primero que debemos saber es que hay tres maneras de que Udev aplique las reglas creadas:

- Al arrancar Udev este aplicará las reglas que tenga definidas
- El Kernel puede mandar uevents a Udev, que se encuentra en el espacio de usuario. Udev parsea los elementos del evento y mira si alguno de los atributos coincide con las reglas que tiene definidas y actúa en consecuencia.
- Se puede forzar la recarga de las reglas con el comando: `udevadm control --reload-rules`. Más adelante aprenderemos más cosas sobre `udevadm`.

3.5.2 Las reglas de Udev

Una regla no es nada más que una cadena del tipo

clave=valor que nos ayuda a identificar un dispositivo seguido por una serie de acciones que realizar sobre dicho dispositivo. Un ejemplo sería la cadena:

clave1 = valor1, clave2 != valor2... acción1, acción2...

El formato de las reglas es sencillo. Usarlas, sin embargo no lo es tanto. Para poder escribir una regla tenemos que referirnos a un dispositivo concreto. Tanto si ya se encuentra conectado al sistema como si se trata de un dispositivo que vamos a conectar necesitaremos poder averiguar la información suficiente que nos permita identificarlo de forma única.

Juntando conceptos

Al inicio cuando hemos empezado a hablar de Udev hemos presentado el sistema sysfs y el directorio /sys donde encontramos toda la información referente a los dispositivos conectados al sistema. Es entonces a ese directorio a donde iremos a buscar la información sobre los dispositivos.

Nos encontramos entonces que tenemos un directorio con información para el dispositivo que queramos siempre que este se encuentre ya conectado al ordenador. Sin embargo, los ficheros que contienen estos directorios no son comprensibles por humanos, es decir por nosotros. De la misma manera en este directorio no tendremos información sobre los dispositivos que conectamos a nuestra máquina después del arranque, es decir dispositivos plug&play. Necesitamos, una manera para extraer la información de estos dispositivos.

Sea cual sea la manera en la que debemos extraer información recurriremos al comando `udevadm`. Este programa ofrece una interfaz para trabajar con Udev en multitud de ocasiones. Será pues muy útil en todas las situaciones.

- Si tenemos un dispositivo `plug&play` el comando `udevadm monitor` nos ofrecerá información de todo dispositivo que se conecte o desconecte del equipo. Recordemos que el kernel genera un evento que llega a Udev. Aprovecharemos este evento para que el comando `udevadm` nos dé la información que precisamos.
- Si por contra, se trata de un dispositivo fijo, toda la información la podremos encontrar en el directorio `/sys`. Lo primero que debemos hacer es localizar la ruta completa que equivale a nuestro dispositivo. Para ello podemos guiarnos por el tipo de dispositivo si sabemos cual es o por el slot de la máquina al que está conectado. Normalmente dentro de `sys` encontraremos sub-directorios suficientemente intuitivos como para encontrar el directorio que estemos buscando. Una vez localizado el path del dispositivo en el directorio `/sys` recurriremos de nuevo al dispositivo `udevadm`. Lo invocaremos como: `udevadm info -a -p /syspath` donde `/syspath` es el path que hemos encontrado.

Sea cual sea el método por el que encontramos información

del dispositivo, este nos ofrecerá un conjunto de parejas clave valor que nos darán información estructurada de forma jerárquica. Como ya se vio en los directorios es habitual en los sistemas Linux encontrar organizaciones jerárquicas, esto facilita mucho la manera de trabajar puesto que es fácil hacer suposiciones fundadas sobre como funcionarán elementos, programas o servicios desconocidos. En el caso de Udev obtendremos información en los siguientes niveles:

- **KERNELS:** Como llama el kernel al dispositivo inicialmente.
- **SUBSYSTEMS:** A que sistema está conectado el dispositivo (pci, scsi).
- **DRIVERS:** Nombre del controlador que gestiona el dispositivo.
- **ATTRS:** Los atributos son variables i dependen del tipo de dispositivo, normalmente los veremos expresados como ATTRS{nombre}=valor.
- **ENV{var}=="valor":** Variables de entorno.

Toda la información mostrada en forma de clave/valor podrá ser usada en las reglas exactamente como se muestra en la información para identificar un dispositivo. Ya sabemos entonces con que deben corresponderse las parejas clave=valor de las que hablábamos en el formato de las reglas. Solo debemos tener en cuenta que para referirse a un dispositivo de forma única debemos ser cuidadosos y usar atributos que no den pie a confusión. Por ejemplo, si tratamos de identificar un disco duro y lo hacemos a través del atributo

vendorID, ID del vendedor, es posible que estemos identificando a más de un dispositivo, sobre todo si nuestra máquina tiene más de un disco duro y todos son del mismo fabricante, cosa que suele ser habitual. Deberemos, por tanto, tratar de identificar los dispositivos de la forma más específica posible. Siempre que queramos afectar a un solo dispositivo, por supuesto.

Es posible que en alguna ocasión tengamos la necesidad de hacer referencia a varios dispositivos. Podremos usar entonces expresiones regulares o substituciones en las expresiones clave/valor. Entraremos en detalle sobre la substitución de cadenas o caracteres en el capítulo 5 y un poco más adelante veremos algunas de las más comunes a Udev. Mientras tanto podemos ver algunos ejemplos para hacernos una idea.

`KERNEL=="sd[b-g]"` coincidirá tanto con `sdb` como con `sdc,sdd,sde,...,sdg`

`ATTRS{model}=="*SD*"` en este caso los asteriscos substituyen a cualquier cadena de caracteres. Es decir, esta pareja clave-valor es equivalente a cualquiera de las siguientes:

`ATTRS{model}=0054SD2689`

`ATTRS{model}=ASG05SD01FE89A04`

`ATTRS{model}=01SD02`

etc

Advertencia

Aunque en la teoría cualquier expresión regular puede ser usada en la substitución de los valores, en la práctica Udev no siempre

entiende todas las expresiones regulares que se le pasan. Suele ser por tanto extremadamente recomendable testear todas las reglas antes de darlas por buenas.

Ya tenemos los atributos, es momento entonces de ver que podemos hacer una vez somos capaces de referenciar uno o varios dispositivos. El objetivo del texto es comprender como funciona Udev y proveer al lector con la información más precisa y actualizada posible, aunque a veces este simplificada, en el caso de Udev esto resulta complicado puesto que cambia entre distintas versiones del kernel y a veces la teoría no se corresponde 100% con lo que sucede en la práctica. La información que se da a continuación es lo más correcta y actualizada posible, sin embargo es posible que haya algún cambio que afecte a la manera de tratar con las opciones, por eso de nuevo, es necesario insistir: **al trabajar con Udev es extremadamente importante testear las reglas que se crean.**

Al realizar acciones, los parámetros ENV que comentábamos anteriormente pueden adquirir especial relevancia, puesto que nos permiten incluir variables de entorno que afectaran a las acciones que se ejecuten a continuación.

Aclaración

Una variable de entorno es una variable dinámica que toma un valor que afecta al comportamiento de los programas de nuestra máquina. Normalmente se pueden manipular desde el terminal y su valor se puede cambiar de forma temporal mediante el comando set o de forma permanente acudiendo a ficheros de configuración.

Las acciones que vamos a poder usar en nuestras reglas son

las que siguen:

- **NAME:** Cambia el nombre que el kernel da al dispositivo.
- **SYMLINK:** Se crea un enlace simbólico al dispositivo con el nombre especificado.
- **OWNER:** Cambia el usuario propietario del dispositivo. Podemos recordar la sección 2.3 para entender mejor esta acción.
- **GROUP:** Cambia el grupo propietario del dispositivo.
- **MODE:** Especifica los permisos para el dispositivo, por ejemplo `MODE="0777"`
- **RUN:** Ejecuta el programa especificado cuando se cumple la regla. `RUN+="path del programa"`. Al usar la acción run se usa += para especificar el valor en lugar de un igual. De esta manera podemos ejecutar más de un script si es necesario y no eliminamos lo que ya se ejecutaba antes de añadir nuestro programa.
- **PROGRAM:** Ejecuta un programa igual que run cuando la regla coincide, sin embargo, el uso de program en lugar de run se da cuando el resultado es importante. Por ejemplo, supongamos que queremos darle un nombre al dispositivo en función a la salida de un script. en ese caso nuestra regla contendría las acciones: `PROGRAM="/scripts/renamer"`, `NAME=%c ó PROGRAM="/scripts/renamer", NAME=%c{1}, SYMLINK=%c{2}`

- **ATTR{key}:** Añade un atributo de nombre key con el valor especificado.

Hay bastantes más opciones disponibles, estas son solo algunas de las más habituales. Dependiendo de la versión del Kernel con la que trabajemos algunas de las otras opciones estarán soportadas o no, en cualquier caso el uso de unas u otras dependerá siempre de lo que queramos hacer. Sabiendo lo que debemos buscar será relativamente sencillo encontrar una relación de las acciones que podemos emplear.

3.5.3 Substitución de cadenas

Finalmente, para terminar con la visión de como usar las reglas de Udev, debemos ver la substitución de cadenas. Igual que para las parejas clave/valor podemos emplear expresiones regulares y substitución de cadenas, también podemos hacerlo para trabajar con las acciones.

- **%k:** substituye al número con el que el kernel identificaría al dispositivo. Por ejemplo si el kernel nombra originalmente al dispositivo como sda3, %k es equivalente a sda3
- **%n:** substituye al número que el kernel ha asignado al dispositivo, por ejemplo para sda3, %n sería igual a 3
- **\$driver:** substituye el valor del driver que se ha encontrado al buscar en las propiedades del dispositivo.
- **%s:** devuelve el punto de montaje del sistema de ficheros sysfs, en casi todas las versiones de Linux

/sys

Existen más substituciones de cadenas posibles, estas son solo algunas de ellas. Si nos fijamos, casi todas empiezan por el carácter '%'. Esto significa que si queremos usar este carácter como parte de alguna acción o algún atributo tendremos que emplear en su lugar '%%'.

Veamos un par de ejemplos:

Queremos dar nombre a un dispositivo en función del nombre que ya le ha dado el kernel. Incluiremos en nuestra regla:

```
PROGRAM="/scripts/renamer %k", NAME=%c{1}, SYMLINK=%c{2}
```

Queremos cambiar el propietario de un dispositivo según el driver que cargue.

```
PROGRAM="/scripts/changeOwner $driver", OWNER=%c
```

Hemos insistido en la importancia de testear las reglas que creamos antes de darlas por buenas. Vamos a ver como hacerlo. De nuevo recurriremos a udevadm

```
udevadm test /syspath
```

Este comando nos informara de cada una de las acciones que ejecutaría y como las ejecutaría, de esta manera podemos hacernos una idea de que reglas quedarían correctamente definidas y cuales necesitarían ser revisadas.

Una vez tenemos probadas las reglas únicamente nos queda forzar su carga. Si revisamos este mismo apartado 3.5, veremos que tenemos dos maneras de hacerlo. La primera reiniciando el sistema y la segunda recurriendo de nuevo al comando udevadm.

```
udevadm control --reload-rules
```


4. Discos y sistemas de ficheros

Hemos hablado de ficheros y hemos visto que los dispositivos son un tipo concreto de fichero. Luego hemos hablado de dispositivo y hemos visto que los discos son un tipo concreto de dispositivo. Este es el último nivel al que vamos a bajar. Los discos o dispositivos de almacenamiento son extremadamente importantes en cualquier sistema actual, Linux o no, puesto que es donde se almacenan los datos. También es posible que si debemos trabajar con sistemas Linux tengamos que acostumbrarnos a trabajar a menudo con ellos. Si todo lo demás lo hemos dejado bien configurado, es posible que los discos se conviertan en uno de nuestros mayores quebraderos de cabeza: los discos se estropean, los usuarios reclaman más espacio, etc...

Antes de entender como trabajan los discos y como vamos a poder interactuar con ellos debemos entender por encima con que estamos trabajando. Un disco es un dispositivo físico que nos permite almacenar información de forma permanente. Entendiendo permanente como que no se pierde cuando deja de alimentarse eléctricamente la máquina.

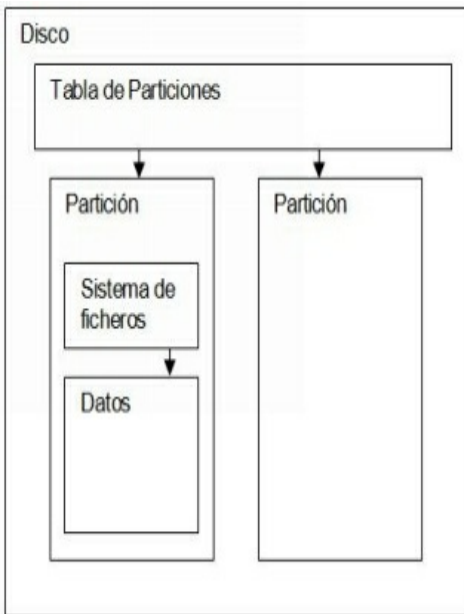
Hoy en día distinguimos dos tipos de disco. Por un lado los tradicionales que están compuestos de varios platos y una aguja que se sitúa sobre estos para poder leer del disco. Habitualmente, al comprar un disco duro nos dicen que estamos comprando, por ejemplo, un disco de 1GB a 5600rpm. Ese último valor nos dice que la aguja es capaz de dar 5600 vueltas cada minuto sobre los platos del disco. Se debe tener en cuenta que este es únicamente el mejor de los casos. Si dibujásemos una circunferencia sobre un plato, eso es lo que

conoceríamos como cilindro. Cada plato tiene varios cilindros. Esto significa que siempre que leamos sobre el mismo cilindro tendremos una velocidad de 5600rpm. Sin embargo, si necesitamos reposicionar la aguja para que lea de otro cilindro o plato, a esa velocidad que nos están vendiendo habrá que restarle el tiempo de recolocación de la aguja. Es, por tanto, importante tener en cuenta los límites de los cilindros para trabajar de la forma más eficiente con los discos. Los sistemas modernos ya no requieren que sea el usuario o el administrador el que se encargue de determinar y aprovechar estos valores, sin embargo, siempre es bueno saber que es lo que tenemos entre manos.

El otro tipo de disco que solemos encontrarnos hoy en día y cada vez más, es el disco de estado sólido o SSD. Este tipo de disco no tiene aguja y no tenemos por tanto la penalización de reposicionarla cuando debemos cambiar de cilindro o de plato. Los discos de estado sólido suelen estar basados en memoria flash o en tecnología RAM (random access memory, memoria de acceso aleatorio) esto significa que, por ahora, suelen tener una vida menor que los discos basados en platos magnéticos y que una vez perdida la información resulta casi imposible recuperarla. Es por ello, que muchas tecnologías de SSD utilizan mecanismos para redundar la información y así evitar pérdidas irrecuperables. Como ventajas nos ofrecen un acceso mucho más rápido a los datos, a menor coste energético y sin apenas ruido. En lo que afecta a los sistemas Linux, debemos saber que los discos de estado sólido acceden a los datos por bloques, normalmente de 4096bytes. Esto significa que cuando trabajemos con el disco, deberemos tratar de alinear las particiones, veremos de qué se trata en la sección 4.1 Particiones y Tablas de particiones, a estos 4096bytes si

queremos obtener el mejor rendimiento posible de estos discos.

Independientemente de cual sea el tipo de disco, la manera que estos van a tener de trabajar con los sistemas Linux o con cualquier otro tipo de sistema es la que se muestra en la imagen.



Las particiones son subdivisiones del disco en partes más pequeñas. Históricamente las particiones respondían a limitaciones del

los propios sistemas Linux a la hora de gestionar los dispositivos. Hoy en día, pese a haber perdido la funcionalidad original, las particiones siguen usándose en la mayoría de sistemas. Por ejemplo, suele ser habitual instalar los componentes del sistema en una partición mientras se mantienen los datos en otra. Esto permite, si algún día no podemos arrancar, recuperar la instalación del sistema sin alterar en ningún momento los datos. Ambas particiones están en el mismo disco y una corrupción de datos o que se estropee el disco no se solucionará por tener particiones pero si que puede ser de utilidad cuando el error no viene de la parte hardware.

En cuanto a las tablas de particiones, son las encargadas de gestionar cada una de las particiones, saber donde empiezan y donde acaban y otro tipo de información. Finalmente, la última capa que encontramos antes de poder acceder a los datos es el sistema de ficheros. El sistema de ficheros no es otra cosa que un base datos de ficheros y directorios con la que estamos acostumbrados a interactuar, sea desde una interfaz gráfica o desde el terminal. Más adelante, entraremos en más detalle sobre los sistemas de ficheros.

4.1 Tablas de particiones y particiones

Dado que las tablas de particiones preceden en importancia de gestión a las propias particiones empezaremos por ellas. La tabla de particiones es la encargada de identificar y manejar las distintas particiones de nuestros discos. Tradicionalmente la tabla de particiones usada era conocida como Master Boot Record (MBR o DOS). Este tipo de tabla de particiones es capaz únicamente de trabajar con discos de hasta 2 TB de capacidad. Esto puede parecer mucho, pero es habitual que al trabajar con servidores se convierta en

una cantidad ridículamente pequeña. Además, en el caso de la tabla de particiones MBR tenemos un límite de 4 particiones por disco.

La limitación de 4 particiones por discos se reduce tan solo a particiones físicas. Para solventar el límite de 4 particiones, MBR permite crear tres particiones físicas y una extendida. En la partición extendida es posible crear hasta 23 particiones lógicas.

Advertencia

Esto no es así en todos los sistemas operativos, el soporte a cada tabla de particiones es dependiente al sistema operativo y por tanto esas 23 particiones lógicas pueden no ser 23.

La principal diferencia entre particiones lógicas y físicas está en la forma de gestionarlas por parte de la tabla de particiones. Las particiones lógicas se encontrarán siempre dentro de la partición extendida. Por otro lado, las particiones lógicas no pueden ser *bootables*. ¿Que significa esto? Significa que únicamente podemos instalar un sistema operativo en particiones físicas. Esto quiere decir que una tabla de particiones MBR no podrá soportar más de 4 sistemas operativos en el mismo disco físico.

Existen otros tipos de tablas de particiones como GPT. GPT soluciona la limitación de las 4 particiones físicas. Además soporta discos duros de hasta 9 ZB, zettabytes. 1 zettabyte equivale a 1000 millones de TB. La mayoría de sistemas Linux ofrecen compatibilidad completa o casi completa con este tipo de tablas de particiones. Antes de usarla deberemos asegurarnos que nuestra distribución le da

soporte, aunque en ocasiones, si necesitamos trabajar con discos mayores a 2TB, no nos quedará otra opción.

Tanto MBR como GPT, esta segunda en mayor medida, están basadas en Intel y, en sus versiones anteriores, en IBM PC (hablamos de chips no de procesadores). Esto hace que en algunas ocasiones estas opciones no sean viables. Es el caso de los MAC basados en Power PC. Los MACS que usan Intel tienen soporte completo para GPT y MBR. No es el caso, como decíamos de los MAC Power PC. En este caso la tabla de particiones más habitual es APM (Apple Partition Map) Esta tabla de particiones nos presenta un problema. No es compatible con Windows. Dado que podemos tener una única tabla de particiones por disco eso significa que no podremos tener un sistema operativo Windows en un disco con este tipo de tabla de particiones.

Aclaración

Los distintos sistemas de arranque del sistema están estrechamente relacionados con la tabla de particiones que tengamos en nuestro disco y con los discos en si. En el capítulo 9. Arranque del sistema, profundizaremos en los sistemas de arranque del sistema y en los diversos gestores que lo permiten. Sin embargo, será inevitable que al hablar de discos salgan varias referencias al arranque del sistema. Hemos hablado de que en un solo disco no podríamos tener más de 4 sistemas operativos para MBR o que un Windows o un OSX de MAC no pueden compartir el mismo disco si tenemos una tabla de particiones de tipo APM. Si tenemos más de un disco podemos tener cada disco con una tabla de particiones distintas y evitar estas limitaciones. Sin embargo, debemos ser cuidadosos.

Solo un disco duro puede ser el disco primario en nuestra máquina. Esto significa que el gestor de arranque que tengamos podrá gestionar los distintos sistemas que tengamos en un mismo disco, pero no siempre podrá gestionar todos los sistemas si están en distintos discos duros. Así pues, si queremos tener varios discos y cambiar entre ellos para escoger una tabla de particiones u otra y por tanto un sistema operativo u otro tendremos que recurrir a la BIOS (basic input output system). Esto no significa que con la tabla de particiones adecuada, no podamos tener distintos sistemas operativos en el mismo disco.

Aunque MAC OS está basado en Unix y por tanto muchas de las cosas de este texto serán igualmente validas, el objetivo son los sistemas Linux, es por eso que en este caso nos centraremos en MBR y GPT como tipos de tablas de particiones.

Sea cual sea la elección que hagamos para nuestra tabla de particiones, deberemos tener en cuenta que la tabla de particiones debe existir, no podremos usar, o más bien será muy complicado usar, un disco que no tenga una tabla de particiones. Esta tabla es la que le dirá, por entenderlo de una forma sencilla, a nuestro sistema Linux donde debe empezar y acabar de leer un disco.

4.1.1 MBR

Este tipo de partición nace al final de los años 70 para dar soporte a los ordenadores personales IBM. Más tarde se amplía su uso y ahora tenemos un tipo de partición compatible con todos los sistemas PC.

Master Boot Record o MBR guarda la información relativa a las particiones al inicio del disco. Dado que se trata de información

permanente que debe sobrevivir al apagado de la máquina no queda otra que almacenar-la en el propio disco. En este caso se almacena siempre en el primer sector del disco. En cualquier disco con una tabla de particiones MBR, si copiamos los primeros 512 bytes a otro disco, en las primeras 512 posiciones libres, habremos duplicado el disco, copiando tanto las particiones como la tabla que las gestiona.

```
dd if=/dev/sd* of=mbr.bk bs=512 count=1 #Copiamos la tabla de  
particiones a mbr.bk
```

```
dd if=mbr.bk of=/dev/sd* bs=512 count=1 #Copiamos la tabla de  
particiones a un nuevo disco
```

Aún así, la tabla de particiones no ocupa el primer sector completo. No ocupa esos 512 bytes si no que ocupa una parte de ellos. Esos primeros bytes suelen repartirse como se mostrará a continuación. Es importante recalcar el “suelen” en la frase anterior puesto que con los años esto puede estar sujeto a algunos cambios menores. Al final, sin embargo, debe mantenerse la compatibilidad por lo que los cambios no son muy relevantes. Los primeros bytes de un disco duro con una tabla de particiones MBR se distribuyen así:

Offset del sector (decimal)	Offset del sector (hexadecimal)	Longitud	Descripción
000-445	0000-01BD	446 bytes	Área de código
446-509	1BE-1FD	16 bytes	Tabla de particiones
510-511	1FE-1FF	2 bytes	Firma del registro

- **Del byte 000-445:** Área de código. Este código es el que se carga en primer lugar y es por tanto el encargado de contener el gestor de arranque o el programa encargado de lanzar el cargador del sistema operativo. Es muy importante que este código sea de como mucho 439bytes puesto que los bytes siguientes suelen estar reservados para la firma del disco y para un separador de 2 bytes.
- **Del byte 446-559:** La tabla de particiones en si misma, aquí es donde se almacena la información referente a las particiones en nuestro disco duro.
- **Del byte 510-511:** Una firma que identifica el MBR y que nos indica su final. Siempre es la misma. 0x55AA en su notación hexadecimal o AA55h en notación hexadecimal también pero en little endian.

Aclaración

Una rápida aclaración sobre little endian y big endian: ambas denominaciones son únicamente una manera de representar los números en un ordenador. La diferencia estriba en si se escriben primero los número más significativos o los menos significativos. Es decir, el sistema big endian escribe las cifras de izquierda a derecha mientras que el sistema little endian lo escribe de derecha

a izquierda. Esto es una simplificación, en un ordenador no hay derecha e izquierda, simplemente posiciones de memoria más altas o más bajas. Siguiendo con la simplificación, la mejor manera de entenderlo sea, probablemente, un ejemplo: el número dieciséis (10 en hexadecimal) podría escribirse como: 10 o como 01 según la notación.

Dentro de la tabla de particiones situada en los bytes del 446 al 559 de nuestro primer sector de disco, encontramos la tabla de particiones. Esta tabla contiene 4 espacios para especificar 4 entradas de particiones. Cada uno de estos espacios cuenta con 16 bytes. Si hacemos las cuentas veremos que 16 bytes por 4 espacios disponibles hacen un total de 64 bytes. Entendemos también porque no podemos tener más de 4 particiones físicas en un disco duro con una tabla de particiones MBR: no hay espacio suficiente para representar todas las particiones. Dentro de cada entrada para la tabla de particiones tenemos 16 bytes que se reparten, para cada una de las entradas, como sigue:

Offset (decimal)	Longitud	Descripción
0	1 bytes	Indicador de arranque (80h=activo)
1-3	3 bytes	Primer sector en CHS
4	1 bytes	Tipo de la partición
5-7	3 bytes	Último sector CHS

8-11	4 bytes	Primer Sector
12-15	4 bytes	Tamaño de la partición en sectores

Se entiende que los offsets mostrados son relativos a la tabla de particiones y a la entrada de partición a la que corresponden.

- **Indicador de arranque:** Indica si se puede arrancar desde la partición o no. Para poder arrancar desde ella debe contener un sistema operativo válido.
- **Primer sector en CHS:** CHS significa por sus siglas en inglés: Cilindro, Cabezal, Sector. Estos 3 bytes codifican entonces el cilindro el cabezal y el sector inicial de nuestra partición. Si hacemos un cálculo rápido veremos que esto no nos permite llegar más allá de los 200 MB. No es posible, por tanto, iniciar una partición más allá de los 200MB solo con estos valores.
- **Tipo de la partición:** Con este byte podemos indicar el tipo de la partición. Como se puede observar, existen un máximo de 256 (2^8) tipos posibles. Por ahora no están todos ocupados. No entraremos a revisar los 256 tipos que hay ni que valor tienen pero si que resulta interesante revisar algunos de ellos:
 - **82:** Partición Swap para sistemas Linux
 - **83:** Partición para sistemas de ficheros nativos Linux

- **EE:** Es un tipo para indicar una falsa partición que ocupa todo el disco. En caso de querer tener el disco como un disco de datos sin preferencia por ningún sistema de ficheros concretos y con una sola partición este tipo puede ser nuestra solución.
- **05 ó 0F:** Indica una partición extendida. En el segundo caso suele usarse cuando la partición extendida empieza más allá de del cilindro 1024.
- **Ultimo sector en CHS:** De nuevo los valores CHS. Conjuntamente con los valores anteriores permiten especificar los valores adecuados para el Cilindro, Cabezal y Sector.
- **Primer Sector:** Especifica el primer sector donde empieza la partición. Esta referencia es absoluta. Es decir, el primer sector es el 0. Esto significa que, dado que tenemos una limitación de 4 bytes para este sector, el último sector posible puede ser 0xFFFFFFFF. A512 bytes por sector, 4294967295 (0xFFFFFFFF en decimal) por 512 = 2199023255040 bytes. O lo que es lo mismo, no podemos usar discos de más de 2TB, o más concretamente de más de 2048GiB si usamos una tabla de particiones MBR.
- **Tamaño de la partición en sectores:** El nombre es bastante claro, el tamaño de 4 bytes para este campo nos muestra también la limitación de 2TB inherente a

este tipo de tabla de particiones.

Podríamos profundizar mucho más en los sistemas MBR, sin embargo hemos visto un detalle de como se estructuran que nos ayuda a hacernos la idea de como funciona este tipo de tabla de particiones.

4.1.2 GPT

GPT, de las siglas GUID Partition table, surge como un tipo de tabla de particiones que intenta paliar las limitaciones de MBR. A diferencia de MBR, en este tipo de tabla de particiones no se requiere de un área de código para poder ejecutar el gestor de particiones o para poder lanzar el arranque del sistema. Para poder realizar estas tareas las tablas de particiones GPT confían en una interfaz desarrollada por Intel conocida como EFI. Esta interfaz ofrece funcionalidades a muchos niveles, puesto que está pensada para interactuar como puente entre el firmware base y el sistema operativo, en nuestro caso el sistema Linux. La idea es remplazar a la antigua BIOS ofreciendo una interfaz más amigable y con más funcionalidades antes de arrancar el sistema. Podríamos decir que hablamos de una BIOS avanzada. Una de las cosas que nos ofrece EFI es la capacidad de carga de las unidades de almacenamiento.

Aunque podría resultar muy interesante, no entra dentro del objetivo de este texto discutir las características del sistema EFI, aún así, no resulta complicado encontrar información al respecto por Internet.

Pese a confiar en el sistema EFI para el arranque, la tabla de particiones GPT, mantiene en los primeros bytes del disco un MBR. Este se mantiene por motivos de compatibilidad con sistemas que no den

soporte a GPT. La tabla de particiones MBR toma determinados valores para que el disco sea visto como una única partición de tipo GPT por aquellos sistemas incapaces de entender este tipo de tabla. Para ello se le asigna como tipo de partición el tipo 0xEE. Los sistemas EFI ignoran directamente la parte MBR incluida en los discos.

Para la gestión de las particiones, GPT ha substituido el modelo CHS; recordemos Cilindro, cabezal, sección; por el sistema LBA o de direccionamiento por bloques lógicos. Realmente LBA es una abstracción sobre CHS que permite numerar cada bloque mediante una etiqueta empezando en 0. La substitución permite al sistema CHS subyacente tomar valores mayores. No deberemos preocuparnos de como gestionar dichos valores. Normalmente este valor está también limitado por la parte superior.

En este tipo de tablas, la LBA0 está destinada al MBR para ofrecer compatibilidad. LBA1 corresponde a la cabecera de la tabla de particiones y a continuación se listan las LBA que corresponde a cada una de las particiones. Cada LBA se corresponde a un sector de disco, igual que pasaba con MBR anteriormente, que equivalía a un sector de disco. Las cabeceras que identifican cada partición necesitan de 128 bytes. Si hacemos una sencilla división veremos que cada LBA nos da la opción de disponer de 4 particiones. No solo eso, además tenemos la oportunidad de mayores particiones puesto que ahora disponemos de 128 bytes para representar cada partición y no solo de 16 bytes como pasaba en MBR.

Es importante recalcar antes de seguir, que en el caso de GPT la información de las particiones así como la cabecera de la tabla de particiones se guardan de forma redundante, tanto al principio del disco como al final. Esto significa que, a diferencia de MBR, no será

sencillo duplicar la información de las particiones y tabla de particiones de un disco. No es sencillo porque, en primer lugar, tendremos que guardarnos algo más que el primer sector y en segundo lugar, porque será necesario guardar la información del final del disco también. Tendremos entonces que conocer cuantos sectores tiene el disco y cuantos comprenden los LBA de las particiones para poder duplicarlo. Suele ser habitual, aunque esto no es siempre así, que los primeros 34 o 63 sectores estén ocupados por LBAs.

Así pues, podremos hacer en este caso uso de comandos similares a los que usábamos en MBR, solo que serán algo más complejos.

```
dd if=/dev/sd* of=mbr.bk bs=512 count=63 #Copiamos los primeros 63 sectores
dd if=mbr.bk of=/dev/sd* bs=512 count=63
dd if=/dev/sd* of=mbr2.bk bs=512 count=63 skip=18863 #Copia 63 sectores a partir del bloque 18863
dd if=mbr2.bk of=/dev/sd* bs=512 count=63 seek=18863 #Copia 63 sectores sin escribir en los primeros 18863 bloques del destino
```

Como se puede ver en los comando anteriores, el comando dd nos permite copiar únicamente los últimos bloques, sin embargo, deberemos conocer de antemano cuales son los últimos bloques.

Ahora que nos hemos hecho una idea general de como se estructura una tabla de particiones GPT, vamos a ver más en detalle como son este tipo de tablas. Vamos a echar un vistazo tanto a la cabecera de la tabla que se encuentra después el MBR como a la manera en como podemos representar cada una de las particiones. Dado que la LBA1 está ocupada por la cabecera de la tabla GPT

vamos a ver que es exactamente esta cabecera:

Offset (decimal)	Longitud	Contenido
0	8 bytes	Firma ("EFI PART", 0x 45 0x46 0x49 0x20 0x50 0x41 0x52 0x54)
8	4 bytes	Revisión, es decir la versión de GPT
12	4 bytes	Tamaño de la cabecera en bytes, normalmente (0x5C 0x00 0x00 0x00) que son los 92 bytes representados en esta tabla
16	4 bytes	CRC32 de la cabecera (comprobación de integridad)
20	4 bytes	Reservados. Deben tomar el valor 0.
24	8 bytes	LBAd de la cabecera. Recordemos que hay redundancia
32	8 bytes	LBAd de soporte.
40	8 bytes	Primera LBAusable para particiones
48	8 bytes	Última LBAusable para particiones

56	16 bytes	Identificador global único del disco
72	8 bytes	Primer LBA para los LBA que representan las particiones. Siempre es el LBA2
80	4 bytes	Número de particiones permitidas
84	4 bytes	Tamaño de una entrada de partición. Como hemos dicho suelen ser de 128 bytes.
88	4 bytes	CRC32 de los LBA que representan las particiones.

El resto de la cabecera está reservado y solo contiene ceros. No parece relevante, pero es importante completar la cabecera de la tabla con 0's. Los contenidos de esta cabecera son en si mismos suficientemente informativos.

Aclaración

CRC32 es un método de comprobación basado en redundancia cíclica de los datos que está orientado a garantizar la integridad de los mismos. Algo que resulta crítico cuando trabajamos con un disco.

Los LBA que permiten representar las particiones se agrupan en una estructura conocida en inglés como array. Así pues, el array de LBA es la estructura que contiene todos los LBA que representan alguna partición.

Cada partición está representada por su propia cabecera. Esta cabecera, como se ha reiterado en el texto, suele ocupar 128 bytes, permitiendo de esta manera representar un espacio mucho mayor que en su predecesor MBR. Vamos a ver como se estructura:

Offset (decimal)	Longitud	Contenido
0	16 bytes	Tipo de partición GUID
16	16 bytes	Identificador global único de partición
32	8 bytes	Primer LBA en little endian
40	8 bytes	Último LBA (incluido)
48	8 bytes	Atributos (solo lectura, oculta, evitar montado automático, etc...)
56	72 bytes	Nombre de la partición

Los GUID del tipo de partición, igual que en MBR definirán que uso le podemos dar a cada una de las particiones de nuestro disco. Este GUID es altamente dependiente del sistema operativo y del uso que le queramos dar a nuestra partición, es por ello que vamos a ver cuales nos resultan los más interesantes para trabajar con los sistemas Linux.

- **024DEE41-33E7-11D3-9D69-0008C781F39F:**

Partición MBR.

- **C12A7328-F81F-11D2-BA4B-00A0C93EC93B:**
Partición de Sistema EFI.
- **EBD0A0A2-B9E5-4433-87C0-68B6B72699C7:**
Partición de datos para sistemas Linux.
- **A19D880F-05FC-4D3B-A006-743F0F84911E:**
Partición RAID para sistemas Linux.
- **0657FD6D-A4AB-43C4-84E5-0933C84B4F4F:**
Partición de SWAP en sistemas Linux.
- **E6D6D379-F507-44C2-A23C-238F2A3DF928:**
Partición de gestión de volúmenes virtuales(LVM) en sistemas Linux.

No suele ser habitual preocuparse por estos valores, puesto que los comandos o programas que empleemos para realizar las particiones o la tabla de particiones gestionarán la mayoría de ellos, si no todos, por nosotros. De la misma manera, los atributos de una partición los podremos especificar de una forma más amigable que insertando números hexadecimales en un offset dado. Pese a ello, siempre resulta interesante conocer la estructura interna. En un momento dado, sabemos que insertar un bit concreto en un offset concreto puede hacer que nuestra partición, a la que no teníamos acceso, deje de estar en modo de solo lectura. Los atributos que se pueden añadir a una partición son bastante numerosos puesto que se tratan a nivel de bit. Esto significa que si un bit está a 1 ese valor se activa, si está a 0 se desactiva. Este tipo de valores basados en el valor que toma un bit se conocen usualmente como flags. Esto significa

también que tenemos 8bytes por 8 bits en cada byte, 64 bits de atributos o, dicho de otra manera, 64 posibles atributos. Así por ejemplo:

- El bit 1 especifica que el firmware EFI debe ignorar la partición.
- El bit 60 es para activar el modo de solo lectura
- El bit 62 es para hacer la partición oculta
- El bit 63 es para evitar el auto montado

Disponemos de toda la información de la que necesitamos para hacernos una idea de como el sistema operativo trabaja con nuestros discos. No solo eso, si no que seguro que se nos han planteado también algunas preguntas acerca de como arranca el sistema. No debemos preocuparnos, más adelante, en el capítulo 9, hablaremos sobre el arranque del sistema.

4.2 La herramienta dd

dd es una herramienta muy potente que permite leer de un fichero o de un stream de entrada, un dispositivo de tipo `s` o `c`, y escribir lo leído en un fichero o stream de salida. La funcionalidad que ofrece dd es extremadamente útil tanto para trabajar con dispositivos como con discos. dd no limita su utilidad únicamente a estos trabajos, si no que además permite realizar copias de datos en el formato especificado y según los bloques y tamaño de bloques especificado. Si no se especifica ninguna codificación ni formato, por defecto dd trabajará, tanto de entrada como de salida con los datos bit por bit. Podemos, solo con la funcionalidad básica, asomarnos ya al potencial

de esta herramienta.

dd if=origen of=destino ...

Su funcionamiento es muy sencillo, pero como seguro nos hemos dado cuenta su sintaxis no es similar a la de los otros comandos que nos hemos encontrado hasta ahora. Esto es porque pese a ser un comando que encontraremos en todos o en la mayoría de sistemas Linux, dd ha heredado su sintaxis de un antiguo lenguaje IBM.

Vamos a ver que opciones tenemos:

- **if=<path>** Especifica el fichero, recordemos que un dispositivo es un fichero, de entrada. Es decir, de donde leeremos
- **of=<path>** Especifica el fichero de salida, Es decir, donde escribiremos.
- **bs=<número>** Especifica el tamaño del bloque en que se leerá y se escribirá. Entendemos por tamaño de bloque, cuantos bytes serán leídos y escritos “simultáneamente”. La combinación de las dos siguientes opciones es incompatible con la actual, sin embargo permiten un mayor nivel de detalle.
- **ibs=<número>** Especifica el tamaño de bloque que se leerá.
- **obs=<número>** Especifica el tamaño de bloque que se escribirá.
- **count=<número>** Especifica el máximo número de bloques que deben ser copiados.

- **skip=<número>** Se salta los bytes especificados del fichero de entrada antes de empezar a leer.
- **seek=<número>** Se salta los bytes especificados del fichero de salida antes de empezar a escribir.

dd tiene bastantes más opciones. Estas se pueden encontrar fácilmente en Internet o en el manual. Además, la combinación de dd con otros comandos resulta en una combinación muy potente y extremadamente útil. Como hemos visto en el apartado 4.1 Tablas de Particiones y particiones, nos puede permitir clonar la tabla de particiones de un disco, cambiar atributos de las tablas de particiones u otras muchas cosas.

Todos los valores que se han mostrado como numéricos en las opciones anteriores pueden ir acompañados por las siguientes letras que aplican un modificador sobre su valor. Para ello la letra debe ir inmediatamente después de los números, sin espacio alguno que los separe.

- **w:** 2 bytes. Multiplicará por 2 el valor que pongamos.
- **K:** 1024 bytes. O lo que es lo mismo, el valor que hayamos puesto representará kilobytes en lugar de bytes.
- **M:** 1024·K bytes. Es decir, el número expresado serán MB.
- **G:** 1024·M. Es decir, el número expresará GB.
- **T:** 1024·G. Es decir, el número expresará TB.

Estos son los más comunes y los que se aceptan en la mayoría

de sistemas Linux, aunque esto dependerá de la versión de dd que estemos usando y de lo actual que esta sea.

4.3 La partición Swap o espacio de swap

En los sistemas Linux es habitual contar con una partición destinada a swap o con un espacio destinado a ello, un fichero por ejemplo. Este espacio junto con la memoria RAM de la máquina conforma la memoria virtual del sistema. La partición o espacio de swap viene a ser, para entenderlo de una forma sencilla, un espacio en disco que actúa como memoria RAM cuando es necesario descargar la memoria RAM o cuando esta se encuentra completamente llena. Debemos tener en cuenta, sin embargo, que el acceso a la swap es más costoso en términos de tiempo que el acceso a la memoria RAM.

La manera en la que nuestro sistema trabaja con el espacio de swap es configurable, como la mayoría de cosas en los sistemas Linux. Aunque en este caso, la configuración que podemos llevar a cabo es menor puesto que, a fin de cuentas, será el kernel quien gestione este espacio. Para realizar la configuración disponemos del fichero `/proc/sys/vm/overcommit_memory`. Este fichero contiene un único número en su interior que puede tomar los valores 0, 1 ó 2.

- **0:** Deja en manos del Kernel valorar cuanta RAM y SWAP tiene disponibles cuando un programa solicita memoria y permite que el kernel deniegue las solicitudes de memoria cuando lo considere oportuno.
- **1:** Todas las peticiones de solicitud de memoria se responderán satisfactoriamente. Este comportamiento puede llevar a que el sistema se quede sin memoria y se vea obligado a terminar algunos programas. Se

terminarán programas antes de denegar una solicitud de memoria.

- **2:** Teóricamente sigue el mismo comportamiento que 0 con la diferencia que en lugar de ser el kernel quien decide si asignar memoria, pese a no disponer de ella, es adecuado o no. Nunca se asignará memoria si no se tiene suficiente. Podríamos decir que es un comportamiento como en 0 pero todavía más extremo.

4.3.1 Usar o no espacio de swap

En los sistemas Linux suele ser habitual contar con el espacio de SWAP, de hecho si permitimos durante la instalación de un sistema Linux que este particione nuestro disco de forma automática, casi seguro que terminaremos teniendo una partición de swap en nuestro sistema. Sin embargo, puede que no siempre nos convenga disponer de una partición de swap.

En algunas circunstancias, como por ejemplo si queremos poder hibernar la máquina, no nos queda otro remedio que tener un espacio de swap. Si no disponemos de él, nuestro sistema Linux no nos permitirá hibernar el ordenador. Esto es debido a que al hibernar el sistema se hace un volcado de la memoria RAM en uso para poder recuperar el estado de la máquina más tarde. Este volcado suele hacerse a una zona de memoria permanente, no tendría sentido mantenerlo en RAM puesto que se trata de memoria volátil y pierde la información si deja de recibir corriente eléctrica.

Tener un espacio de swap puede resultar también útil si se trabaja con el sistema de ficheros tmpfs. Más adelante entraremos en detalle acerca de los distintos sistemas de ficheros. Tmpfs básicamente

permite guardar los datos en memoria RAM en lugar de hacerlo en disco. En este caso no es imprescindible contar con una partición o espacio de swap. Pero si se hace un uso intensivo y mal controlado del espacio de ficheros en memoria podemos llegar a agotarla. Es aquí donde podemos encontrar alguna utilidad a la Swap.

Por último puede llegar a ser interesante tener un espacio de swap para disponer de mayor memoria virtual o para servidores de ficheros en que los procesos de compresión, descompresión y copia de ficheros pueden llegar a ser intensivos y sobre ficheros grandes. En estos casos, si no se dispone de mucha memoria RAM en la máquina, podría llegar a resultar interesante contar con un espacio de SWAP.

¿Que se debe discutir entonces si parece que todo son ventajas? La cuestión se plantea porque no todo son ventajas. La partición de swap no se redimensiona dinámicamente, por lo tanto estamos usando un espacio de disco que podría ser útil para otras cosas. Aunque hoy en día este no es un motivo de peso puesto que los discos son bastante baratos. Encontramos, además, otros motivos como que es posible que suponga un menor rendimiento para el sistema. El espacio de memoria es un espacio con unos tiempos de respuesta de lectura y escritura muy bajos, cosa que se refleja positivamente en la velocidad del sistema. Si nosotros complementamos eso con un disco duro a 5600rpmes muy posible que el acceso a la SWAP, más que una ayuda, se convierta en un cuello de botella.

Finalmente, el uso de SWAP puede aumentar el desgaste del disco duro. Puede parecer absurdo en los discos magnéticos, aunque también se desgastan. Vemos el problema más rápidamente si pensamos en los SSD que tienen un número de escrituras limitadas. Además los discos SSD, aunque esto está mejorando mucho

últimamente, suelen bajar su rendimiento a medida que se van agotando sus ciclos de escritura.

En cualquier caso la decisión de tener o no una partición e SWAP depende de cada uno, cabe plantearse si vale la pena tenerla o si realmente estamos trabajando de manera que nos va a ser de utilidad.

4.3.2 Usar ficheros o particiones como espacio de swap

Es posible usar una partición o un fichero como SWAP y aumentar así la memoria virtual de la que dispone nuestro sistema. Decidir si queremos usar este tipo de “expansión” depende de diversos factores que debemos valorar. En cualquier caso, en los sistemas Linux tenemos a nuestra disposición los comandos *swapon* y *swapoff* que nos permiten habilitar una partición o fichero como swap o deshabilitarla como tal según nos convenga. Antes de poder registrar un espacio de swap en el kernel deberemos crearlo.

swapon y *swapoff* nos ofrece multitud de opciones que son sencillas de encontrar si realizamos una sencilla búsqueda por Internet o si recurrimos al manual: *man swapon*. Comentamos aquí dos opciones que resultan muy interesantes. Sobre el funcionamiento de */etc/fstab*, que se menciona en las opciones, hablaremos más adelante, sin embargo por ahora es interesante conocer que es un fichero de configuración del sistema que permite, entre otras cosas, asociar sistemas de ficheros a determinados dispositivos y particiones.

- **-a:** Permite habilitar o deshabilitar todos los espacios de SWAP que se encuentren en */etc/fstab*
- **-e:** Se salta aquellos ficheros que no existen sin dar

por ello un error o advertencia. Puede parecer realmente absurdo tener en el fichero `/etc/fstab` una partición que no existe para intentar usarla como SWAP, sin embargo, pensemos en el siguiente caso: supongamos que tenemos un disco duro extraíble o un pendrive y que queremos usar alguna de sus particiones como espacio de SWAP. Que pasaría si alguien desconectase el dispositivo plug and play e intentásemos habilitar todas las zonas de SWAP? Obtendríamos un error. Sin embargo, si especificamos esta opción trabajaremos únicamente sobre aquellas que estén disponibles en el sistema en el momento de ejecutar el comando.

4.3.2.1 Particiones

Si se trata de una partición, lo primero que deberemos hacer es asegurarnos de que la partición está vacía, podemos usar, por ejemplo, la herramienta `dd` para borrar el contenido de una partición antes de usarla como una partición de swap. Una vez hecho esto, si nuestra partición es `/dev/sdb1` deberemos crear el espacio de swap y luego registrarlo en el kernel para que esta pueda usar la partición como espacio de swap. Haríamos lo siguiente:

```
dd if=/dev/zero of=/dev/sdb1 bs=512 #Borramos la partición con dd
mkswap /dev/sdb1 #Creamos el espacio de SWAP
swapon /dev/sdb1 #Habilitamos el uso de la partición como SWAP
```

4.3.2.2 Ficheros

En el caso de un fichero, antes de poder crear el espacio de SWAP deberemos crear un fichero del tamaño que nos interese. En este caso lo más sencillo es usar de nuevo la herramienta dd, por ejemplo:

```
dd if=/dev/zero of=ficheroSwap bs=1024k count=1024
```

En el caso del ejemplo, estaríamos creando un fichero vacío de 1MB para usarlo como swap. Hoy en día este tamaño resulta ridículo, sin embargo es trabajo de cada uno adaptarlo a sus necesidades. Una vez disponemos del fichero creado con el tamaño que más nos guste, el procedimiento es el mismo que en el caso de las particiones.

```
mkswap ficheroSwap
```

```
swapon ficheroSwap
```

4.4 Creando particiones y una tabla de particiones

En la sección 4.1, hemos visto que son las tablas de particiones y las particiones y como trabaja con ellas nuestro sistema. Hemos visto que al final se reduce a información muy concreta que se guarda al inicio o al final de nuestros discos y que ello nos permite trabajar más tarde con estos dispositivos. Vamos a ver ahora como crear estas particiones de una forma sencilla. Si tuviésemos que crear las particiones a nivel de bytes y offsets como se ha explicado, deberíamos ser verdaderos expertos en la gestión de discos para poder usarlos. Afortunadamente, hay distintas herramientas que permiten trabajar con los discos, entre las más conocidas encontramos:

- parted

- gparted
- fdisk

Cualquiera de ellas es igualmente válida. Funcionan además de manera similar, eso si, cada una con sus características, en este caso nos vamos a centrar en fdisk. No por que sea mejor o peor, simplemente por una decisión personal.

Antes de ver como trabajar con fdisk, debemos aclarar que los sistemas Linux requieren al menos una partición. Esta partición es donde se sitúa la raíz del sistema, es decir /. Si creamos una única partición, esta contendrá además el espacio de swap y el espacio de usuario. Suele ser habitual en los sistemas Linux tener al menos dos particiones, una para la raíz del sistema y otra para el espacio de SWAP. Si se decide no trabajar con swap, igualmente suele ser habitual tener una partición para la raíz del sistema y otra para los usuarios. Es decir, una partición para / y otra para /home. Algunas distribuciones ofrecen la funcionalidad de particionar el disco automáticamente en el momento de la instalación. En estos casos suele ser habitual tener al menos una partición de swap y otra para la raíz y se ofrece al usuario la posibilidad de crear una tercera para la /home.

Después de este pequeño inciso, lo primero que deberemos hacer es decidir que particiones queremos o necesitamos para nuestro sistema. Antes de ello si queremos listar las particiones y tablas de particiones ya existentes, podemos invocar a fdisk con la opción -l. Para usar el comando fdisk, necesitaremos, de un sistema Linux ya instalado o emplear alguna de las alternativas en formato Live CD, como por ejemplo el Live CD de gparted.

Aclaración

Un live CD es un sistema, Linux en el caso que nos ocupa, que permite arrancar todo el sistema desde el CD sin necesidad de instalarlo. Este tipo de sistemas, contienen todos los componentes necesarios y suelen cargar los distintos módulos y sistemas de ficheros en memoria RAM.

`fdisk -l`

Disk /dev/sda: 320.1 GB, 320072933376 bytes

255 heads, 63 sectors/track, 38913 cylinders, total 625142448 sectors

Units = sectors of 1 * 512 = 512 bytes

Sector size (logical/physical): 512 bytes / 512 bytes

I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk identifier: 0x00077068

Device	Boot	Start	End	Blocks	Id	System
/dev/sda1	*	2048	499711	248832	83	Linux
/dev/sda2		501758	625141759	312320001	5	Extended
/dev/sda5		501760	625141759	312320000	83	Linux

Toda la información mostrada debería sonarnos, si no es así podemos echar un vistazo a la sección 4.1 Tabla de Particiones y particiones, puesto que los títulos que nos ofrece la salida de `fdisk` son bastante explicativos.

Una vez conocemos la información de lo que tenemos en el ordenador lo primero que debemos hacer, como siempre que queramos trabajar con discos, es identificar con que dispositivo vamos a trabajar. A continuación deberemos invocar a `fdisk` pasándole como parámetro el disco con el que vamos a trabajar.

`fdisk /dev/sda`

Después de ejecutar el comando, `fdisk` quedará a la espera de que le digamos que queremos hacer. Estas son algunas de las opciones que nos ofrece la aplicación:

- **a:** Permite indicar que una partición será bootable o, lo que es lo mismo, que se podrá arrancar desde ella.
- **d:** Elimina una partición
- **l:** Lista los tipos de particiones con los que podemos trabajar
- **m:** Muestra todas las opciones disponibles para `fdisk`
- **n:** Crea una nueva partición
- **o:** Crea una nueva tabla de particiones MBR
- **p:** Muestra por pantalla la tabla de particiones según la tenemos
- **q:** Sale de `fdisk` sin guardar los cambios
- **v:** Verifica que la tabla de particiones sea correcta
- **w:** Guarda los cambios y sale de `fdisk`

Como vemos, si dudamos, solo debemos apretar 'm' para obtener una lista de todas las acciones disponibles. La totalidad de acciones disponibles es mayor que la mostrada aquí. Con esta información tenemos lo necesario para particionar el disco según conveniencia. Hay que tener en cuenta que al seleccionar cada una

de las opciones el propio programa nos preguntará por toda la información necesaria para completar los datos de cada partición según se veía en la sección 4.1.

Advertencia

Es importante tener en cuenta que modificar la tabla de particiones no es complicado, sin embargo, es más que probable que al modificar la tabla de particiones seamos incapaces de recuperar los datos que contenía dicha partición. También se debe recordar que las tareas que conciernen a particionar discos o trabajar con sus sistemas de ficheros no pueden ser realizadas por un usuario que no disponga de permisos de administración en el sistema.

4.5 Sistemas de ficheros

Ahora que ya disponemos de la tabla de particiones y de las particiones en si mismas lo único que nos separa de poder usar el disco es el sistema de ficheros. Técnicamente podemos trabajar con el disco una vez particionado con herramientas como dd. Es una opción, pero trabajando sin sistema de ficheros perderemos multitud de herramientas interesantes como cd o ls. El sistema de ficheros es el encargado de manejar el espacio de disco disponible según la jerarquía de directorios que ya hemos visto en el capítulo 2. Ficheros y jerarquía de directorios. Se encarga, además, de proveer otras funcionalidades, como las herramientas para crear, mover, copiar, o destruir ficheros entre otras. El sistema de ficheros es también el encargado de indicar a que fichero pertenece cada bloque o si estos bloques están o no ocupados. Tradicionalmente, la gestión de los sistemas de ficheros es manejada por el kernel, pero en sistemas

modernos es posible encontrar sistemas de ficheros en el espacio de usuario.

Podríamos decir que el sistema de ficheros es una manera estructurada de almacenar la información que corresponde a los datos que almacenaremos en nuestros dispositivos. Entre otras funcionalidades, los sistemas de ficheros se encargan de proporcionar:

- Seguridad.
- Mecanismos para evitar la fragmentación.
- Soporte para cuotas de disco.
- Soporte para la gestión de ficheros.
- Integridad de los datos.
- Y bastantes funcionalidades adicionales

Los sistemas de ficheros dependen en gran medida del sistema operativo, así pues, es posible que un MAC no comprenda un sistema de ficheros creado por un Windows, de la misma manera que es posible que un Windows no comprenda un sistema de ficheros creado por un Linux. Aunque si estamos dispuestos a trabajar únicamente con sistemas Linux podemos decantarnos por uno de los sistemas especialmente diseñado para sistemas Linux, nuestra elección del tipo de sistema de ficheros dependerá no solo de que máquina estamos usando si no de que máquinas podemos llegar a usar. Por ejemplo, darle a un pendrive un sistema de ficheros como ext4 que es específico de sistemas Linux puede hacer que no podamos leerlo en sistemas MAC o Windows. Debemos ser cuidadosos pues con las elecciones que conciernen al sistema de ficheros. Algunos de los sistemas de ficheros más conocidos son:

- **ext4:** Es un sistema de ficheros nativo para sistemas Linux que trabaja con un mecanismo de journaling, más adelante veremos lo que es, para garantizar la integridad de los datos. ext4 es una evolución de otro sistema muy importante para los sistemas Linux: ext3. Además este sistema ofrece compatibilidad hacia atrás, permitiendo que sistemas ext3 y ext2 sean montados como si se tratase de sistemas de ficheros ext4.
- **ext3:** Otro sistema de ficheros muy importante para los sistemas Linux, igual que el anterior, cuenta con un mecanismo de Journaling para garantizar la integridad de los datos. Adiferencia de su sucesor, sin embargo, ofrece soporte para ficheros más pequeños y para un número menor de subdirectorios. Igual que su sucesor, ofrece también compatibilidad hacia atrás. Este sistema de ficheros resulta importante porque a día de hoy, aunque la mayoría de kernels usan el sistema de ficheros ext4, el sistema ext3 sigue estando extendido.
- **FAT:** El sistema de ficheros por excelencia de Windows. Han ido apareciendo mejoras de este sistema de ficheros para dar soporte a ficheros más grandes, como es el caso de extFAT. Algunas mejoras de este sistema como vFAT son soportadas por la mayoría de sistemas Linux al mismo tiempo que son soportadas por los sistemas Windows. vFat se convierte entonces en un buen candidato para usar

tanto en máquinas Windows como Linux.

- **NTFS:** Un sistema de ficheros para Windows que también puede ser montado en Linux. NTFS es un sistema de ficheros basado en meta datos que permite trabajar con ficheros muy grandes, de hasta 16TB. Sin embargo por la cantidad de información que guarda NTFS no se recomienda trabajar con este sistema de ficheros con discos menores a 2GB. Esto hoy en día no supone un problema pero evidencia algunas de las limitaciones de este sistema de ficheros. Al igual que los sistemas nativos de Linux, ext3 y ext4, NTFS cuenta con un mecanismo de Journaling para mantener la integridad de los datos.

4.6 Inodos y Ficheros

Hemos hablado con anterioridad, en la sección 2.1 Ficheros y directorios, de lo que es un fichero. Ahora ya sabemos que nuestro disco, nuestra unidad de almacenamiento, se divide en particiones que se gestionan por una tabla de particiones y que dentro de estas particiones encontramos ficheros que son gestionados por un sistema de ficheros. Ahora con más razón que antes, podremos decir que un fichero es una unidad lógica de almacenamiento que agrupa información relacionada entre si bajo el mismo nombre. Eso nos lleva a que para tener un fichero necesitaremos, por lo menos un nombre, y necesitaremos guardar también otro tipo de información. La estructura del sistema de ficheros que nos ayuda a almacenar la información referente a los ficheros es el *inode* o nodo-i en castellano. Aunque hay que apuntar que el nombre del fichero no se encuentra en la

estructura del inode.

El *inode* no es más que una estructura de datos que nos permite representar de una forma ordenada la información relativa a un fichero. Este tipo de estructura se usa únicamente en los sistemas Linux o más generalmente en los sistemas Unix y nos permite almacenar junto con la información relevante del fichero otros meta datos. La información que nos muestra el comando ls es información que se almacena en los *inodes*. La manera de almacenar los meta datos sobre los ficheros depende, evidentemente, del sistema de ficheros, esto nos conduce a que sistemas de ficheros como NFS no trabajan con inodes.

Vamos a ver como trabajan realmente los *inodes*. Lo primero que debemos saber es que un *inode* puede representar cualquier tipo de fichero. Si se trata de un directorio en su interior contendrá la información de los otros inodes que contiene el directorio, si se trata de un fichero regular contendrá la información referente al fichero, meta datos y la información necesaria para acceder al contenido del fichero. Un inode es una tabla que contiene algo similar a las siguientes entradas:

- Tipo de archivo y protección.
- Número de referencias.
- Propietario.
- Grupo del propietario.
- Tamaño.
- Puntero a bloque de datos 0.
- Puntero de bloque de datos 1.

- ...

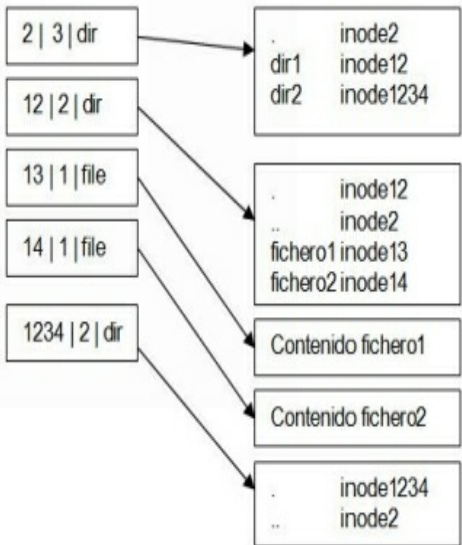
Para entender como funcionan los *inodes* y la estructura jerárquica de directorios, vamos a recurrir a una simplificación de como se trabaja con estas estructuras de datos. Pese a que no contaremos con toda la información que manejan estas estructuras si que nos permitirá hacernos una idea de su funcionamiento.

Identificador del inode	Número de referencias	Tipo de inode
-------------------------	-----------------------	---------------

En seguida veremos un ejemplo de como se ve una jerarquía de directorios según inodes en lugar de nombres. Vamos a aclarar, antes de ello, que significan los valores de la tabla anterior.

- **Identificador del inode:** Es un número que representa el inode. Cada inode tiene un identificador que es representado por un número en base decimal.
- **Número de referencias:** La cantidad de nombres de fichero o directorios que están asociados con este inode. Parece un concepto complicado, en el ejemplo sin embargo quedará más claro. Aún así hay que pensar que el nombre del fichero o directorio no se almacena en la propia estructura del inodo. Un mismo inode puede estar asociado a diversos nombres de fichero o directorio. Así suele ser habitual que el inodo de un directorio tenga al menos 2 referencias.
- **Tipo de inode:** Nos indica con que tipo de fichero está asociado el inode. Suele tratarse de ficheros

regulares o directorios.



Ahora que ya sabemos que significa cada elemento de los que hemos presentado vamos a ver un ejemplo de como trabajar con los inodes. Veremos un ejemplo teórico y a continuación veremos un ejemplo práctico usando algunos comandos de Linux que ya nos sonaran.

Lo primero que observamos es que el primer inode tiene 3

referencias. El primer inode equivale al primer cuadro de la columna de la izquierda. Si nos fijamos detalladamente, veremos en la columna de la derecha que, efectivamente, el inode2 aparece nombrado 3 veces. Si nos damos cuenta, esto significará que un directorio que no sea el directorio raíz tendrá forzosamente un mínimo de 2 referencias. Una del directorio en que se puede encontrar. La otra vendrá dada por él mismo. Supongamos que queremos abrir el fichero2. Los pasos a seguir serían los siguientes:

- Identificamos los componentes de la ruta a la que queremos acceder dir1/fichero1. Es decir un directorio y un fichero.
- Empezamos la búsqueda de dir1 en el directorio en que nos encontramos, no empieza por '/', se trata por tanto de una ruta relativa. En este caso vamos a la parte que contiene los datos del inode 2 y encontramos que dir1 pertenece al inode 12.
- Examinamos el inode12 y vemos que realmente se trata de un directorio.
- Examinamos el contenido del inode 12 y localizamos el fichero2 que está asociado con el inode14
- Vamos al inode 14 y comprobamos que se trata de un inode vinculado a un fichero regular.
- Accedemos al contenido del fichero.

Todo este trabajo lo hace el Kernel, no debemos hacerlo nosotros manualmente, ya tenemos suficiente con recordar rutas y configuraciones. Aún así, podemos ver como funciona. También nos

ayuda a darnos cuenta de un detalle. Cuando ejecutamos el comando *rm* que nos permite borrar un fichero, realmente no estamos eliminando el fichero. Es decir el inode no está siendo borrado. Lo que se está haciendo es desvincular el nombre de fichero del inode que le corresponde. Después, se disminuye la cuenta de referencias del inode en 1. Si un inode en algún momento tiene 0 referencias, es eliminado. Este es el motivo, por el que los programadores, aquellos que programen en C por ejemplo, disponen de la función *unlink* además de la función *remove*.

¿Como podemos ver todo esto en una forma práctica?
¿Existen de verdad los inodes? La manera más sencilla de comprobarlo es con el uso del comando *ls*. De nuevo recurrimos a este comando pero esta vez le pasaremos la opción *-li* para que nos muestre los inodes.

```
ls -li  
12 dir1 1234 dir2
```

Los números delante de *dir1* y *dir2* nos indican el inode que corresponde a esos directorios. El comando *ls* nos ha proporcionado la información referente a los inodes, necesitaremos recurrir a otros comandos para poder hacer uso de esta información. Supongamos que queremos eliminar un fichero en nuestro ordenador donde el nombre contiene un conjunto de caracteres que somos incapaces de escribir.

```
ls -la  
ad????.txt
```

En este caso podemos recurrir a los inodes para borrar el

fichero.

ls -i

18 ad???.txt

find . -inum 18 -delete

El comando `find` está buscando en el directorio '.', es decir el actual, el fichero con inode 18 para borrarlo.

Para terminar con el tema de los *inodes*, seguro que una de las cosas que nos viene a la mente es: ¿cómo puede el Kernel saber que bloques están libres para asignarlos a un fichero? En este caso una de las maneras más sencillas de mantener el control sobre los bloques libres es mantener un mapa de bits para los bloques. Si un bloque tiene asignado un uno (1) es porque está en uso. Si un bloque tiene asignado un cero (0) no está en uso y puede ser asignado, por ejemplo, cuando creamos un nuevo fichero. La manera de tratar así los datos implica que al borrar un inode, simplemente se pone a 0 el bit que indicaba que ese bloque estaba ocupado. Si reflexionamos sobre ello, llegamos a la conclusión que la información no se ha borrado completamente del disco, si no que simplemente el kernel la ha considerado borrada. Es por esto que muchos programas de informática forense son capaces de recuperar datos que han sido borrados de nuestro ordenador.

Juntando conceptos

¿Cómo podemos borrar entonces los datos de forma segura? Seguro que si le damos un par de vueltas a la herramienta `dd` y releemos la sección 4.3.2.1 se nos ocurre una solución al problema de borrar los datos de una forma segura empleando el comando

dd. El uso de este comando conjuntamente con el dispositivo /dev/zero nos provee de lo necesario para borrar un fichero o dispositivo de forma segura.

4.7 Journaling

Una de las cosas más importantes para cualquier sistema de almacenamiento de datos, ya sea una base de datos, un sistema de ficheros o cualquier otro sistema, es mantener la integridad de los datos. Es decir, asegurar que los datos que hay almacenados siguen siendo los datos que guardamos.

Para ello los sistemas de ficheros nativos de Linux usan un mecanismo basado en transacciones. Una transacción es una interacción con una estructura de datos compleja en la que todos los procesos que la componen deben realizarse de una sola vez. La idea del Journaling es llevar un registro detallado de todas las operaciones que se realizan con las estructuras de datos para asegurar que si la transacción falla en un momento dado podrá recuperarse el estado anterior al fallo. Recordemos que los sistemas Linux, al igual que la gran mayoría de sistemas modernos, son sistemas multiusuario y multitarea. Esto afectará también a como tratar la integridad de los datos, puesto que mientras realizamos una transacción debemos impedir que ninguna otra tarea o usuario pueda alcanzar los datos y comprometer su integridad.

Pongamos un ejemplo: supongamos que dos usuarios quieren escribir su nombre en el fichero "nombre.txt". Este fichero tiene como objetivo tener el nombre de todos los usuarios del sistema, a razón de uno por línea. ¿Que sucede si el usuario Alberto y el usuario Benito abren el fichero al mismo tiempo e intentan escribir su nombre?

Si no tenemos algún mecanismo de control lo más probable es que acabe pasando algo similar a lo siguiente:

cat nombre.txt

BenAlibetrtoo

Por suerte nuestros sistemas implementan mecanismos para evitar estos problemas.

Advertencia

Si programamos a bajo nivel deberemos ser nosotros los encargados de evitar estas problemáticas en lugar del sistema.

En el caso de los mecanismos de journaling sobre los sistemas de ficheros únicamente se aplica el registro del sistema de transacciones sobre:

- Inodes.
- Estructuras de directorios.
- Bloques libres de disco o mapas de bits de bloques.

Hay otros mecanismos encargados de mantener la integridad de los datos, normalmente ofrecidos por los propios comandos o programas o por llamadas al sistema. Más allá del control en los comandos, las llamadas al sistema y los programas, mantener el sistema de ficheros integro es imprescindible. El sistema de Journaling es el encargado de garantizar la integridad del sistema de ficheros. Su funcionamiento es más o menos el siguiente:

- Se bloquean las estructuras de datos implicadas. Normalmente algunas o las tres mencionadas

anteriormente.

- Se habilita un espacio para guardar el journal o diario de las transacciones que se realizarán.
- Se anota en el journal la modificación a realizar y como deshacerla, por si hubiese un error y fuese necesario.
- Se realiza la modificación.
- Se borra el journal.
- Se desbloquean las estructuras implicadas.

Si en algún momento se quieren deshacer las transacciones, antes de borrar el journal se pueden deshacer una a una siguiendo el orden. Si al arrancar el sistema se detecta que una transacción quedó pendiente, es decir, si el journal no ha sido borrado, únicamente se deben deshacer todas las modificaciones una a una para volver a tener el sistema de ficheros en un estado coherente y consistente.

4.8 Creando un sistema de ficheros

El comando `mkfs` es el que nos va a permitir crear un sistema de ficheros. Para trabajar con este comando y según las cosas que ya hemos visto, antes de crear un sistema de ficheros deberemos saber, en que partición queremos crear el sistema de ficheros y que tipo de sistema de ficheros queremos. Por ejemplo:

```
mkfs -t ext4 /dev/sdb1
```

El comando anterior permite crear un sistema de ficheros (make filesystem) de tipo `ext4` en `/dev/sdb1`. `mkfs`, es un comando que

nos sirve únicamente como “cara bonita” de una serie de programas que son los que realmente van a crear el sistema de ficheros. Así pues, la llamada anterior es equivalente a:

```
mkfs.ext4 /dev/sdb1
```

Esto lleva a que sea relativamente sencillo conocer cuales son los tipos de sistemas de ficheros que podremos crear desde nuestro sistema. Para ello, nos será suficiente con ejecutar:

```
ls -l /sbin/ | grep mkfs
```

para ver una lista de todas las utilidades mkfs que tenemos en nuestro sistema. Si nos fijamos en la salida del comando anterior podremos apreciar que los distintos programas mkfs son links simbólicos a otros programas. Esto nos lleva a dos conclusiones. La primera de ellas, que si no podemos crear un sistema de ficheros concreto es posible que podamos instalar un nuevo programa que nos lo permita. La segunda, que si necesitamos buscar documentación de mkfs.ext4 no buscaremos documentación de ese comando, aunque podemos hacerlo, si no del programa al que apunta el link simbólico, muy probablemente mke2fs.

Hay que tener en cuenta que solo es necesario crear el sistema de ficheros una vez por cada partición. Crear un sistema de ficheros en una partición que ya dispone de uno supone la destrucción virtual de todos los datos de la partición. Destrucción virtual porque los datos seguirán estando almacenados en los discos, sin embargo su recuperación será muy difícil o imposible debido a la nueva estructuración de inodes, mapas de bits para bloques y otras re-estructuraciones.

Aclaración

Los tipos de sistemas de ficheros son conocidos, coloquialmente como formatos. La acción de dar formato a un disco o dispositivo de almacenamiento se conoce como formatear. Al formatear un disco o USB en Windows o MAC realmente se está creando un nuevo sistema de ficheros, no borrando los datos del disco. Los datos del disco son irrecuperables por la reestructuración que se comenta más arriba. Esto también nos dice que si nos piden formatear un disco en Linux deberemos recurrir a los comandos mkfs.

4.8.1 Comprobar un sistema de ficheros

Igual que disponemos del comando mkfs (*make filesystem*) para crear un sistema de ficheros en una partición, disponemos de un comando que nos permite testear la integridad de un sistema de ficheros. Normalmente los errores en el sistema de ficheros suelen darse por un mal apagado del ordenador, aunque pueden suceder por otros motivos. El comando fsck (*filesystem Check*) nos permite comprobar estos errores y nos da la opción de recuperarnos de algunos de ellos.

```
fsck /dev/sdb1
```

Igual que en el caso de mkfs, fsck es simplemente un programa que llama a otros programas, cada uno distinto según el sistema de ficheros de la partición que le pasemos como parámetro. A diferencia de mkfs, en este caso el comando detectará por él mismo el sistema de ficheros, por lo que no deberemos pasárselo como parámetro.

Advertencia

Comprobar los errores de una partición que se encuentre montada, más adelante veremos que significa esto, puede resultar fatal para los datos. Cualquier cambio en los datos o en el sistema de ficheros mientras se realiza la comprobación y podemos acabar con datos desaparecidos o con un sistema de ficheros inservible.

4.9 Montando un sistema de ficheros

Montar un sistema de ficheros no es otra cosa que asociar un sistema de ficheros a un dispositivo para permitir que el sistema sea capaz de trabajar con él. Se trata de decirle al sistema que, por ejemplo, vamos a encontrar un sistema de ficheros ext4 para el dispositivo `/dev/sdb1` en el punto de montaje `/disco`. El punto de montaje será el punto de acceso que tendremos nosotros a la partición con ese sistema de ficheros. Podemos hacer esto con el comando *mount*.

```
mount -t ext4 /dev/sdb1 /disco
```

```
cd /disco #Podremos explorar el contenido de la partición /dev/sdb1
```

El punto de montaje debe existir antes de ejecutar la instrucción. De la misma manera, aunque se puede ejecutar la instrucción desde el espacio de usuario, en la mayoría de ocasiones, es necesario contar con permisos de administrador para poder montar un sistema de ficheros. Usualmente, el tipo del sistema de ficheros no es necesario especificarlo, puesto que el propio programa se encargará de detectarlo.

Si invocamos el comando *mount* sin pasarle ningún

parámetro, en lugar de asociar un sistema de ficheros con un dispositivo y un punto de montaje, vamos a poder ver una lista de todas las particiones montadas en nuestro sistema, junto con el punto de montaje, el tipo de sistema de ficheros y las opciones con las que se ha montado dicho sistema.

Las opciones que podemos especificar al sistema de ficheros afectan directamente a la forma en que podemos trabajar con él. Podremos especificar estas opciones con la opción `-o` y pasándole una lista de opciones separadas por comas. Por ejemplo:

```
mount -t ext4 /dev/sdb1 /disco -o noexec,ro
```

Algunas de las opciones más comunes que podemos emplear son:

- **exec:** Permite la ejecución de programas en el sistema de ficheros.
- **noexec:** No permite la ejecución de programas en el sistema de ficheros.
- **suid:** Si recordamos la sección 2.3 Permisos y propiedades de ficheros, se hablaba de añadir el bit SUID. Esta opción permite que se ejecuten programas con aquel flag.
- **nosuid:** Evita que se ejecuten programas con el flag SUID.
- **ro:** Monta el sistema de ficheros en modo solo lectura.
- **rw:** Monta el sistema de ficheros en modo lectura-escritura.

Además *mount* tiene una opción muy interesante. La opción `-a` permite montar todos los sistemas de ficheros conocidos por el sistema. Pero, ¿cómo conoce los sistemas de ficheros nuestro Linux? De hecho, ¿cómo sabe al arrancar el sistema, que sistemas de ficheros debe de montar y donde?. Esto es posible gracias a un fichero de configuración: el fichero `fstab`. Como todo fichero de configuración del sistema lo encontraremos en el directorio `/etc`. Estamos hablando por tanto del fichero `/etc/fstab`

4.9.1 El fichero `/etc/fstab`

El fichero `/etc/fstab` es el que le permite al sistema saber que sistemas de ficheros debe asignar a que particiones y con que punto de montaje. Es este fichero al que el sistema recurre al arrancar para que todas nuestras particiones y todos nuestros ficheros estén disponibles. Es, por tanto, a este fichero donde deberemos ir si queremos que nuestro sistema de ficheros se monte al iniciar el sistema o si queremos poder montarlo con la opción `-a`. En otras palabras, es donde deberemos añadir cada sistema de ficheros que queremos que sea conocido por el SO.

El fichero `/etc/fstab` es un fichero de texto plano con un formato bastante sencillo de recordar.

```
cat /etc/fstab
```

```
[...]
```

```
/dev/sda1 /ext4 defaults 0 0
```

El fichero tiene una línea por cada sistema de ficheros a montar. Cada fila tiene 6 columnas. Por columnas el contenido de este fichero es como sigue:

- **Dispositivo o UUID:** El dispositivo o el identificador

único del dispositivo.

- **El punto de montaje:** Desde donde el usuario accederá al contenido del dispositivo.
- **El tipo del sistema de ficheros.**
- **Las opciones:** Estas opciones son las mismas que en el caso del comando mount. Veremos, sin embargo, que en este caso puede tomar algunos valores que no hemos comentado antes. Las opciones deben introducirse en una lista separada por comas.
- **Backup para el dump:** Es usado por la herramienta dump para saber si debe hacer una copia de seguridad o no del sistema de ficheros. Si toma el valor uno, hará una copia de seguridad, si toma el valor 0 no hará copia de seguridad. Si no se tiene la utilidad dump instalada se debe poner un 0. Lo recomendable, de todas maneras es poner un 0 siempre que se tengan dudas sobre este valor.
- **Orden de comprobación de integridad:** Este campo puede tomar el valor 0,1 ó 2. Establece el orden de prioridad en que deben ser verificados los sistemas de ficheros por la utilidad fsck. El sistema de ficheros montado en la raíz siempre debe tomar el valor 1, el más prioritario. Cualquier otro sistema de ficheros que queramos que sea comprobado debe tomar el valor 2. Si no queremos que un sistema sea comprobado especificaremos el valor 0. Hay que tener en cuenta

que todo lo que tenga un número distinto de 0 será comprobado al inicio del sistema. Esto significa que debemos ser cuidadosos o acabaremos tardando un tiempo considerable en poder arrancar el sistema.

Una vez conocemos los campos y tenemos el fichero a nuestro gusto, seremos capaces de ejecutar la opción `mount -a` para montar todos los sistemas de ficheros conocidos. No solo eso si no que además podremos usar el comando `mount` sin especificar un dispositivo siempre y cuando el punto de montaje se encuentre en el fichero `/etc/fstab`.

Las opciones del sistema de ficheros en `/etc/fstab`, además de las opciones del comando `mount`, pueden tomar algunos valores que pueden resultar muy útiles, veamos algunos de ellos:

- **auto:** El sistema de ficheros será montado automáticamente al inicio del sistema o con el comando `mount -a`.
- **noauto:** No se montara al iniciar el sistema ni al ejecutar `mount -a`
- **sync:** La escritura en disco debe ser síncrona. Es el sistema operativo el que decide siempre cuando escribir en disco, es posible que al hacer una escritura el sistema la demore hasta que considere oportuno, quizás por disponer de más datos, realizar la verdadera escritura en disco. La opción `sync` fuerza que cada operación vaya directamente a disco.
- **user:** Permite a cualquier usuario montar el sistema de

ficheros

- **nouser:** Únicamente un usuario con privilegios de administrador puede montar el sistema de ficheros.
- **defaults:** Opciones por defecto, se traduce en las opciones (rw,suid,dev,exec,auto,nouser,async)
- **noatime:** No actualiza las fechas de acceso en los inodes. Puede suponer una mejora de rendimiento. En casos como tarjetas SD suele marcarse esta opción para alargarles la vida útil.
- **nodiratime:** Igual que la opción anterior pero únicamente para directorios

Con esta información tenemos suficientes datos para poder montar nuestros sistemas de ficheros y trabajar con ellos cómodamente. No solo eso, si no que somos capaces de decirle al sistema que monte automáticamente nuestros sistemas de ficheros en el arranque del sistema. Para terminar solo nos falta un matiz. En los sistemas Linux modernos es posible que nos encontremos con un directorio llamado `/etc/fstab.d/`. En estos casos, cualquier fichero que incluyamos en este directorio se tratará como una parte del fichero `fstab` del que ya hemos hablado.

Advertencia

Cuando trabajemos con `fstab` debemos ser cuidadosos. Un error de sintaxis o cualquier otro error en este fichero y es posible que seamos incapaces de montar nuestros sistemas de ficheros o incluso de arrancar el sistema. Por ello es recomendable realizar

una copia de seguridad antes de modificar /etc/fstab.

5. Bash, profundizando

Conocemos unos cuantos comandos para trabajar en los sistemas Linux que nos hacen la vida más cómoda o que nos permiten hacer exactamente aquello que queremos con el sistema en el que trabajamos. Conociendo los comandos, un mejor conocimiento del terminal Bash nos permitirá trabajar de una manera más sencilla.

5.1 History

La mayoría de terminales, los que no son bash también, nos ofrecen la posibilidad de recuperar comandos que ya hemos ejecutado usando la flecha arriba de nuestro teclado. En nuestro caso nos centraremos en el funcionamiento de bash. Encontrar información sobre el terminal que prefiera cada uno, sin embargo no debería ser difícil y en algunos casos podría ser bastante similar a la que aquí se presenta.

Los comandos que el terminal almacena se conocen como historia (history en inglés). Por tanto si en nuestro terminal ejecutamos history obtendremos una lista de los últimos comandos ejecutados por nuestro usuario. History, por defecto, suele almacenar un número limitado de comandos, 500 en las distribuciones Linux basadas en Debian, esto significa que pasado un tiempo los comandos más antiguos dejarán de ser accesibles.

Recuperar un comando que ejecutamos hace 5 días puede ser muy arduo si lo hacemos a base de apretar 300 veces la flecha hacia arriba de nuestro teclado. Es por eso que history nos ofrece una serie de herramientas para poder interactuar de una forma rápida con los comandos ejecutados.

Ctrl+r nos permite buscar un texto en history. Esto significa que podemos buscar un comando que hayamos usado previamente apretando Ctrl+r y luego escribiendo cualquier parte que tuviese el comando que buscamos. ¿Que pasa si la búsqueda que realizamos no nos devuelve el comando que nos interesa pero nosotros sabemos que ejecutamos el comando que estamos buscando? Lo único que tenemos que hacer después de introducir la cadena a buscar es apretar Ctrl+r repetidas veces para ir viendo resultados cada vez más antiguos de nuestra búsqueda.

Además de Ctrl+r tenemos otras maneras de interactuar con el historial de nuestros comandos:

- !comando Ejecutará el último comando que empezase por comando
- ¡?home? Ejecutará de nuevo el último comando que contuviese home
- <espacio>comando Ejecuta comando sin guardar este en el historial de comandos

Si no aparece el comando buscado puede deberse a que history guarda un número limitado de comandos con lo cual es más que probable que llegue un punto en el que empezemos a perder comandos antiguos que hayamos ejecutado. Por defecto, el historial de bash guardará cada comando que ejecutemos. Si ejecutamos 10 veces seguidas el mismo comando guardará 10 veces el mismo comando. No solo eso, si no que si nos equivocamos 10 veces escribiendo el mismo comando guardará 10 veces el mismo comando erróneo. Por suerte, este comportamiento puede cambiarse. Vamos a ver como.

Para aumentar el número de comandos que podemos

almacenar tenemos que recurrir a dos variables que regulan el tamaño de history.

HISTFILESIZE

HISTSIZE

La diferencia entre ambas es que la primera se mantiene entre distintas sesiones, es decir entre distintas sesiones de usuario mientras que la segunda sirve únicamente para la sesión en curso. Para modificar el valor que tienen ejecutaremos:

```
export HISTFILESIZE=<numero de elementos>
```

```
export HISTFILESIZE=1000
```

Con esto decidimos cual va a ser el número máximo de comandos que va a guardar nuestro historial de comando. Si queremos que bash guarde un número indefinido de comandos ejecutados en el historial basta con dejar el número de elementos vacío.

```
export HISTSIZE=
```

Seguimos teniendo el “problema” de que el historial guarda comandos repetidos. Es decir si ejecutamos 'ls' tres veces, guardará 3 veces el comando ocupando 3 de las posiciones máximas que tiene nuestro historial para guardar caracteres. Este puede ser el comportamiento deseado en un sistema multiusuario en el que se quiere tener un control absoluto de los comandos que ha ejecutado cada usuario, también puede no serlo, por ejemplo en sistemas de uso personal. Si queremos cambiar este comportamiento podemos hacerlo mediante HISTCONTROL. En este caso lo haremos como sigue:

```
export HISTCONTROL=ignoredups
```

De esta manera evitaremos que se guarden comandos idénticos en nuestro historial de comandos.

Hay que tener en cuenta que cualquier cambio que se realice sobre el historial con los comandos anteriores no será permanente en el sistema. Una vez reiniciemos la máquina los perderemos. Si queremos que los cambios sean permanentes tendremos que añadir estos comandos tal como se han mostrado aquí a un fichero de configuración. Podemos añadir los cambios a `/etc/profile` o a `/home/usuario/.bashrc`.

Según hemos hablado en Jerarquía de directorios, todos los ficheros de configuración del sistema se encuentran en el directorio `/etc`. Esto nos permite, si le damos algunas vueltas, ver que si añadimos los comandos a `/etc/profile` estos afectaran a todos los usuarios del sistema. Por contra si lo añadimos al fichero `.bashrc` dentro de un directorio de usuario, `/home/usuario`, lo que obtendremos es una configuración que afecta únicamente a los terminales de dicho usuario. En manos de cada uno queda decidir a quien quiere que afecten los cambios que realice sobre el historial.

Cuidado

El caso de `bash` es especial, hay otros programas que tienen sus ficheros de configuración fuera de `/etc`, en `/etc` se encuentran los ficheros de configuración del sistema. Aún y así, es muy habitual encontrar también en `/etc` los ficheros de configuración de la mayoría de servicios.

5.2 Variables especiales del terminal

En muchas ocasiones trabajamos con programas o comandos

de los que nos interesa aprovechar una parte o de los que nos interesa saber que identificador tienen. En Linux cada proceso que se ejecuta tiene un identificador único. Si la frase ha sonado a chino no hay que preocuparse, más adelante hablaremos sobre procesos, que son y que información nos aportan. Por ahora, podemos decir que cada programa que se ejecuta en nuestra máquina es un proceso.

Volviendo al tema del terminal, los identificadores de los procesos pueden ser interesantes para trabajar con ellos. También es posible que nos interese repetir tan solo el último parámetro que le hemos pasado a un comando anterior. Por ejemplo, es habitual que después de crear un nuevo directorio queramos entrar en él. Las variables especiales del terminal van a ser las que nos facilitarán todos estos trabajos en los sistemas Linux. Es importante no confundir estas variables con los comandos guardados en el historial de comandos, estas variables tienen siempre el mismo valor, solo que este valor depende de los comandos recientemente ejecutados.

Bastantes de los valores que toman las variables que vamos a mostrar tienen mucho que ver con procesos. Si no entendemos bien cual es el objetivo de la variable o cual es el valor que toma no debemos preocuparnos. Más adelante, en el capítulo 6, trataremos largo y tendido los procesos en Linux, en ese momento entenderemos exactamente cual es el valor que va a tomar cada una de las variables. Puede ser una buena idea volver atrás en ese momento.

- **\$?** : Valor de retorno del último parámetro ejecutado
- **\$_** : Último parámetro del último comando ejecutado
- **\$!**: Identificador de proceso para el último comando ejecutado en segundo plano

- **!!:** El último comando ejecutado

Además disponemos de variables sin más. Las variables son símbolos que pueden tomar distintos valores. Estas variables nos permitirán trabajar cómodamente con elementos que se repiten a menudo o que cambian solo parcialmente. Vamos a ver como usarlas y a continuación veremos algunos ejemplos de su uso. Asignar valor a una variable es algo tan simple como poner una cadena de caracteres cualquiera considerando que debe empezar por una letra o una barra baja y luego puede tener cualquier secuencia de caracteres alfanuméricos que queramos, seguida de un igual y el valor que queremos darle a la variable.

```
home_path="/home/dani"  
variable=3
```

Es muy importante que el símbolo igual suceda a la cadena que designa la variable **sin espacios en blanco**, de otra manera el terminal no entenderá que estamos asignando una variable.

Una vez le hemos asignado un valor a una variable podremos acceder a él siempre que queramos, teniendo en cuenta que su valor se perderá al cerrar el terminal o al cerrar la sesión, lo que suceda antes. Para acceder al valor de una variable lo haremos poniendo el nombre de la variable precedido del símbolo dólar.

```
cd $home_path  
echo $variable
```

También podemos almacenar en variables el resultado de ejecutar otros programas o comandos. Si recordamos la sección 3.1 Bash, salida estándar y otros, vimos que podíamos redireccionar la

salida de un comando a otro con el uso de pipes. En este caso lo que haremos será guardarnos la salida de un comando para usarla como más nos convenga. Para guardar el resultado de un comando en una variable emplearemos el carácter `''`.

```
directorio_actual='pwd'
```

El trabajo con variables nos permite hacer trabajos complejos. Vamos a ver un ejemplo:

```
cd /root
usuario=Juan
directorio_actual='pwd'
# Creará una directorio con path /root/Juan
mkdir $directorio_actual/$usuario
cd $_
pwd
/root/Juan
mkdir "carpetaDe${usuario}"
ls
carpetaDeJuan
```

5.3 Path

PATH es una variable especial en los terminales de los sistemas Linux. Esta variable permite al terminal saber en que rutas debe buscar los programas cuando tratamos de ejecutar un comando. Es decir, cuando nosotros escribimos en un terminal el comando ls. La variable PATH le indica al terminal donde debe ir a buscar un programa de nombre ls para ejecutarlo. Si el terminal no encuentra el comando en ninguna de las rutas que tiene definida en la variable PATH

obtendremos un error informándonos de que el comando no existe.

Juntando conceptos

Comentábamos en el capítulo 2 que siempre que queramos ejecutar un programa desde cualquier directorio en el que nos encontremos ese programa deberá encontrarse en `/bin`; `/usr/bin`; `/sbin` o `/usr/sbin`. Ahora sabemos que eso no es exactamente así. Aunque la convención es que todos los programas ejecutables se encuentren en esos directorios, podremos ejecutar cualquier programa desde cualquier directorio donde nos encontremos siempre que el programa a ejecutar se pueda encontrar en una de las rutas especificadas en `PATH`.

Las distintas rutas especificadas en la variable `PATH` se separan por el carácter ':' y el orden en el que las encontremos es muy importante. Los comandos o programas a ejecutar se buscarán en las distintas rutas por orden de aparición de esta. Esto significa que en el caso siguiente:

```
echo $PATH  
/usr/local/bin:/usr/bin:/bin
```

Si tenemos un programa llamado `ls` en `/usr/local/bin` y otro llamado exactamente igual en `/bin`. El comando `ls` del directorio `/bin` nunca llegará a ejecutarse.

Podemos manipular esta variable a nuestro antojo ahora que sabemos como asignar valor a las variables. Por ejemplo, si quisiésemos añadir el directorio `dir` a nuestra variable `PATH` tan solo deberíamos hacer:

PATH=dir:\$PATH

PATH=\$PATH:dir

Poner el nuevo directorio delante o detrás del contenido de PATH dependerá de nuestras necesidades. Hay que ser cuidadoso al manipular esta variable puesto que podríamos borrar sin querer todas las rutas que ya tiene guardadas.

Como sucede con otras variables si le asignamos un valor a esta variable ese valor se perderá en cuanto cerremos la ventana del terminal o en cuanto cerremos sesión. Si queremos hacer los cambios permanentes deberemos recurrir a un fichero de configuración. Si recordamos la sección anterior, veremos que debemos añadir estos mismos comandos al fichero `/etc/profile` o bien al fichero `/home/usuario/.bashrc` según nos convenga.

5.4 Autocompletar

Una de las funciones más útiles que nos ofrece bash es la opción de autocompletar. Es cierto que la funcionalidad no está siempre disponible en todos los sistemas Linux aunque siempre es posible instalarla. Debido a que la instalación de esta funcionalidad depende de cada distribución y no es el objetivo del texto tratar cada distribución ni una en concreto, no entraremos en detalles sobre la instalación. Sin embargo, no resulta difícil encontrar información en Internet.

Cuando estamos trabajando con el terminal, si apretamos la tecla tabulador <TAB> este intentará autocompletar lo que hayamos escrito hasta ese momento. El autocompletado es específico de cada aplicación. Eso significa que si ya hemos introducido un comando, el autocompletado, cuando este activo, tratará de autocompletar según

los parámetros que dicho comando requiera.

La función de autocompletar, por defecto, tratará de autocompletar lo escrito como una variable si empieza por \$, con un nombre de usuario si empieza por ~, como un nombre de PC o hostname si lo que hemos escrito empieza por @ o, si no consigue una correspondencia en ninguno de los anteriores como un comando. Si tampoco consigue una correspondencia con ningún comando entonces intentará autocompletar con un nombre de fichero. Por defecto, si presionamos <TAB> una vez y bash no es capaz de autocompletar, si presionamos dos veces consecutivas nos ofrecerá una lista de posibles alternativas.

Hay que tener en cuenta que en caso de ambigüedad el terminal únicamente realizará el autocompletado hasta donde le es posible autocompletar sin ambigüedad. Es decir:

```
ls
```

```
fichero1.txt
```

```
fichero2.txt
```

```
cat fi <TAB>
```

```
cat fichero #No autocompletará más puesto que no puede distinguir  
entre ambos
```

En función del primer carácter de la cadena a autocompletar, el sistema interpretará una u otra cosa.

- \$ Trata de autocompletar con una variable.
 - echo \$ <TAB><TAB> --> \$PATH; \$BASH ...
- ~ Trata de autocompletar con un nombre de usuario
 - cd ~<TAB><TAB> --> ~dani/; ~oscar/

- @ Trata de autocompletar con un nombre de PC
 - ssh dani@<TAB><TAB> --> @servidor; @web Estos valores se recogen de /etc/hosts. Si el fichero está vacío no obtendremos ningún valor.
- Cualquier otro inicio se tratará de autocompletar con un comando
 - mkd<TAB> --> mkdir
- Si no se encuentra ningún comando se tratará de autocompletar con un fichero
 - cd /etc/sud<TAB> --> /etc/sudoers

Lo que hemos comentado hasta ahora se trata del autocompletado estándar que nos ofrece bash. Es posible ir más allá. Hemos comentado que en caso de una aplicación o comando la función de autocompletado intentaría autocompletar según los parámetros requeridos. Esto se consigue mediante el autocompletado programado. Para poder trabajar con él tenemos el comando *complete* que nos permitirá modificar el autocompletado ya existente o añadir nuestras propias funciones para el autocompletado. Vamos a tratar de ver como funciona antes de tratar de añadir nuestras propias especificaciones.

Vamos a referirnos a especificación de autocompletado como cualquier definición de autocompletado que se haya definido para cualquier nombre mediante el uso del comando *complete*. Cuando tratamos de autocompletar una palabra se sigue la secuencia siguiente:

1. Si el nombre identificado tiene una especificación de

autocompletado, esa especificación genera la lista de posibles autocompletados.

2. En caso contrario, si se trata de completar sobre una cadena vacía, es decir si no se ha escrito nada, todas las especificaciones que se hayan definido mediante el comando *complete -E* se usarán para generar la lista de posibles candidatos.
3. Si la palabra que se trata de autocompletar es la ruta de un fichero o directorio, se trata primero de encontrar una especificación de autocompletado para la ruta concreta. Si no da resultado, se trata de buscar una especificación para el nombre que precede a la última barra. Es decir en */home/dani/ca* se buscaría únicamente para *ca*.
4. Si ninguna de las anteriores permite obtener una lista de candidatos para realizar el autocompletado, se utilizarán todas aquellas especificaciones que se han definido mediante *complete -D*.
5. Si tampoco se encuentra ninguna especificación que cumpla con la regla 4, entonces el autocompletado estándar, el que hemos definido más arriba que autocompletaba según nombres de usuario y otras especificaciones, se aplica para obtener la lista de candidatos.

Esta es la secuencia seguida para saber si es posible ofrecer candidatos para el autocompletado. Lo que todavía no hemos visto es

como se obtienen los candidatos. Para ello vamos a ver como funciona el comando *complete*. Este comando nos ayudará, por un lado, a comprender como puede el sistema ofrecernos una lista de posibles candidatos y por otro lado a poder definir nuestras propias especificaciones de autocompletado. No veremos todas las opciones en profundidad pero si las necesarias para hacernos una idea.

Lo primero que debemos saber, es que si todavía no tenemos activado el autocompletado programable podemos hacerlo ejecutando en cualquier terminal

```
./etc/bash_completion
```

Una vez lo tenemos activado ya podemos recurrir al comando *complete*. Este comando tiene multitud de opciones. Quizás la primera que nos interese ver, para hacernos una idea de como va a funcionar es:

```
complete -p
```

Esto nos mostrara las especificaciones de autocompletado que ya tengamos definidas. Podemos definir especificaciones que hagan referencia a las ya mencionadas anteriormente. Puede parecer absurdo definir una especificación para obtener los directorios con la función autocompletar, pero imaginad que queremos que el comando *ls* os devuelva únicamente directorios cuando apretamos el tabulador, en este caso podemos crear una especificación de autocompletado para ello.

- **complete -c [nombre]** : Ofrecerá posible comandos para autocompletar nombre.
- **complete -d [nombre]**: Ofrecerá únicamente

directorios cuando autocompletemos nombre.

- **complete -f [nombre]:** Ofrecerá únicamente ficheros al autocompletar nombre.
- **complete -h [nombre]:** Completará únicamente con nombres de PC (hostnames).
- **complete -u [nombre]:** Completará con usuarios al autocompletar nombre.
- **complete -v [nombre]:** Completará con variables al autocompletar nombre.
- **complete -g [nombre]:** Completará con grupos de usuario.
- **complete -j [nombre]:** Completará con nombres de programas en ejecución siempre que el control de programas en ejecución(o trabajos) esté activado.

Hay todavía algunas más que nos permiten especificar cual va a ser la lista de elementos disponibles para autocompletar. Podemos juntarlas todas ellas en la manera en que más nos convenga para obtener las combinaciones deseadas. Si las combinamos, sin embargo, deberemos tener en cuenta el orden en que las reglas especificadas van a ser interpretadas por la función de autocompletar para no obtener sorpresas desagradables. Más adelante, hablaremos sobre el orden de las distintas reglas del autocompletado programado. El potencial del comando *complete* no acaba aquí. Con algunas opciones podemos llegar a definir nuestras propias listas de opciones o incluso hacer que la lista de elementos dependa de un programa propio.

- **-W wordlist:** Permite mostrar como opciones de autocompletar todas aquellas mostradas en la lista wordlist. La separación de las distintas opciones se hará según la variable IFS (Internal Field Separator, Separador de campos interno). Esto significa que debemos ser cuidadosos a la hora de redactar la lista para no obtener resultados inesperados. Se trata de una variable, por tanto podemos consultar su valor reverenciándola con el símbolo \$.

echo \$IFS.

También podemos darle el valor deseado a dicha variable mediante un = como ya hemos visto o mediante el comando: export IFS=" ", en caso de que queramos separar por espacios.

- **-P valor y -S valor:** Permiten añadir un sufijo o un prefijo a la lista de resultados posibles para el autocompletado. Por ejemplo:

```
complete -P '$' -u ./miPrograma
```

```
./miPrograma <TAB><TAB>
```

```
./miPrograma $dani #Cuando el usuario sería únicamente dani
```

- **-X filtro:** Permite especificar un filtro para los nombres de fichero, cada opción que coincida con un patrón en el filtro es eliminada de la lista de posibles valores para el autocompletado. Si el patrón empieza con el símbolo '!' cada opción que no coincida con el patrón

es eliminada de la lista de posibilidades.

- **-C comando:** Ejecuta el comando especificado y coge el resultado como posibilidad para el autocompletado. Si se quiere devolver más de una posibilidad debe hacerse imprimiendo, con el comando echo, por ejemplo, cada opción en una línea distinta sin ningún separador adicional. Esta opción resulta muy interesante porque permite definirnos nuestros propios comandos o programas para ofrecer una lista de posibilidades de autocompletado 100% personalizada.
- **-F función:** Permite definir una función de la que se obtendrá una lista de posibilidades para el autocompletado. Este método es similar al anterior, sin embargo es más complejo de utilizar a la par que ofrece un mayor potencial. Ofrece un mayor potencial porque mientras que la opción -C ejecuta el comando en un entorno controlado bajo nuestro terminal, la opción -F ejecuta el comando en el mismo entorno en que se está tratando de autocompletar, ofreciendo por ello acceso a determinadas variables y valores que de otra manera no tendríamos. Es muy importante que los resultados se guarden en la variable COMREPLY si se desea que lleguen a mostrarse como opciones de autocompletado. Para hacer uso de esta opción debemos definir una función como la del ejemplo:

```
_parser_options()
```

```
{
local curr_arg;
curr_arg=${COMP_WORDS[COMP_CWORD]}
COMP_REPLY=( $(compgen -W '-i—incoming -o—outgoing -m—missed' -
- $curr_arg ) );
}
```

- **COMP_REPLY:** Es una lista que contiene los resultados que se ofrecerán en el autocompletado.
- **COM_WORDS:** Contiene una lista de todas las palabras que se han tecleado antes de tratar de autocompletar
- **compgen:** Es un comando que genera una lista de posibles valores para el autocompletado dependiendo de las opciones que reciba. La opción **-W** tiene el mismo efecto que en el comando **complete**. En este caso el objetivo de **compgen** es devolver los resultados en el formato adecuado.

Es importante recalcar que en el caso del uso de las funciones, estas deberán guardarse en un fichero y luego ser invocadas antes del comando *complete* mediante el uso de:

source fichero_de_la_función

De esta manera las tendremos disponibles para poder usarlas en el comando **complete**. Es importante en el comando **complete** especificar el nombre de la función y no del fichero que la

contiene, es por eso que resulta interesante distinguir ambos nombres.

Como hemos comentado al usar *complete*, debemos ser cuidadosos en el orden con el que se ejecutan las distintas opciones para no llevarnos alguna sorpresa inesperada cuando realizamos el autocompletado. El orden de las instrucciones a grandes rasgos es el siguiente:

1. Las opciones especificadas con letras minúsculas son las que se tienen en cuenta en primer lugar.
2. A continuación se considera la lista de posibilidades que se haya especificado con la opción `-W`.
3. Cualquier función especificada con la opción `-F` es ejecutada y se obtiene la cadena de resultados.
4. Cualquier comando especificado con `-C` es ejecutado y su resultado obtenido.
5. Cualquier filtro especificado con la opción `-X` se ejecuta a continuación, modificando si es necesario los valores obtenidos anteriormente.
6. Por último, las opciones especificadas por `-P` y `-S` se ejecutan modificando los valores previamente obtenidos.

Si queremos que todos estos cambios sean permanentes, deberemos añadirlos a uno de los ficheros de configuración: `/etc/profile` o `/home/usuario/.bashrc`.

5.5 Algunos comandos básicos

Es muy posible que ya se conozcan varios o incluso muchos

de los comandos más útiles para usar en los sistemas Linux. Estos comandos que se presentan no son dependientes exclusivamente del terminal si no que son comunes a todos los sistemas Linux. Aún cuando es posible que en muchas ocasiones ya se conozcan estos comandos nunca está de más echarles un vistazo. Aunque todos los comandos suelen tener diversas opciones, en este caso no se presentará ninguna o se presentarán únicamente las más relevantes. No conocer las opciones hoy día no supone un problema puesto que sabiendo lo que se busca es sencillo encontrar información.

Usaremos una notación común para representar los comandos mostrados. Todas las palabras que se encuentren entre los caracteres <> representan parámetros obligatorios. Las palabras que se encuentren entre [] son parámetros opcionales. Todas aquellos palabras que no se encuentran entre ningún tipo de carácter forman parte del comando.

cat <fichero1> [fichero2 fichero3 ...]

Muestra por la salida estándar el contenido de los ficheros que se especifiquen. cat trabaja de forma secuencial, es decir que el orden en que se especifiquen los ficheros sí que importa.

head [opciones] <fichero>

Muestra las primeras líneas de fichero por la salida estándar. La opción -n <número> nos permite decidir cuantas líneas nos debe mostrar del fichero.

tail [opciones] <fichero>

Funciona igual que head con la diferencia que muestra las últimas líneas de un fichero. La opción -n <número> de nuevo nos permite decidir cuantas líneas queremos mostrar. El comando tail presenta otra opción interesante que es la opción -f. La opción -f nos permite ir viendo en tiempo real como se van añadiendo líneas al final de un fichero. Para salir de esta visión interactiva únicamente tendremos que apretar la combinación Ctrl+C.

ls [opciones] [ruta]

Lista el contenido de un directorio. Por defecto, si se le invoca sin parámetros muestra el contenido del directorio actual. Es interesante conocer algunas de las opciones de ls.

- **-a.** Muestra todos los ficheros, también los ocultos.
- **-l.** Muestra información sobre las propiedades y permisos de los ficheros.
- **-R.** Actúa de forma recursiva, es decir, lista también el contenido de los subdirectorios.

cp [opciones] <origen1> [origen2 origen3 ...] <destino>

Copia un origen en destino. Si se especifica más de un origen el destino debe ser un directorio. Si se especifica un solo origen el destino puede ser un directorio o un fichero. Si el destino es un fichero se sobrescribirá el contenido o se creará el fichero en caso de que no existiese. Si es un directorio se mantendrá el nombre del fichero en origen. Para poder especificar directorios como origen se debe invocar el comando con la opción -r.

mv [opciones] <origen1> [origen2 origen3 ...] <destino>

Mueve un fichero o directorio de origen a destino. Si especificamos dos ficheros dentro del mismo directorio únicamente lo estaremos renombrando. Igual que con el comando cp si especificamos más de un origen el destino debe ser un directorio.

touch <ruta>

Crea un fichero vacío de nombre ruta. Si el fichero ya existe no hará nada pero la fecha de última modificación será actualizada.

rm <fichero1> [fichero2 fichero3 ...]

Borra un fichero, el fichero borrado no se puede recuperar mediante métodos normales, puesto que se elimina del sistema. Hay que ser muy cuidadoso con el comando rm precisamente por ello. Si en lugar de ficheros comunes queremos especificar directorios deberemos especificar la opción -r. Si queremos eliminar ficheros de solo lectura deberemos especificar la opción -f.

Advertencia

Al combinar las opciones -rf puesto que si tenemos permisos de súper usuario podemos causar un verdadero desastre en nuestro sistema.

cd <ruta>

Cambia al directorio indicado por ruta. Si ruta empieza por / estamos navegando a un directorio absoluto, es decir relativo a la raíz

del sistema. En caso contrario, estamos navegando a un directorio relativo a la ruta donde nos encontremos. Recordemos el capítulo 2.

pwd

Devuelve la ruta completa del directorio en el que nos encontramos trabajando.

mkdir <directorio>

Crea un directorio de nombre "directorio". Si directorio es una ruta válida se crea el subdirectorio en la ruta especificada. Si queremos crear un directorio debemos tener en cuenta que solo se intentará crear el directorio cuyo nombre va tras el último carácter '/'. Así pues si intentamos crear /home/dani/videos/cumpleaños y el directorio videos no existe el comando mkdir devolverá un error. Para que cree toda la estructura de directorios podemos especificar la opción -p. Si lo hacemos, en el ejemplo anterior se crearán tanto el directorio videos como el directorio cumpleaños dentro del directorio dani.

rmdir <directorio>

Permite borrar un directorio. Lo usual es usar la opción -r del comando rm, sin embargo también existe alternativa. Este comando a diferencia de rm-r fallará en caso de que el directorio contenga algún fichero, sea del tipo que sea.

grep [opciones] <expresión>

Este es un comando extremadamente útil. Su uso en solitario no es muy habitual. Normalmente se utiliza este comando

redireccionando la salida de otro comando hacia grep con un pipe o tubería. Grep nos muestra por la salida estándar las líneas de un fichero o de la entrada que corresponda que coinciden con una expresión. Vamos a verlo con un ejemplo. Supongamos que tenemos un fichero de 4000 líneas con contenido similar a:

```
[INFO] test info  
[WARN] test info  
[INFO] test info  
[...]
```

Así hasta llenar las 4000 líneas del fichero. Si empleamos el comando cat este nos mostrara 4000 líneas en nuestro terminal, estas no cabrán, eso sin duda. Sin embargo a nosotros solo nos interesan aquellas líneas que contengan el patrón [WARN] así pues recurrimos al comando grep.

```
cat fichero | grep [WARN]
```

Esta combinación de comandos nos permitirá eliminar todas las líneas que no empiecen con [WARN]. Esto es válido para cualquier combinación que se nos ocurra. Hay que considerar que la salida del comando grep puede redirigirse de nuevo al comando grep. Además grep presenta una opción muy interesante que es la opción -v. Esta opción permite invertir la selección, es decir permite seleccionar todas aquellas líneas que no coinciden con la expresión.

Existen multitud de comandos y programas en los sistemas

Linux y nosotros mismos podemos ampliar el número de ellos desarrollando nuestros propios programas. Sin embargo, los comandos presentados se usan muy a menudo para trabajar en cualquier sistema Linux, sobretodo, pero no exclusivamente, si debemos ser los responsables de gestionar un sistema.

6. Procesos y threads

Otro de los puntales claves de cualquier sistema operativo son los procesos que se ejecutan en cada uno de estos sistemas. Junto con los ficheros y los usuarios que veremos más adelante, es lo que realmente nos permite trabajar con nuestros sistemas. En los sistemas Linux no es diferente y saber trabajar con ellos nos puede ser de gran ayuda.

Un proceso puede definirse de una forma sencilla como un programa en ejecución. Así pues, por fin sabemos lo que es un proceso. Más concretamente podríamos definirlo como una unidad de proceso gestionada por el sistema operativo. Así pues, cuando hablamos de un sistema multiusuario y multitarea o multiproceso, nos referimos a que puede tener al mismo tiempo más de un programa en ejecución.

Hoy en día parece que afirmar esto es una obviedad. Sin embargo, si nos paramos a pensar en las máquinas de hace unos años o en las máquinas de uso doméstico de hoy en día quizás nos daremos cuenta que podemos tener al mismo tiempo varios programas ejecutándose, varios procesos, y que normalmente estos procesos son bastantes más que el número de procesadores que tiene nuestra máquina. ¿Como es posible que se ejecuten simultáneamente más procesos que unidades de procesamiento tiene nuestra máquina? La respuesta es sencilla y bastante obvia una vez se conoce: no se puede.

Los procesos tienen diversos requisitos y necesidades. Por ejemplo, pueden necesitar estar esperando a poder leer o escribir en

un fichero, pueden estar esperando a que el usuario introduzca datos, pueden estar esperando a recibir datos de la red, pueden estar procesando, etc. A fin de cuentas pueden estar ejecutándose, esperando por entrada/salida de cualquier tipo o en algún otro estado que se comentará más adelante. Es tarea del sistema operativo asignar tiempo de procesamiento a uno u otro proceso según las necesidades de este y según el estado de cada proceso. Es responsabilidad de nuestro sistema Linux, por tanto que el usuario tenga la sensación de multitarea aprovechando los distintos estados para solapar ejecuciones. El detalle de los núcleos de procesamiento disponibles y de estas limitaciones nos afectará pocas veces. Únicamente si debemos desarrollar aplicaciones donde el tiempo o los recursos son realmente críticos deberemos tener en cuenta estos detalles.

La manera usual de funcionar de cualquier planificador consiste en interrumpir muy brevemente la ejecución de todos los procesos para realizar las tareas básicas del Kernel, por ejemplo mantener la información en la memoria RAM, e introducir el planificador para asignar tiempo de cómputo a uno u otro proceso.

Todo proceso está formado por un activo y por un flujo de ejecución. El flujo de ejecución se corresponde con las instrucciones del programa que se van ejecutando y es conocido como thread o hilo de ejecución. Cuando tenemos un proceso con múltiples threads tenemos entonces un proceso con múltiples flujos de ejecución. El tiempo de ejecución de cada CPU es asignado por threads, aunque teniendo en cuenta a que proceso pertenecen. La otra parte que tiene todo proceso es su activo. Este activo se compone de:

- El espacio de memoria asignado al proceso.

- Las variables globales.
- Ficheros abiertos por el proceso.
- Los procesos hijos, es decir creados por el proceso.
- Temporizadores.
- Otros.

Estos componentes son comunes a todos los hilos de ejecución y serán por tanto los que permitirán la comunicación entre estos. Serán también los que permitan establecer los límites de recursos para cada uno de los procesos.

Los demonios o servicios no son otra cosa que procesos que se ejecutan en nuestro sistema y que tienen unas características concretas. Estos procesos son bastante habituales en los sistemas Linux y suelen identificarse porque tienen una `d` en su nombre. Por ejemplo `sshd` es el demonio de un programa que permite recibir conexiones remotas, a través de la red, de otras máquinas. Las características comunes en estos procesos son:

- Se arrancan al iniciar el sistema.
- No mueren. Es decir, están siempre en ejecución, suele ser habitual que cuando un demonio deja de ejecutarse haya otro programa que esté pendiente de volverlo a levantar, es decir de ejecutarlo de nuevo.
- Suelen ser procesos que se ejecutan en segundo plano y que no tienen asociado ningún terminal.

- Suelen estar a la espera de un evento para empezar a trabajar.
- Es habitual que deleguen las tareas a otros procesos.

6.1 Listando los procesos en nuestro sistema

Cada proceso en un sistema Linux está identificado por un número o PID, process ID por sus siglas en inglés. Este identificador es único para cada proceso y es el mismo durante todo el ciclo de vida del proceso. Esto nos puede llegar a ser sumamente útil puesto que recordando un número somos capaces de trabajar con el proceso mientras esté vivo.

Para listar los procesos en nuestro sistema recurriremos al comando `ps` (*process snapshot*).

```
ps
PID TTY STAT TIME      COMMAND
520 p0 S 0:00 -bash
31956      p3 R 0:00 ps
[...]
```

Este es un ejemplo de la salida que obtendríamos para el comando `ps`. Vamos a echar un vistazo a lo que significa cada columna, aunque si hemos ido siguiendo el texto completo algunas de las columnas nos resultarán bastante intuitivas.

- **PID:** El identificador numérico del proceso
- **TTY:** El terminal o dispositivo sobre el que está

ejecutándose el comando.

- **STAT:** El estado del proceso. Más adelante veremos en que estados podemos encontrar a los procesos en nuestros sistemas Linux.
- **TIME:** Aunque esta columna parece intuitiva, nada más lejos, puesto que nos muestra el tiempo de CPU que el proceso ha usado hasta ahora. Esto significa que el proceso puede llevar 45 minutos ejecutándose a la espera de un evento y el tiempo mostrado será 0:00. Únicamente el tiempo que el proceso ha estado ejecutando instrucciones se computa aquí.
- **COMMAND:** El comando o programa que corresponde con ese identificador de proceso.

Al igual que muchos otros comandos, ps puede invocarse usando opciones para modificar la información o los detalles que ofrece. A diferencia de otros comandos, ps no recibe sus opciones, es decir sus argumentos, precedidos del carácter '-', si no que simplemente recibe todas sus opciones separadas del comando por un espacio. Algunas de las opciones más conocidas son:

- **x:** Muestra todos los procesos que corren en el sistema que pertenecen al usuario que lanza el comando
- **a:** Muestra los procesos para todos los usuarios
- **u:** Incluye información más detallada de los procesos

- **w:** Muestra los comandos completos. Normalmente solo se muestra la parte del comando que cabe en la línea de salida del comando ps.
- **t:** muestra los procesos que pertenecen al terminal especificado. Si no se especifica terminal, muestra los procesos que pertenecen al terminal en uso.
- **m:** muestra también los threads de cada proceso. Por defecto ps muestra únicamente los procesos, sin embargo, si distinguimos entre threads y procesos, ¿porque no listarlos también?

El comando ps puede aceptar los parámetros al estilo tradicional de UNIX, es decir precedidos por el carácter '-', puede aceptar los parámetros según el estilo de GNU, es decir precedidos por dos caracteres '--' o puede aceptarlos en el formato BSD que es como se ha descrito anteriormente. El más habitual suele ser el que se ha mostrado más arriba. Hay que ser cuidadoso pues los parámetros no son los mismos usándolos según una u otra nomenclatura.

Al listar los procesos con el comando ps debemos considerar todavía dos cosas, la primera de ellas es que la columna tty puede tomar el valor '?'. Esto significa que el proceso no está asociado a ningún terminal o que se desconoce el terminal al que está asociado. La segunda cosa a tener en cuenta son los estados que puede tener un proceso. Esta es quizás una de las informaciones más relevantes que nos puede ofrecer el comando ps. Vamos a ver algunos de los estados que podemos encontrarnos:

- **S:** Espera interrumpible. Normalmente esperando a que suceda un evento.

- **R:** Ejecutándose o esperando ser ejecutado.
- **D:** Espera ininterrumpible. Normalmente en algún proceso de entrada/salida
- **Z:** Difunto. Un estado bastante malo, significa que el proceso ha muerto pero el proceso padre ha sido incapaz de liberar sus recursos todavía.
- **X:** Nunca deberíamos ver este estado. Significa que el proceso ha muerto. Si vemos este estado es porque el proceso ha sido incapaz de liberar los recursos que tenía asignados en el momento de terminar. Tenemos, en este punto, unos recursos ocupados por un proceso que no existe.

En algunas ocasiones, es posible que encontremos más de un carácter para representar el estado. Suele ser siempre así para BSD, encontraremos igualmente en la mayoría de sistemas Linux estos modificadores de los estados, veamos que significan algunos de ellos:

- **<:** Alta prioridad.
- **N:** Baja prioridad.
- **I:** Se trata de un proceso con múltiples hilos.
- **s:** El proceso lidera una sesión. Una sesión no es otra cosa que una manera de agrupar procesos para que el sistema pueda tratar con ellos de una forma sencilla. Por ejemplo, suele ser habitual que una ventana terminal sea líder de sesión, de esta manera al cerrar el terminal es más fácil lidiar con

todos los procesos que pertenecen a dicha sesión.

6.2 Señales

Una señal es un mensaje del kernel destinado a un proceso. Vendría a funcionar de una manera similar a como trabajan las excepciones de hardware. La idea detrás de las señales reside en que cualquier proceso que este en ejecución y reciba una señal detendrá su ejecución para tratar la señal y en caso de que la señal no implique terminar el proceso, una vez tratada, el proceso recuperará su ejecución normalmente.

Normalmente las señales solo pueden ser enviadas por el kernel o por otro proceso. Cuando se trata de un proceso, este únicamente puede mandar señales a procesos que han sido iniciados por el mismo usuario, de otra manera se necesitarán permisos de súper usuario. Las señales permiten enviar diversos mensajes a los procesos, de manera que podemos controlar su ciclo de vida con el uso de señales. En algunas ocasiones, si somos nosotros los desarrolladores de alguna aplicación, incluso podremos interactuar con las señales desde la aplicación para obtener el comportamiento que consideremos oportuno.

El comando *kill* nos permite mandar señales a un proceso si conocemos su PID.

`kill 1359 #Manda la señal TERM(terminar) al proceso con PID 1359.`

Por defecto el comando *kill* siempre manda la señal TERM al PID especificado, sin embargo es posible enviar la señal que deseemos pasándosela como parámetro.

kill -2 1359

kill -INT 1359

Cualquiera de las dos formulas anteriores es correcta y de hecho ambas hacen lo mismo. En este caso mandar una señal, la de interrupción del teclado. Suele ser habitual esta señal para parar o finalizar procesos en los sistemas Linux. De hecho, esta señal es el equivalente al famoso Ctrl+C que ha ido apareciendo a lo largo del texto.

Las señales disponibles en el sistema son bastante variadas. A continuación veremos algunas de ellas. Están expresadas en dos formatos, ambos válidos para usar junto con el comando kill. Sin embargo, a la primera columna debemos quitarle las tres primeras letras en caso de querer pasarlas como parámetros. ¿Por que incluir esas letras entonces? En el lenguaje de programación C, por ejemplo, se usan con el nombre completo y en los manuales de los sistemas Linux aparecen también con estos nombres, así que usaremos sus nombres completos.

Señal (nombre)	Señal (valor numérico)	Descripción
SIGINT	2	Interrupción del teclado
SIGQUIT	3	Cerrar desde el teclado
SIGILL	4	Instrucción no permitida o desconocida
SIGABRT	6	Señal para abortar el proceso

SIGKILL	9	Señal de muerte
SIGPIPE	13	Salta cuando se escribe en un pipe pero no hay nadie dispuesto a leer. Normalmente no salta hasta que el pipe no está lleno.
SIGTERM	15	Señal de terminación
SIGTSTP	18,20,24	Señal de parada

Las anteriores son sólo una muestra de las señales que hay. Se trata, quizás de las más comunes. No todas ellas tienen el mismo efecto y las descripciones de las señales son muy vagas. Sin embargo, el tratamiento de una u otra señal puede ser modificado por los procesos, por lo que dar una descripción detallada resulta bastante complicado. Pese a eso, las señales SIGSTOP y SIGKILL son señales que no pueden ser tratadas por los programas que desarrollemos, si no que fuerzan siempre la terminación del proceso sea cual sea el estado de este. Así pues, si algún proceso nos está causando problemas siempre podemos recurrir a:

kill -9 PID

Finalmente, hay algunas de estas señales que tienen una equivalencia directa en cuanto a combinación de teclas del teclado. Son las siguientes:

- **CTRL+C:** kill -2. Manda una señal de interrupción al proceso.

- **CTRL+Z:** kill -{18,20,24} ó kill -TSTP. Detiene el proceso, este se puede continuar con el comando fg o bien con kill -CONT.
- **CTRL+l:** kill -3. Envía la señal de cierre al proceso.

6.3 Viendo los procesos en tiempo real

Aunque el comando ps nos da muchísima información acerca de los procesos de nuestro sistema, nos muestra los procesos en un único instante del tiempo, ps viene del inglés *process snapshot*. Si necesitamos ver en tiempo real que es lo que está haciendo nuestro Linux y que procesos se están ejecutando, deberemos recurrir al comando top.

top

top - 11:36:06 up 1:03, 2 users, load average: 0,38, 0,31, 0,32

Tareas: 249 total, 2 ejecutar, 247 hibernar, 0 detener, 0 zombie

%Cpu(s): 1,1 usuario, 0,4 sist, 0,0 adecuado, 98,4 inact, 0,0 en espera, 0,0 hardw int, 0,0 softw int, 0,0 robar tiempo

KiB Mem: 16348032 total, 4008256 used, 12339776 free, 275200 buffers

KiB Swap: 16690172 total, 0 used, 16690172 free. 1255640 cached Mem

PID	USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	HORA+
4436	dani	20	0	982792	154512	91896	S	3,0	0,9	0:56.36
4807	dani	20	0	902092	240028	74752	S	2,3	1,5	0:21.27

chrome

3072 dani 20 0 1138196 282700 110108 S 2,0 1,7 2:14.22

chrome

1608 dani 20 0 404816 138800 55904 S 1,3 0,8 2:39.46

Xorg

2649 dani 20 0 1978388 258592 93712 S 1,0 1,6 2:06.50

cinnamon

4017 dani 20 0 1027028 355312 80784 S 1,0 2,2 1:24.25

chrome

2404 dani 9 -11 512236 12708 9988 S 0,7 0,1 0:17.32

pulseaudio

4879 dani 20 0 718964 52856 31332 S 0,7 0,3 0:02.09

chrome

2685 dni 20 0 2421212 119344 32568 S 0,3 0,7 0:16.68

dropbox

3206 dani 20 0 864884 168212 56048 S 0,3 1,0 0:43.57

chrome

4762 dani 20 0 622428 33680 23552 S 0,3 0,2 0:01.74 gnome-terminal

4849 root 20 0 0 0 0 S 0,3 0,0 0:00.09

kworker/1:2

1 root 20 0 33888 4368 2600 S 0,0 0,0 0:01.27

init

2 root 20 0 0 0 0 S 0,0 0,0 0:00.00

kthreadd

3 root 20 0 0 0 0 S 0,0 0,0 0:00.01

ksoftirqd/0

Como vemos, al ejecutar el comando top los parámetros se van actualizando cada segundo, dándonos información nueva de lo que pasa en nuestro sistema. Como se puede observar, top nos ofrece información sobre el estado de nuestra máquina y sobre los recursos utilizados, tanto de manera global como por proceso.

Vamos a ver la información que nos ofrece este comando tan útil paso por paso. Lo primero que vamos a ver es el tiempo de actividad del sistema y su carga media.

```
top - 11:36:06 up 1:03, 2 users, load average: 0,38, 0,31, 0,32
```

En primer lugar vemos que nos informa de la hora actual del sistema. En segundo lugar se nos muestra cuanto tiempo lleva el sistema encendido. A continuación se muestra el número de usuarios que se encuentran conectados en ese momento al sistema. Finalmente muestra tres valores. El primer valor es la carga media del sistema en el último minuto, el segundo es la carga media en los últimos 5 minutos y el último se corresponde con la carga media en los últimos 15 minutos. Estos tres valores son los que nos van a permitir saber si hay una carga excesiva en el sistema o si se trata de un pico de carga temporal.

```
Tareas: 249 total, 2 ejecutar, 247 hibernar, 0 detener, 0 zombie
```

La segunda línea, que se muestra inmediatamente arriba, es bastante explicativa pues nos muestra el número de procesos de nuestro sistema y el estado de cada uno. Los procesos que marca como hibernando son procesos que se encuentran a la espera de que suceda algo, bien sea entrada/salida o cualquier otro evento. Por otro lado, los procesos zombies son aquellos que se han quedado huérfanos, es decir aquellos cuyos padres ya no se están ejecutando.

A continuación las tres líneas siguientes muestran el uso de recursos del sistema. La primera línea muestra el uso de CPU, la segunda línea muestra el uso de memoria física y la tercera el uso de la memoria virtual. De hecho si nos fijamos cuidadosamente veremos que top nos está mostrando, separado del resto de la línea por ':', que información nos va a dar.

```
%Cpu(s): 1,1 usuario, 0,4 sist, 0,0 adecuado, 98,4 inact, 0,0 en
espera, 0,0 hardw int, 0,0 softw int, 0,0 robar tiempo
KiB Mem: 16348032 total, 4008256 used, 12339776 free, 275200
buffers
KiB Swap: 16690172 total, 0 used, 16690172 free. 1255640
cached Mem
```

En este caso los valores que nos muestra en las distintas líneas son los siguientes:

- **(us)usuario:** Indica el % de CPU usado por el usuario.
- **(sy)sistema:** Indica el % de CPU usado por el kernel.
- **(id)inactivo:** Indica el % de CPU usado por procesos inactivos.
- **(wa)en espera:** Indica el % de CPU usado por procesos en espera.
- **hi(interrupciones de hardware):** Indica el % de CPU gastado en interrupciones de hardware.

- **si (interrupciones de software:** Igual que el anterior, con la diferencia que este es exclusivo de interrupciones de software.

Se debe tener en cuenta que el porcentaje de CPU usado por top puede no ser real. Al final, el uso de CPU se calcula en función del tiempo de uso de la CPU de cada proceso y se calcula cuanto de ese tiempo de CPU se ha dedicado a usuario, cuanto al sistema, etc. Esto significa, que los primeros datos de CPU quizás no sean en exceso relevantes.

En cuanto a las dos líneas siguientes, ambas muestran, en este orden, la memoria total, la utilizada, la libre y la que se encuentra en buffers o en caché. Aunque no entraremos en detalles sobre este último valor, en la mayoría de casos, podemos considerar este valor como parte de la memoria libre.

Una vez hemos pasado estas primeras líneas de información encontramos un listado de los procesos que se ejecutan en nuestra máquina con un montón de información desglosada por columnas que nos proporcionan muchísimos datos sobre los procesos de nuestro sistema. En la gran mayoría de ocasiones resulta imposible conocer exactamente todos los procesos mostrados por el comando top. Las columnas que encontramos para cada uno de los procesos y la información que nos proporcionan es la siguiente:

- **PID:** El identificador del proceso.
- **USUARIO:** Usuario propietario del proceso.
- **PR:** La prioridad de la tarea. En general suele ser habitual ver a todos los procesos con la misma prioridad. Esto afecta a como el planificador del

sistema tratará la tarea.

- **NI:** La manera en que la prioridad es asignada al proceso. Un valor negativo indicará una alta prioridad. Un valor neutro, 0, indicará que la prioridad no debe considerarse con este proceso.
- **VIRT:** La memoria virtual usada por el proceso en Kb.
- **RES:** La memoria física usada por el proceso en Kb.
- **SHR:** Memoria compartida usada por el proceso. Cada proceso suele tener asignado su espacio de memoria. En algunas ocasiones es necesario compartir información entre procesos. En estos casos se recurre a la memoria compartida. La memoria compartida puede encontrarse como un directorio en /dev/shm, en la mayoría de distribuciones Linux. Si recordamos el capítulo 3. Dispositivos, podemos darnos cuenta que se trata de una interfaz que nos ofrece el sistema.
- **S:** Se trata del estado del proceso. Aquí igual que sucedía con el comando ps, los estados son representados con uno o dos caracteres que son los mismos que listábamos en el caso de ps.
- **%CPU:** El porcentaje de CPU usado por el proceso. Igual que antes no se trata del porcentaje de CPU real usado si no del porcentaje de tiempo que el proceso ha usado la CPU. Este porcentaje es

relativo a la última actualización de top, por lo que se debe ser cuidadoso al interpretar su contenido. En ocasiones podemos ver que este porcentaje supera el 100%. Esto no significa que nuestro Linux se ha vuelto loco, simplemente en la mayoría de ocasiones nuestros sistemas Linux nos muestran el porcentaje en función al número de CPUs de los que disponemos. Esto significa que, si nuestra máquina tiene 8 CPUs, el porcentaje máximo que podemos observar es 800% en lugar del 100%.

- **%MEM:** Porcentaje de memoria RAM (física, no virtual) utilizada por el proceso.
- **TIME+:** Muestra el tiempo total de CPU que ha usado el proceso desde que se inició. Esto significa que no se trata de cuanto rato lleva ejecutándose el proceso, si no cuanto de ese tiempo ha estado haciendo verdaderos cálculos. Es posible que en lugar de TIME+, el comando top nos muestre TIME, en este caso la información que nos da es la misma, la diferencia estriba en que TIME+ tiene mayor granularidad, es decir, es más preciso.

6.3.1 Interactuando con el comando top

El comando top es capaz de mostrarnos mucha información y aún y así, sus funcionalidades no terminan ahí. Como muchos otros comandos top puede ser invocado con diversas opciones a las que se llama usando la sintaxis habitual de '-' precediendo a la opción.

En el caso de top además contamos con una serie de comandos interactivos, es decir una serie de opciones que podemos invocar mientras estamos observando los distintos procesos que están en ejecución en nuestro sistema Linux. Veamos algunos de los más comunes:

- **d:** Por defecto top se refresca, es decir actualiza la información, cada 3 segundos. Podemos cambiar este valor presionando la tecla d e introduciendo luego el nuevo valor en segundos que debe marcar el refresco de top.
- **k:** Permite especificar un PID, luego la señal SIGTERM es enviada a dicho proceso.
- **r:** Permite cambiar la prioridad de un proceso. Antes de poder cambiar la prioridad nos pedirá el PID para poder saber a que proceso se le debe cambiar la prioridad.
- **shift+w:** Permite guardar los resultados del comando top al fichero /root/.toprc. Por supuesto, debemos tener permisos para escribir en este directorio
- **h:** Permite invocar la ayuda sobre las distintas opciones y comandos.

Existen más opciones y caracteres, se pueden obtener la lista de aquellos que acepta nuestro programa top, simplemente invocando la opción h mientras ejecutamos el comando top.

Hay un par más de opciones que también resultan muy útiles.

Supongamos que queremos guardar la salida del comando `top` a un fichero porque queremos realizar la caracterización de un sistema o por el motivo que sea. Estar constantemente apretando `shift+w` no es cómodo ni, en ocasiones, viable. Así pues necesitamos una manera de guardar estos datos. Para ello `top` nos ofrece dos opciones muy interesantes, aunque no se trata de las únicas opciones que tiene este comando.

- **-n:** permite realizar el comando un número `n` de veces y luego termina su ejecución
- **-b:** Lanza `top` en modo consola de manera que es mucho más sencillo guardar los resultados en un formato comprensible.

Conociendo las opciones, para solucionar nuestro problema, podríamos realizar algo como lo siguiente:

```
top -n 10 -b > salida_top.txt
```

6.4 Planificación de tareas

En los sistemas Linux encontraremos un par de herramientas que nos serán extremadamente útiles: `cron` y `at`. Este par de herramientas permiten planificar la ejecución de una tarea de una manera bastante sencilla pero que a la vez nos ofrece una flexibilidad asombrosa.

Básicamente lo que nos van a permitir ambos comandos es ejecutar nuestras tareas, es decir, programas, comandos o scripts, en el momento en el que queramos, en el caso de `at` y de forma periódica o puntual en el caso de `cron`. Vamos a ver como trabajan estas dos herramientas.

6.4.1 at

El comando `at` permite programar tareas, aunque estas no serán persistentes. ¿Que significa esto? Significa que en el momento que apaguemos el ordenador todas las tareas programadas con `at` serán eliminadas.

Podemos invocar el comando `at` de la siguiente manera:

```
at 11:00
```

```
at 11:00 15/09/2015
```

Si la hora y la fecha son correctas, `at` no nos va a dejar programar una tarea en el pasado, nos aparecerá un prompt, es decir un puntero en el terminal que nos da la opción de escribir. Desde ese momento podemos escribir tantos comandos como queramos, tal como si los estuviésemos escribiendo en un terminal. Una vez hayamos terminado de introducir todas las tareas que queremos programar para esa fecha simplemente debemos presionar `Ctrl+D` y saldremos del comando `at`, dejando las tareas correctamente programadas.

Una vez hemos programado nuestras tareas podremos consultarlas con el comando:

```
atq
```

Si nos fijamos en la salida de `atq`, veremos que junto con cada tarea nos aparece un número identificador. Este número es único para cada tarea y nos permite identificarlas por si quisiésemos trabajar con ellas. Por ejemplo, nos podría resultar interesante borrar una de las tareas que tuviésemos programadas. En este caso podríamos recurrir a:

atrm <identificador_de_tarea>

Para hacernos la vida más fácil el comando at permite trabajar con algunas palabras en inglés que nos harán la vida más fácil. A continuación se muestran algunos ejemplos, no son todos los existentes, pero no es difícil encontrar información sobre estos detalles en Internet.

at midnight #Se ejecutarán las tareas a medianoche

at 10:32 +5 days #Se ejecutarán las tareas a las 10:32 de aquí a 5 días

at 10:32 +5 weeks #Se ejecutarán las tareas a las 10:32 de aquí a 5 semanas.

at 10:32 APR 20 #Se ejecutará la tarea a las 10:32 del 20 de abril

En el caso de especificar los meses hay que tener en cuenta que se deben expresar siempre en mayúsculas las tres primeras letras del mes en inglés. Para saber cuales son las opciones que acepta vuestro comando at es posible que sea necesario recurrir al manual. Recordemos que la mayoría, si no todas las distribuciones de Linux, cuentan con el comando man. Este comando nos permite ver el manual de la mayoría de comandos y programas de nuestro sistema.

man at

6.4.2 cron

El comando cron permite programar tareas de forma persistente, es decir podemos apagar el ordenador y al volverlo a

encender las tareas seguirán programadas, siempre y cuando el tiempo de ejecución no haya pasado. Cron no es capaz de ejecutar comandos mientras la máquina está apagada. Aún así, nos permite programar tareas periódicas o esporádicas con una flexibilidad que resulta muy útil en muchísimas ocasiones. A diferencia del comando `at`, tenemos la posibilidad de ejecutar las tareas cada minuto o cada día sin mucho esfuerzo.

La manera de planificar tareas con cron es usando el fichero `crontab`, dado que se trata de un fichero de configuración, si recordamos la sección 2.2 Jerarquía de directorios, no puede estar en otro lugar que en `/etc`. Aunque se puede editar este fichero directamente es recomendable usar el comando

`crontab -e`

En este caso, vamos a asumir que se usa mediante la invocación de este comando en todas las ocasiones. Al ejecutar el comando se nos permitirá, si es la primera vez que lo ejecutamos y si contamos con diversos editores de texto, escoger con que editor de texto vamos a trabajar. Apartir de aquí se nos da la opción de editar un fichero en el que podemos poner las tareas que queremos planificar. Una vez hayamos terminado simplemente cerramos el fichero guardando los cambios, esto se hará de una u otra manera dependiendo del editor, y ya tendremos nuestras tareas planificadas.

El fichero contiene una tarea por línea y cada línea sigue el formato siguiente:

variables para ejecutar los comandos, ya deberíamos saber que significan todas

Podemos refrescar la memoria en el capítulo de Bash

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
```

tareas

***** comando a ejecutar

En las tareas, lo primero que tenemos son los datos para definir la fecha de ejecución, seguida del comando. Esta fecha de ejecución se define por 5 valores. Estos valores deben leerse en conjunto puesto que se modifican entre ellos. Los 5 valores nos dicen, de izquierda a derecha:

- **Minuto:** Puede tomar un valor entre 0-59. Si ponemos el valor 58 por ejemplo se ejecutara en el minuto 58 de la hora y día que especifiquemos, no cada 58 minutos. Si especificamos un * se ejecutará cada minuto. Si queremos especificar que la ejecución debe darse cada 5 minutos deberá expresarse el valor */5.
- **Hora:** Puede tomar un valor entre 0-23. Funciona de la misma manera que el anterior.
- **Día del mes:** Puede tomar un valor entre 0-31. Funciona igual que los anteriores.
- **Mes:** El mes expresado en valor numérico (1-12) o bien con las tres primeras letras en minúsculas del nombre del mes en inglés. Si especificamos un * se

ejecutará cada mes.

- **Día de la semana:** Expresado en valor numérico (0-7). Hay que tener en cuenta que el domingo es tanto el 0 como el 7. Si especificamos un * se ejecutará cada día de la semana.

Puede parecer algo confuso, pero algunos ejemplos nos pueden ser de utilidad para entender exactamente como funciona:

Ejecutar un comando cada minuto. Un asterisco en un campo significa #todos.

***** comando

Ejecutar un comando cada 5 minutos los lunes. La barra indica #fracciones

* / 5 *** 1 comando

Ejecutar un comando el primero de cada mes a las 6 de la mañana

0 6 1 ** comando

Ejecutar un comando cada minuto de 6 a 7 de la mañana

* 6 *** comando

Ejecutar un comando el 25 de octubre a las 20:32

32 20 25 10 * comando

Ejecutar un comando cada 2 dias a las 7 de la mañana

0 7 * / 2 ** comando

Ejecutar un comando a las 7 de la mañana y de la tarde. Se
usa el separador para especificar dos valores
0 7,19 *** comando

Además de los distintos valores que se han mostrado, numéricos; asteriscos y /, cron nos ofrece también algunas construcciones que nos pueden ser muy útiles. Estas construcciones deben substituir a los 5 valores. Algunas de ellas son las siguientes:

- **@yearly:** Se ejecuta 1 vez al año.
- **@monthly:** Se ejecuta una vez al mes.
- **@weekly:** Se ejecuta una vez a la semana.
- **@daily:** se ejecuta una vez al día.
- **@reboot:** se ejecuta al reiniciarse el sistema.

Aclaración

Al especificar una de las construcciones de la lista cron interpretará que la instrucción debe completarse en el primer instante posible. Dicho de otra forma, si especificamos @yearly, la acción se ejecutará el 1 de enero a las 00:00. Si especificamos @weekly la acción se ejecutará cada lunes a las 00:00.

Dado que se trata de una herramienta muy potente existen maneras de controlar que usuarios tienen acceso a poder planificar tareas con cron. Para ello se hace uso de dos ficheros de configuración. En ausencia de estos ficheros, dependiendo de la distribución de Linux que usemos, tendremos que o bien todos los

usuarios pueden programar tareas, o bien que únicamente puede hacerlo el súper usuario. Estos ficheros son

- **/etc/cron.allow:** Donde se definen los usuarios autorizados a razón de uno por línea
- **/etc/cron.deny:** Donde se definen los usuarios no autorizados a razón de uno por línea.

¿Por que tener dos ficheros? Eso depende de la política que se quiera implementar. Si se opta por una política de lista blanca, especificar solo aquellos usuarios que tienen acceso, se hará uso del primero. Si se desea emplear una política de lista negra, permitir el acceso a todos menos a aquellos que cumplen unos requisitos concretos, se empleará el segundo de los ficheros.

Una vez hayamos terminado con cron podemos consultar que tareas tenemos programadas con

crontab -l

6.4.2.1 Como funciona cron

Cron debe leer las tareas que tiene pendientes y ejecutarlas en el momento preciso. Pero, ¿como hace esto? Cron es otro proceso más y gasta, por tanto, recursos del sistema. Si cada instante estuviese realizando comprobaciones no tendríamos recursos en nuestro computador para hacer otras tareas o para otros procesos. Así pues, la idea tras el trabajo de cron es la siguiente:

1. Lee las tareas pendientes.
2. Determina si alguna debe ejecutarse en este instante de tiempo.

1. En caso afirmativo, ejecuta la tarea programada.
3. El proceso se pone a “dormir” durante un minuto.
4. Vuelve a 1.

Este es un comportamiento arcaico de cron y no muy eficiente puesto que si no hay tareas o tenemos muy pocas tareas le estamos dando a cron recursos del sistema cada minuto, cuando estos recursos podrían ir destinados a otros procesos. Obtenemos entonces un funcionamiento similar al siguiente:

1. Se busca el fichero crontab para todos los usuarios.
2. Para cada fichero se determina la próxima fecha en que debe ejecutar-se una tarea.
3. Se colocan todos los comandos una lista de eventos.
Para más información sobre la lista de eventos podéis realizar una búsqueda en Internet sobre “Franta-Maly”.
4. Se escoge el primer evento de la lista y se calcula cuanto tiempo falta hasta que sea ejecutado.
5. Se pone el proceso cron a dormir durante el tiempo que se ha calculado en el paso anterior.
6. Al despertar se comprueba que la fecha es correcta y se lanza el comando que corresponda.
7. Se calcula si el comando debe volver a ser ejecutado.
 1. En caso afirmativo se re-encola el comando con el nuevo cálculo de tiempo.

8. Vuelta a 4.

Si nos fijamos cuidadosamente veremos que los pasos de 1 a 3 solo se ejecutan 1 vez al lanzar el demonio de cron. Esto significa que si modificamos el fichero `/etc/crontab` a mano, nuestra tarea no va a entrar a la lista de tareas pendientes hasta que no reiniciemos el demonio de cron o la máquina. Es por eso que se recomienda usar el comando `crontab -e`, puesto que el comando gestiona la inclusión de la tarea en la lista de eventos por nosotros.

7. Recursos y monitorización

Hemos hablado de ficheros, hemos hablado de dispositivos y de procesos. Son pocos los puntales que quedan por descubrir de cualquier sistema Linux sin embargo, ningún sistema operativo podría existir si no fuese capaz de aprovechar los recursos de las máquinas subyacentes. Es por eso que ser capaz de monitorizar los recursos de los que disponemos y el uso que se les da es un activo importante para cualquiera que trabaje con un sistema Linux.

7.1 Visión general de los recursos

Cualquier sistema de hoy en día tiene como objetivo aprovechar los recursos que tiene la máquina que hay por debajo. De hecho, esa es la función del sistema operativo, permiternos interactuar con el hardware de una manera sencilla y al alcance de todos.

En este sentido siempre que trabajemos con nuestros sistemas Linux nos va a ser de utilidad saber con que recursos contamos y cual es el estado de estos recursos. Por supuesto, los recursos de los que más oímos hablar son también los que nos brindan mayor utilidad. Hablamos de la CPU, la memoria RAM y el espacio de disco del que disponemos.

7.1.1 cpu

La cpu o las cpus, puesto que hoy en día todos los ordenadores disponen de más de una, son las que nos permiten realizar cálculos, es decir ejecutar programas y mantener vivos los procesos. Los distintos procesos de nuestro ordenador compiten por obtener tiempo de procesamiento y son las CPUs las que nos van a permitir dárselo. Este número de CPUs viene dado por el procesador

que tengamos. Por ejemplo: un intel i7.

En el directorio `/proc` encontraremos cantidad de información relacionada con el procesador. Concretamente, en la mayoría de distribuciones podremos ejecutar el comando:

```
cat /proc/cpuinfo
```

Este comando nos va a dar una gran cantidad de información sobre nuestra CPU: quien es el fabricante, si podemos hacer multithread y todas las características que tienen las CPUs que tenemos instaladas en la máquina. De la misma manera, podremos obtener información similar mediante el comando:

```
lshw
```

Esta información no nos aporta información del sistema en tiempo real. Podemos obtener información de lo que nuestra máquina tiene instalado pero no de como se está comportando. Como se están comportando nuestras CPUs viene dado por lo que se conoce como la carga del sistema. La carga del sistema no es otra cosa que el tiempo de CPU que consumen los procesos. Si consumen mucho tiempo, apenas tendremos oportunidades para realizar nuevos cálculos por lo que nuestro sistema se ralentizará. Podemos obtener información en tiempo real de la carga del sistema con el comando `uptime`.

```
uptime
```

```
18:17:07 up 68 days, 3:57, 6 users, load average: 0.16, 0.07, 0.06
```

Como vemos `uptime` nos devuelve la hora actual, el tiempo que la máquina lleva encendido y tres medidas de la carga de nuestras CPUs. De estos tres valores ya se habló en la sección 6.3 Viendo

procesos en tiempo real y simbolizan la carga media del último minuto, la carga media de los 5 últimos minutos y la carga media de los 15 últimos minutos. Una carga alta no tiene que ser necesariamente algo negativo. Aunque una carga superior al 70%, o el equivalente si tenemos más de una CPU, suele indicar que nuestra máquina empieza a saturarse y que habrá ralentización.

Los números que se muestran no están normalizados. Esto significa que si tenemos 16 núcleos de procesamiento no deberíamos preocuparnos por tener una carga de 5.0 puesto que nuestro máximo teórico es 16.0. De la misma manera si en el primero de los valores observamos un valor de 16.0 tampoco deberíamos preocuparnos, puede deberse a un pico de procesamiento. Si esta carga se mantiene y llega a aparecer en el último de los valores, la media de carga de los últimos 15 minutos, habrá que ponerle remedio cuanto antes.

7.1.2 Memoria

La memoria RAM (*Random Access Memory*) es un recurso que nos permite almacenar información y acceder a ella sin necesidad de tener una tabla de particiones o un sistema de archivos. Precisamente este tipo de memoria recibe su nombre por la manera en que se accede a ella. El nombre, que proviene del inglés, son las siglas de memoria de acceso aleatorio.

Este tipo de memoria es mucho más rápido que los métodos de almacenamiento magnéticos o de estado sólido. El precio por Megabyte es también mayor caro que el de los discos duros. Además, plantea otro problema, la tecnología de la memoria RAM es muy rápida pero se basa en condensadores que se cargan y se descargan (1s y 0s) para guardar la información. El hecho de que la tecnología use de

condensadores significa que si no reciben alimentación eléctrica, la información se degrada hasta perderse. Así pues, una vez apagado el ordenador el contenido de la memoria RAM desaparecerá.

Con estas especificaciones la mayoría de máquinas establecen una jerarquía para guardar los datos con los que deben trabajar. Los datos que deben perdurar entre reinicios de la máquina se guardan siempre en los discos duros, o para ser más exactos en memoria terciaria, no son solo discos duros si no cualquier dispositivo de almacenamiento duradero. Los datos con los que más está trabajando el procesador se guardan en la memoria caché que es una tipo de memoria todavía más rápido que la RAM y por tanto, más caro. Finalmente los datos con los que el sistema debe trabajar pero que no caben en la cache se guardan en RAM.

Para obtener información sobre el estado de la memoria en nuestro sistema podemos recurrir a `top`, como ya se vio en la sección 6.3 Viendo procesos en tiempo real, o podemos recurrir al comando `free` que nos dará una idea de cual es el estado actual.

`free -h` # Muestra el estado de memoria en un formato comprensible por humanos

`free -g` # Muestra el estado de la memoria en GB

`free -ht` # Muestra el total de memoria de nuestro sistema

`free` nos muestra por defecto una información muy similar a la que nos mostraba el comando `top`, en este caso lo hace de una manera un poco más comprensible si se le pasa la opción correcta. Además de mostrarnos el cómputo de la memoria ocupada y libre nos muestra también el cómputo de la memoria en buffers i en swap que debe añadirse al cómputo de memoria libre y de memoria ocupada. Si le

especificamos la opción –t el programa free hará este cálculo por nosotros y nos dará los totales de memoria libre y de memoria ocupada.

Igual que sucedía con el caso de la CPU, tener el 100% de la memoria ocupada significa que no vamos a poder destinar memoria a nuevos programas o a procesos que se encuentren en ejecución y reclamen más memoria, produciendo una ralentización en el sistema o incluso la posibilidad de que algunos procesos finalicen su ejecución al no poder solicitar nueva memoria. Si desarrollamos nuestras propias aplicaciones debemos ser especialmente cuidadosos con esto. La memoria mal gestionada en una aplicación puede llevar a tener errores conocidos como “perdidas de memoria” que pueden llevar a nuestro sistema a quedar congelado debido a la falta de memoria libre. Estos errores son muy difíciles de detectar.

7.1.3 Espacio de disco

Todo sistema operativo cuenta al menos con un disco duro, hoy por hoy suele ser habitual incluso, contar con más de uno. Este suele ser también uno de los recursos importantes en nuestras máquinas puesto que los discos duros son los que almacenan los datos que deben perdurar en el sistema y que suelen ser el principal motivo de disponer de un ordenador en casa o un servidor.

Juntando conceptos

Si volvemos a la sección 4.5 Sistema de ficheros, recordaremos que únicamente vamos a poder trabajar con aquellos dispositivos que

dispongan de un sistema de ficheros y que se encuentren montados en el equipo. Es decir, únicamente vamos a poder trabajar con aquellos sistemas que dispongan de un punto de montaje y de un sistema de ficheros indicado al kernel.

En este caso el valor del que debemos estar pendientes es la capacidad disponible. Suele ser algo en lo que todos nos fijamos cuando compramos un disco duro y la herramienta `df` nos va a permitir conocer el estado en tiempo real.

`df -h`

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda1	322M	133M	173M	44%	/
tmpfs	943M	0	943M	0%	/lib/init/rw
Udev	938M	160K	938M	1%	/dev
tmpfs	943M	0	943M	0%	/dev/shm
/dev/sda2	228M	16M	201M	8%	/boot
/dev/sda3			58G	191M	55G 1%
/home					

Aunque la información que nos da el comando `df` es muy útil, no nos aporta información sobre donde estamos agotando los recursos de almacenamiento de los que disponemos. Es decir, no sabemos en que lugar de `/` tenemos archivos que ocupen un 44% del espacio disponible. Con un 44% de carga no es crítico disponer de esta información. A pesar de lo cual, se debe ser cuidadoso con la capacidad puesto que puede llegar a suponer un grave problema. En los sistemas Linux constantemente se escriben logs, al intentar autocompletar también se escriben ficheros temporales y al intentar ejecutar comandos se escribe en el history como veíamos en la sección

5.4 del capítulo sobre bash. Esto quiere decir que si llegamos a llenar la partición montada en la raíz del sistema es muy posible que nos encontremos con un sistema con el que es bastante difícil, o casi imposible, trabajar. Por ello debemos ser cuidadosos con el espacio restante en disco. En estos casos, para saber donde estamos gastando nuestro espacio en disco, podemos recurrir a la herramienta du.

du *

Nos dará lo que ocupa cada uno de los elementos del directorio actual. Sin embargo, el comando du no nos va a mostrar el tamaño completo de los directorios, si no el de todos sus elementos. Si queremos obtener el tamaño total de los directorios dentro del directorio actual deberemos emplear el comando

du -s *

De esta manera sumará el tamaño del contenido de los subdirectorios para mostrarnos finalmente este valor. Al igual que ls o df, du cuenta también con la opción -h que permite mostrar la salida en un formato comprensible para los humanos. Además, permite especificar un fichero o directorio concreto, tan solo se debe substituir el asterisco "*" por la ruta del fichero del que se quiere calcular el tamaño.

du -hs *

Esta herramienta puede ser tediosa de usar, pero se puede automatizar su uso mediante scripts. Hay que tener en cuenta que en algunas ocasiones puede parecer que nuestro sistema Linux se ha quedado bloqueado puesto que du toma un tiempo considerable de

ejecución, sobre todo si se trata de ficheros de gran tamaño.

7.2 Memoria y fallos de página

La memoria en nuestro sistema, igual que los ficheros, obedece una jerarquía. Para gestionar la memoria los procesadores cuentan con unas unidades conocidas como MMU (Memory Management Unit) o lo que es lo mismo: unidades de gestión de memoria. Estas MMU son las encargadas de traducir las direcciones de memoria virtual a la que acceden los procesos en direcciones de memoria real. Los detalles de porque se hace esta separación entre memoria virtual y real no entran en el ámbito de este texto. Si se siente curiosidad es posible encontrar explicaciones en muchos textos de introducción a la informática o sobre fundamentos de computadores.

Para facilitar el trabajo a las MMU el kernel, es decir el núcleo de nuestro sistema Linux, divide la memoria a la que tienen acceso los programas en trozos que reciben el nombre de páginas. Estas páginas mantienen una estructura mediante las tablas de páginas y la MMU traduce las direcciones de memoria virtual a memoria física basándose en las tablas y su estructura.

Un proceso no dispone desde el inicio de todas las páginas de memoria que va a necesitar a lo largo de su ejecución si no que estas se le van asignando al programa según las necesita. Ya no solo cuando este requiere más memoria si no que cuando se deben leer datos del disco la memoria trabaja de una manera similar. No se lee todo un fichero de golpe si no que este se lee según el tamaño de las páginas de memoria. Esto facilita que si se accede a datos consecutivos en disco estos ya se encuentren en memoria en el segundo acceso dado que no se ha leído únicamente un dato.

Cuando un proceso solicita una página de memoria y la página no está disponible se produce lo que se denomina un fallo de página. Aquellos que se dedican a optimizar el código de algunos programas donde el tiempo es crítico tratan, entre otras cosas, de minimizar el número de fallos de página, puesto que estos siempre suponen una penalización en cuanto al tiempo de ejecución del programa.

Encontraremos dos tipos de fallos de memoria:

- **Fallos menores:** Suceden cuando el proceso solicita una página que ya se encuentra en memoria pero no está localizada por la MMU. Un ejemplo muy claro de esta situación lo encontramos cuando el proceso solicita una nueva página de memoria para guardar por ejemplo una estructura de datos o cuando se solicita una página que se había solicitado previamente pero la MMU no dispone de suficiente espacio para mantener todas las traducciones activas. En estos casos la penalización de tiempo no es especialmente grave y muchas veces son difíciles o imposibles de evitar.
- **Fallos mayores:** Este es un caso más grave puesto que se produce cuando la página no se encuentra en memoria. Es el caso, por ejemplo, de tener que leer datos de un fichero o de un dispositivo de almacenamiento más lento. En este caso el fallo es más grave puesto que ir a disco duro a buscar información, por ejemplo, suele suponer una

demora de tiempo importante. Este tipo de fallos se darán también cuando la máquina se empiece a quedar sin memoria disponible y empiece a usar el espacio de swap para leer o escribir de él. En estos caso sufriremos una penalización en el rendimiento de los procesos bastante importante.

El comando `time` delante del programa o comando a ejecutar nos dará información sobre el uso de CPU del programa ejecutado, el tiempo que ha tardado el programa en terminar su ejecución y sobre los fallos de memoria de este:

```
time ./myScript.sh
0.00user      0.00system      0:00.08elapsed      0%CPU
(0avgtext+0avgdata 3318maxresident)k 648 inputs+0outputs
(3major+234minor)pagefaults 0swaps
```

Como se puede ver, el comando `time` nos está indicando que se han producido 3 fallos mayores y 234 menores. Lo más probable es que los 3 fallos mayores se hayan producido al ir a buscar el código del script a disco así que, probablemente, si volviésemos a ejecutar el mismo comando no obtendríamos los fallos mayores puesto que ya tenemos el código cargado en memoria.

También podemos ver los fallos de página de los procesos que se están ejecutando en nuestro sistema recurriendo al comando `ps`.

```
ps -o minflt,majflt,pid
```

El comando `top` también es capaz de mostrarnos en tiempo real los fallos de página, aunque en el caso de `top` no nos mostrara los

fallos menores. Para poder ver los fallos mientras se está en el modo interactivo de top se debe presionar la tecla 'f' y la tecla 'u' a continuación. Deberíamos ver una nueva columna de nombre nFLT que nos muestra los fallos de página mayores. Sobre todo en procesos que hacen un uso intensivo de memoria y donde el tiempo es crítico este valor nos aporta mucha información.

7.3 Monitorización de ficheros

El número de ficheros abiertos es otra de las cosas a tener muy en cuenta en los sistemas Linux. En primer lugar porque es posible limitar el número de ficheros que un usuario puede tener abiertos al mismo tiempo y en segundo lugar, por lo importantes que son los ficheros en los sistemas Linux. Hemos visto en el capítulo 2. Ficheros y jerarquía de directorios, la importancia de los ficheros, es por tanto interesante ser capaz de saber cual es su estado en nuestros sistemas.

Para esta tarea disponemos de la herramienta lsof. Este comando permite monitorizar no solo los ficheros regulares si no toda clase de ficheros: pipes, directorios, discos, elementos de red y, en general, cualquier tipo de dispositivo.

lsof

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
init	1	root	cwd	DIR	8,1	4096	2	/

Si ejecutamos el comando lsof no vamos a ver, únicamente, la información que se muestra en el ejemplo si no mucha más. La cantidad de información que nos va a ofrecer es considerable puesto que nos

mostrará información de los todos los procesos abiertos. La información que nos ofrece es la siguiente:

- **COMMAND:** El comando correspondiente al proceso abrió el fichero.
- **PID:** El identificador único del proceso.
- **USER:** El usuario que ejecuta el proceso.
- **FD:** Este valor nos puede mostrar dos cosas distintas. Podrá mostrarnos el descriptor del fichero abierto, es decir, un número que permite al proceso identificar el fichero o trabajar con él, o bien puede mostrarnos la utilidad que se le va a dar al fichero. A continuación se muestra algunos valores de los que puede tomar:
 - **cwd:** Directorio de trabajo actual.
 - **txt:** Fichero de texto.
 - **mem:** Fichero de memoria.
 - **mmap:** Fichero de dispositivo cargado en memoria.
 - **<número>:** en el caso que tome un valor numérico nos encontramos en el primero de los casos que definíamos más arriba.
- **TYPE:** Especifica el tipo de fichero. Si recordamos la sección 2.1 Ficheros y directorios, nos vendrán a la mente algunos tipos de ficheros. Estos son algunos de los valores que puede tomar este campo:

- **REG:** Fichero regular.
- **DIR:** Directorio.
- **FIFO:** Normalmente se tratará de un PIPE o tubería.
- **CHR:** Un dispositivo de lectura por caracteres.
- **DEVICE:** Los números que identifican el dispositivo. Recordemos que el sentido de estos números se comentaba en el capítulo 3. Dispositivos.
- **SIZE:** El tamaño del fichero. Los directorios suelen tener un tamaño de 4096 bytes puesto que no se muestra el tamaño de los contenidos del directorio.
- **NODE:** El inode relacionado con el fichero. Recordemos la sección 4.6 Inodos y Ficheros.
- **NAME:** El nombre del fichero.

Igual que otros muchos programas, lsof permite una gran cantidad de opciones. A diferencia de otras utilidades que son más genéricas entre distintos sistemas Linux, lsof tiene una gran cantidad de revisiones que pueden variar entre las distintas distribuciones de Linux, por eso lo mejor para asegurarse de que opciones tenemos disponibles o cuales son válidas es hacer uso de la opción `-h`. Algunas de las más útiles son:

- **-p pid:** Permite ver los ficheros abiertos para un pid concreto.
- **-b:** Indica a lsof que evite funciones de entrada salida que puedan dejar el proceso bloqueado a la

espera de una entrada o salida.

- -l: Evita la conversión de identificadores de usuarios a nombres. En el campo USER mostrará un número. Puede ser útil cuando hay problemas con la resolución de los nombres de usuario.

7.4 Monitorización de entrada/salida

La monitorización de la entrada/salida, en lo que a escrituras en disco se refiere, es un valor importante a tener en cuenta en nuestro equipo. Normalmente al trabajar con programas de terceros no vamos a poder hacer nada o casi nada por reducir el tiempo de entrada salida. Si se da el caso que trabajamos con herramientas o programas propios debemos tener en cuenta que la lectura o escritura en los dispositivos de almacenamiento permanente es una de las tareas que más penaliza a nivel de tiempo. La monitorización de estos elementos responde también a la necesidad de tener bajo control lo que pasa en nuestro sistema Linux.

iostat es la herramienta adecuada para esto. Es posible que en algunas distribuciones de Linux esta herramienta no venga instalada por defecto, sin embargo no nos será difícil encontrarla puesto que se encuentra en los mismos repositorios.

La salida de iostat depende, como en la mayoría de ocasiones, de los parámetros con los que se invoque. En el caso de iostat la salida puede variar de una distribución de Linux a otra, no en los datos que muestra si no en aquellos que muestra por defecto cuando se invoca sin ningún parámetro. Antes de mostrar que valores nos da iostat aclarar que este comando no suele mostrar, por defecto las distintas particiones. Si es el caso, la opción -p ALL nos permitirá

ver los distintos dispositivos de almacenamiento y todas sus particiones.

Algunos de los posibles valores de salida y sus significados son los siguientes:

- **tps:** Indica el número de transferencias por segundo para el dispositivo. Las transferencias tienen un tamaño indeterminado.
- **Blk_read/s, kB_read/s, MB_read/s:** Indica la cantidad de datos leídos del dispositivo por segundo. El primero indica bloques por segundo, el segundo indica kilo bytes por segundo y el último de ellos indica mega bytes por segundo.
- **Blk_wrtn/s, kB_wrtn/s, MB_wrtn/s:** Igual que el anterior, con la diferencia de que hace referencia a escrituras. Indica la cantidad de datos escritos en el dispositivo por segundo.
- **Blk_read:** Muestra el total de bloques leídos del dispositivo. Igual que en los casos anteriores existen también las columnas para kilo bytes y mega bytes.
- **Blk_wrtn:** Muestra el total de bloques escritos en el dispositivo. Cuenta con los modificadores para kilo bytes y mega bytes igual que los anteriores.
- **rrqm/s:** Muestra el número de solicitudes de lectura enviadas al disco por segundo. Las lecturas de un dispositivo no se realizan directamente si no que se encolan una serie de peticiones que se van

resolviendo según sea más adecuado. Como se resuelven las peticiones de escritura o lectura a los distintos dispositivos de almacenaje es una tarea del kernel y no entra en el alcance de este texto. Si se siente curiosidad es posible recurrir a Internet o a textos de fundamentos de computadores.

- **wrqm/s:** Muestra el número de solicitudes de escritura enviadas al dispositivo por segundo.
- **r/s:** Muestra el número total de peticiones de lectura que el dispositivo ha sido capaz de atender después de que estas fuesen encoladas. Al igual que pasa con las transferencias no hay un tamaño fijado para las peticiones.
- **w/s:** Muestra el número total de peticiones de escritura que el dispositivo ha sido capaz de atender por segundo después de que estas fuesen encoladas.
- **await:** Esta medida nos puede ser bastante útil en determinadas ocasiones. Muestra el tiempo medio que tarda el dispositivo en atender cada una de las peticiones que se le hacen. No distingue entre lectura o escritura.

`iostat` muestra, además de los valores anteriores para cada dispositivo, información sobre el uso de CPU como hacen otros comandos que se han visto anteriormente. El gran valor añadido que ofrece reside, no obstante, en la información que aporta sobre los

procesos de entrada/salida. Es por eso que no se comentan los significados de los otros valores de salida, que por otro lado deberían sernos familiares de otros comandos.

Aunque la utilidad de iostat es indiscutible, la información que nos ofrece está orientada a los dispositivos. Esto puede ser inconveniente en ocasiones en la que nuestro interés se centre en un proceso o grupo de procesos concreto. Es para ello que tenemos el comando iotop. Este programa no viene instalado por defecto en todas las distribuciones de Linux, sin embargo es muy fácil encontrarlo para cualquier distribución que tengamos.

iotop al igual que el resto de comandos de monitorización que hemos visto nos muestra la información por columnas. En este caso, esta es alguna de la información que nos ofrece:

- **TID:** Es un identificador único de thread. A diferencia de la gran mayoría de programas de nuestro sistema Linux, iotop nos muestra estadísticas por thread en lugar de hacerlo por procesos.
- **PRIO:** La prioridad con que las tareas de entrada/salida se llevan a cabo. Es una medida tomada en cuenta por el kernel de Linux. Puede tomar básicamente alguno de los valores siguientes:
 - **be:** best effort que en inglés significa mejor esfuerzo. El kernel tratará de posicionar estos trabajos de entrada/salida lo mejor posible.
 - **rt:** real time o en español tiempo real. Esta prioridad es considerada la más alta. El kernel

tratará de ejecutar la operación antes que cualquier otra.

- **idle:** La menor de las prioridades. El kernel solo dará prioridad a estas tareas de entrada/salida cuando no haya ninguna otra tarea pendiente.
- **DISK READ:** Nos da la cantidad de datos por segundo leídos por el proceso o thread. Hay que destacar que en este caso, por defecto, no disponemos de la información sobre el dispositivo del que se lee.
- **DISK WRITE:** Nos da la cantidad de datos por segundo escritos por el proceso o más bien por el thread.
- **COMMAND:** Nos indica a que comando pertenece el thread. Nos brinda una buena oportunidad de identificar el proceso propietario del thread que realiza el consumo.

iotop nos da, por defecto, más información de la mostrada aquí, sin embargo después de haber visto los otros comandos de monitorización similares no debería sernos difícil entender que información nos están ofreciendo las otras columnas.

8. Usuarios

Con los usuarios, completamos el último de los puntos claves de cualquier sistema operativo moderno y, más concretamente, de cualquier sistema Linux moderno. Un usuario en un sistema Linux es cualquier entidad capaz de ejecutar procesos y poseer ficheros. Debemos recordar que siempre que hablemos de sistemas Linux estaremos hablando de sistemas multiusuarios. Una correcta gestión de los usuarios puede llegar a suponer una diferencia muy importante respecto a la seguridad de nuestros sistemas con una gestión nula o inexistente.

8.1 El fichero /etc/passwd

El fichero `/etc/passwd` es un fichero de configuración del sistema, como podemos deducir por la ruta en la que se encuentra. Este fichero contiene gran parte de la información referente a los usuarios que hay en cualquier sistema Linux. Dado que el sistema de usuarios y jerarquías es complejo, no es el único fichero de configuración que atañe a los usuarios.

En este caso se trata de un fichero de texto donde cada línea muestra información de un usuario distinto. Una línea típica del fichero `/etc/passwd` es como sigue:

usuario:x:uid:gid:comentario:home-directory:shell-de-inicio

- **usuario:** Nombre de usuario. Con él se puede acceder al sistema.
- **x:** Antiguamente este campo albergaba la contraseña del usuario. Actualmente está obsoleto

y se mantiene únicamente por motivos de compatibilidad. Aunque se trata de un campo en desuso, dejarlo en blanco implica que el usuario no tendrá contraseña.

- **uid:** Se trata del identificador único del usuario.
- **gid:** El identificador único del grupo al que pertenece el usuario.
- **Comentario:** Un comentario sobre el usuario.
- **Directorio Home:** El directorio de trabajo del usuario. Una vez entre en el sistema el usuario tendrá este directorio como directorio raíz.
- **Shell-de-inicio:** El terminal con el que trabajará el usuario cuando entre al sistema. En este caso podría ser, por ejemplo: `/bin/bash` que es el que hemos visto. Es posible dejar este campo vacío, en ese caso el usuario existirá en el sistema pero será incapaz de conectarse remotamente o de autenticar-se en local.

Suele ser habitual en todos los sistemas Linux contar con una cuenta con nombre de usuario `root` que es la que goza de permisos de administración completos pero el nombre `root` no define una cuenta de administración por si mismo. Lo importante en estos casos no es el nombre de usuario si no el identificador. Una cuenta con `uid=0` será en cualquier sistema Linux la cuenta de administración, ya tenga como nombre de usuario `root` o `pepe`.

Pese a que puede parecer una tontería, resulta interesante

realizar una copia de seguridad del fichero `/etc/passwd` al trabajar con él. Si por error cambiásemos algún usuario el error no sería muy grave. Por contra, si por cometiésemos un error en la línea referente al usuario `root` es más que probable que no fuésemos capaces de arrancar el sistema. Aunque se ha dicho que el usuario con `uid=0` puede tener cualquier nombre, para la mayoría de distribuciones es imprescindible mantener el nombre de usuario de `root` para este `uid`. Así pues no es recomendable cambiar esta línea en el fichero `/etc/passwd` si no se sabe perfectamente lo que se está haciendo. Si se quieren dar a otro usuario distintos permisos de administración existen otras maneras de hacerlo que veremos en la sección 8.6.

8.2 El fichero `/etc/shadow`

El fichero `/etc/shadow` es otro de los ficheros de configuración asociado a los usuarios en nuestros sistemas Linux. En este caso, el fichero contiene información acerca de los usuarios y de las contraseñas de estos. Se trata de un fichero de texto con una línea por usuario con el siguiente formato:

`usuario:password:lastchg:min:max:warn:inactive:expire:`

- **usuario:** Nombre de usuario. Debe existir en el fichero `/etc/passwd`
- **password:** La contraseña del usuario. Normalmente se encuentra cifrada por lo que no es reconocible a simple vista.
- **lastchg:** La última fecha de modificación de la contraseña expresada en segundos desde el 1/1/1970

- **min:** La duración mínima, en días, de la contraseña. Es decir cuanto tiempo debe pasar desde el último cambio de contraseña hasta que se permita al usuario volver a cambiarla. El usuario root o con UID=0 ignora esta restricción.
- **max:** Cada cuantos días se fuerza al usuario a cambiar la contraseña.
- **warn:** Con cuantos días de antelación se avisará al usuario que debe cambiar la contraseña.
- **inactive:** Indica cuantos días pueden pasar desde que expira la validez de la contraseña hasta que la cuenta es bloqueada. Es decir, si toma el valor 0, si el día que caduca la contraseña no se cambia por una nueva, la cuenta de usuario quedará bloqueada. Si se le da a este campo el valor -1, la cuenta no quedará nunca inactiva por mucho que expire la contraseña. En este caso siempre se permitirá al usuario entrar a cambiarla. -1 es la opción por defecto. Por lo que a menudo vemos un campo vacío.
- **expire:** Fecha en que la cuenta expira. Se representa como un valor en días desde el 1/1/1970. Esto permite definir cuentas de usuario que serán válidas sólo durante un periodo de tiempo concreto.

El último campo de las líneas del fichero /etc/shadow es un

campo reservado por lo que usualmente las líneas en este fichero suelen terminar con ':'. Esto no es porque deban terminarse así, si no porque el último campo no toma valor alguno.

En cuanto a las contraseñas almacenadas en este fichero se debe tener en cuenta que están cifradas. Incluso cuando esto dificulta que den información, el fichero `/etc/shadow` puede ser crítico para la seguridad de nuestros sistemas. Es por ello que deben ajustarse cuidadosamente los permisos de este fichero para que sean los más restrictivos posibles sin llegar a ser un impedimento.

Un '*' en el campo de contraseña indica que la cuenta se encuentra deshabilitada, mientras que un campo vacío puede llegar a suponer un grave problema de seguridad puesto que indica que la cuenta no requiere contraseña. Usualmente en la mayoría de los sistemas Linux, la contraseña se almacena mediante una función de resumen o función hash irreversible. Sin embargo, esto no debe significar que no se tomen medidas de seguridad. Hay ataques que pueden hacer uso del fichero `/etc/shadow` para obtener las contraseñas de nuestros usuarios.

8.3 Grupos de usuarios

Linux dispone, aparte de usuarios, de grupos. Los grupos no son más que una manera de poder asignar permisos de manera más sencilla a un conjunto de usuarios que comparten características. Resulta más fácil de entender quizás con un ejemplo. Supongamos un colegio, lo primero que nos viene a la mente es que habrá dos tipos de usuarios: alumnos y profesores. Podemos definir entonces dos grupos de usuarios: alumnos y profesores, de manera que nos sea fácil definir los permisos para cada grupo. De esta manera podemos permitir a todo

un grupo de usuarios acceder a un fichero mientras que les negamos el acceso a otro grupo.

En los sistemas Linux un usuario puede pertenecer a tantos grupos como nos convenga, sin ningún límite concreto. Por otra parte, todo usuario debe pertenecer como mínimo a un grupo. Si en el momento de crear un usuario no le asignamos ningún grupo la mayoría de sistemas Linux, si no todos, crearán un grupo específico para dicho usuario. Los grupos creados tendrán, en la mayoría de sistemas Linux, un identificador de grupo superior al 500, puesto que por defecto los grupos con GID inferior al 500 están reservados para grupos del sistema.

Para gestionar los grupos a los que pertenece un usuario podemos recurrir al comando `usermod` que nos permitirá añadirlo a uno o varios grupos o eliminarlo de estos. `Usermod` nos da varias opciones. Algunas de las más útiles son las siguientes:

- **-a:** Permite añadir a un usuario a grupos suplementarios. Debe usarse únicamente con la opción `-G`.
- **-G <lista de grupos separados por comas>:** Permite especificar uno o varios grupos suplementarios que añadir al usuario. Si no se especifica la opción `-a` los grupos que se pongan a continuación de `-G` replazarán los grupos actuales del usuario.
- **-g:** El nombre del grupo al que pertenecerá el usuario de ahora en adelante.

No resulta difícil encontrar todas las opciones en los

manuales. Aún así, como su nombre indica, sirve para cualquier tarea que implique una modificación de las características de los usuarios. Veamos un par de ejemplos para que quede más claro.

#Añade a pepe a los grupos de alumnos y profesores

usermod -aG alumnos, profesores pepe

#Modifica los grupos de pepe por administración y profesores

usermod -G administracion,profesores pepe

#Cambia los grupos de pepe por el de profesores

usermod -g profesores pepe

Toda la información de los grupos se guarda en un fichero de configuración del sistema llamado `/etc/group`. Este es un fichero de texto donde cada línea corresponde a un grupo.

grupo:x:gid:lista-usuarios

- **grupo:** El nombre del grupo.
- **x:** En desuso actualmente. Se mantiene únicamente por motivos de compatibilidad. Igual que pasaba con el fichero `/etc/shadow` se usa para representar la contraseña. Si este campo queda vacío no se requerirá contraseña.
- **gid:** El identificador del grupo.
- **lista-usuarios:** Los usuarios pertenecientes al grupo separados por ','.

La contraseña en el caso de los grupos serviría para permitir el acceso al grupo a un usuario que no forma parte de él. En este caso el usuario pasaría a tener los mismos privilegios que tiene el grupo

además de los permisos que él mismo tiene. En la mayoría de distribuciones tanto el campo contraseña del `/etc/group` como el fichero `/etc/gshadow` están en desuso y se recurre al bit GUID para dar los permisos de un grupo a un usuario concreto, recordemos la sección 2.3 Permisos y propiedades de ficheros. Indistintamente de si va a usarse o no, es bueno conocer de la posibilidad de proteger los grupos con una contraseña y de permitir que usuarios se incorporen temporalmente a grupos. Si un usuario va a poder tener acceso temporal a un grupo suele ser habitual añadirlo al grupo en lugar de recurrir a estos sistemas.

8.3.1 Trabajando con los grupos

Aunque modificando los ficheros `/etc/group` y `/etc/passwd` podríamos trabajar completamente con los grupos, es aconsejable realizar los cambios mediante los programas, comandos, que los sistemas Linux nos ofrecen para ello.

En el caso concreto de los grupos hemos visto como añadir grupos a un usuario o como quitárselos. Pero sucederá en muchas de las ocasiones, o en todas, que necesitaremos crear nuevos grupos o gestionar los existentes. Para ello tenemos una serie de herramientas a nuestra disposición.

8.3.1.1 Crear un grupo

Podemos crear un grupo haciendo uso del comando *groupadd*. Este comando normalmente requiere que se le pase como parámetro el nombre del grupo a crear. El resto de datos los coge él mismo según los grupos que ya existan en nuestro sistema y los parámetros que el sistema tenga por defecto.

`groupadd profesores`

Existen parámetros para este comando que modifican su comportamiento, algunos de los más interesantes son:

- **-r:** Crea un grupo del sistema. Es decir con un GID inferior a 500. Únicamente el súper usuario puede hacerlo.
- **-g <GID>:** Permite especificar el ID de grupo que se le quiere asignar al grupo. Si este ya existe, el comando devolverá un error.
- **-f:** Obliga al programa a terminar con una salida exitosa. Si se especifica el comando `-g` con un identificador de grupo ya existente el sistema escogerá el siguiente identificador libre.
- **-o:** permite especificar un grupo con un GID duplicado. Cuidado con esta opción, puede devolvocar en comportamientos no deseados o inesperados.

8.3.1.2 Eliminar un grupo

Eliminar un grupo es todavía más sencillo que crearlo. Para ello únicamente debemos invocar el comando *groupdel* especificando el nombre del grupo que se quiere eliminar.

`groupdel profesores`

Este comando no acepta parámetros. Sin embargo es importante considerar sus valores de salida puesto que nos indicarán si todo ha ido bien, o en caso contrario, que error se ha producido. Las posibles salidas son:

- **0:** Éxito. Todo ha ido bien. Suele ser una convención habitual en sistemas Linux devolver un 0 cuando todo va bien.
- **2:** La sintaxis del comando no es correcta. En este caso deberemos revisar que lo hemos escrito todo bien.
- **6:** El grupo especificado no existe.
- **8:** No se puede eliminar el grupo primario de un usuario. Este error nos está indicando que antes de borrar el grupo debemos borrar, si nos interesa, el usuario que tiene este grupo como primario.
- **10:** No se puede modificar el fichero de configuración de grupos. Suele ser un problema de permisos. Deberemos revisar de que permisos disponemos en el sistema.

En los sistemas Linux los grupos tienen un límite de usuarios. Una vez se supera el límite de usuarios para un grupo se crea un grupo nuevo con el mismo nombre, el mismo GID y la misma contraseña, si la tuviese. Hay que ser muy cuidadoso con este comportamiento ya que el comando `groupdel` podría funcionar de manera imprevista con grupos divididos como los que se acaban de describir.

8.3.1.3 Modificar un grupo

En este caso contamos con el comando `groupmod`. Este al igual que `groupadd` recibe varios parámetros para modificar su comportamiento y al igual que `groupdel` nos da varios códigos de salida para que entendamos que es lo que ha sucedido.

groupmod [opciones] profesores

En este caso los parámetros son muy importantes puesto que son los que van a definir el comportamiento del programa. Si no recibe ninguna opción el comando groupmod no hará nada sobre el grupo que le especifiquemos. En este caso algunos de los parámetros más interesantes son:

- **-n <nuevo nombre>**: Permite cambiar el nombre del grupo por uno nuevo.
- **-o**: Permite cambiar el gid del grupo por uno no único.
- **-g <gid>**: Permite cambiar el identificador del grupo. Si el identificador de grupo especificado ya existe y no se especifica la opción **-o** el comando nos devolverá un error.
- **-p**: Permite especificar una contraseña para el grupo. La contraseña debe proporcionarse cifrada mediante el comando crypt. La contraseña que se introduzca aparecerá por pantalla por lo que no es recomendable este mecanismo para cambiar la contraseña de los grupos. En su lugar se recomienda el uso del comando gpasswd explicado en la sección 8.3.1.4.

En función de si ha podido o no realizar la modificación especificada, el comando groupmod nos puede devolver uno de los siguientes valores.

- **0**: Éxito. Todo ha ido bien.

- **2:** La sintaxis del comando no es correcta.
- **3:** Argumento u opción inválida. Nos hemos equivocado al especificar alguna de las opciones. Probablemente se deba a un error tipográfico, con una sola letra es difícil equivocarse de otra manera.
- **4,6:** El grupo especificado no existe.
- **9:** El nombre especificado ya está en uso.
- **10:** No se puede modificar el fichero de configuración `/etc/group` o `/etc/gshadow`.

Como se puede observar, los errores mostrados son muy similares a los errores de `groupdel`. Esto es debido a un tema de consistencia, podemos confiar al trabajar con grupos que los números de los distintos errores serán los mismos.

8.3.1.4 Grupos y contraseñas

Es recomendable añadir o modificar las contraseñas de los grupos mediante el comando `gpasswd`. Normalmente el uso de contraseñas en los grupos no es necesario puesto que hay otras alternativas, sin embargo, es posible cambiar o especificar una nueva contraseña para un grupo mediante este comando.

`gpasswd grupo`

Si no se le especifican opciones se cambiará la contraseña del grupo o se especificará una nueva en caso que no tuviese. Si el grupo no está protegido por contraseña, podremos trabajar con él siguiendo lo especificado en la sección 2.3 Permisos y propiedades de ficheros o podemos recurrir igualmente a `gpasswd`. Por supuesto, todas

las capas de seguridad que se le puedan añadir al sistema son bienvenidas, por tanto queda a discreción del administrador del sistema decidir si el compromiso facilidad de administración - seguridad justifica el uso de contraseñas para los grupos.

- **-a <usuario>**: Añade un usuario al grupo especificado
- **-d <usuario>**: Elimina a un usuario del grupo especificado.
- **-r**: Elimina la contraseña del grupo
- **-A <administrador1>, ..., administradorN**: Añade una lista de administradores al grupo
- **-M <usuario1>, ..., usuarioN**: Permite especificar una lista de miembros del grupo

Únicamente las opciones A y M pueden ser especificadas al mismo tiempo. El resto de ellas no pueden ser combinadas. Los administradores de un grupo tienen potestad para añadir o eliminar miembros de un grupo, por lo que se debe ser cuidadoso al combinar ambas opciones.

Al usar los comandos anteriores, se debe tomar en cuenta que el uso de estos se puede ver modificado por los valores por defecto respecto a la configuración de usuarios que especifique cada sistema y que pueden ser distintos en cada distribución de Linux. Estos valores son configurables en todos los sistemas Linux aunque es posible que las configuraciones no se encuentren siempre en el mismo fichero o directorio. Más adelante hablaremos de como cambiarlas.

8.4 Creando y borrando usuarios

Aunque al instalar nuestro sistema Linux se crearán algunos usuarios que el propio sistema para trabajar, es habitual que queramos crear nuestros propios usuarios. Por ejemplo, algunas distribuciones ya se encargan de crear un usuario genérico con nuestro propio nombre, en el momento de realizar la instalación. Aún así, suele ser habitual tener la necesidad de crear nuevos usuarios o, incluso cuando estos ya no nos van a ser de utilidad, borrarlos. Como en la mayoría de comandos en Linux, recurrir al inglés nos va a dar unas indicaciones bastante acertadas de como crear y borrar usuarios. Lo haremos con los comandos `useradd` y `userdel`.

Empecemos por el añadido de usuarios. En este caso tendremos multitud de opciones para este comando que nos permitirán una flexibilidad casi absoluta al trabajar con usuarios. Sean cuales sean los parámetros que especifiquemos al crear un usuario el procedimiento siempre funciona, más o menos, como sigue:

1. Se editan los ficheros `/etc/passwd`, `/etc/shadow`, `/etc/group` y `/etc/gshadow`
2. Se crea el directorio raíz del usuario también conocido como el `home` del usuario
3. Se asignan los permisos y se define al usuario como propietario de su directorio `home`.

Así pues lo que necesitamos es invocar el comando `useradd` y el hará el trabajo de añadir el usuario por nosotros.

`useradd pepe`

`useradd` tiene algunos parámetros que modifican su comportamiento. Algunas de las opciones más relevantes se muestran

a continuación:

- **-D:** Usa los valores por defecto del sistema para crear una nueva cuenta de usuario. No suele ser habitual emplear esta opción.
- **-b <directorio>:** Permite especificar el directorio base para el directorio raíz del usuario. Al nombre especificado se le añade el nombre de usuario que se está creando. Este valor es /home por defecto. Es decir, si este parámetro toma el valor /var para un usuario. Este tendrá su directorio raíz en /var/usuario/.
- **-m:** Crea el directorio raíz del usuario si este no existe. Si se especifica la opción -b y no se especifica esta opción el directorio raíz del usuario debe existir en el equipo u obtendremos un error.
- **-d <directorio>:** Especifica el directorio raíz del usuario. Lo habitual es añadir el valor especificado al directorio base. La ruta resultante no tiene por que existir. Si no existe, sin embargo, el directorio no se creará.
- **-e <fecha expiración>:** Permite especificar la fecha en que la cuenta de usuario expirará. La fecha debe expresarse en formato AAAA-MM-DD para que sea tomada en cuenta.
- **-g <GID>:** Permite especificar el grupo principal para el usuario.

- **-p <contraseña>:** Permite especificar la contraseña para el usuario. Se debe proveer la contraseña cifrada mediante el comando crypt. No es recomendable sin embargo cambiar la contraseña mediante este parámetro. Se recomienda el uso del comando passwd en su lugar una vez el usuario ya esté creado.
- **-s <shell>:** La shell o terminal que usará el usuario al autenticarse en el sistema. Suele ser común no especificar un terminal concreto y dejar que el sistema ponga en este campo el shell que usa la distribución por defecto.
- **-r:** Crea una cuenta de usuario del sistema. Las cuentas del sistema no contienen información temporal, no expiran. Además el UID de una cuenta del sistema usualmente estará por debajo de 1000, asumiendo, claro está, que no se han cambiado los valores por defecto del sistema.

Así pues si se quiere crear un usuario se procederá de una manera similar a la siguiente:

#Crea el usuario y el directorio raíz del usuario pepe

useradd -m pepe

#Añade una contraseña al usuario pepe

passwd pepe

Todos los valores que no son especificados en el comando useradd se toman de los valores por defecto del sistema. Estos valores

se almacenan en unos ficheros que veremos más adelante. Algunos de ellos es posible cambiarlos mediante el propio comando `useradd`. Si se especifica la opción `-D` seguida de alguna de las opciones que este comando permite y sin especificar un nombre de usuario lo que estaremos haciendo es actualizar los valores por defectos de nuestro sistema Linux.

Al igual que sucedía en el caso de los grupos, el comando `useradd` nos dará una salida u otra en función de si todo ha ido bien o si por contra, se ha producido algún error. Para conocer los posibles valores que nos puede devolver este comando, o cualquier otro, así como la lista completa de parámetros podemos recurrir a los manuales del sistema:

`man useradd`

8.4.1Borrar usuarios en sistemas Linux

De la misma manera que creamos usuarios tenemos el comando `userdel` para eliminarlos una vez ya no son útiles. El borrar un usuario funciona más o menos a la inversa de su creación. Si se borra la información de usuario del fichero `/etc/passwd` y del fichero `/etc/shadow` este deja de existir a ojos del sistema. Por otra parte, estos ficheros no contienen la totalidad de la información referida al usuario. Por ello es mejor recurrir al comando `userdel`.

`userdel usuario`

Si no se especifica ningún parámetro el comando `userdel` únicamente borrará el usuario de nuestro sistema Linux. El borrado de un usuario fallará si este se encuentra conectado al sistema. Los diferentes parámetros nos permitirán modificar este comportamiento.

Estas son algunas de las opciones de las que disponemos:

- **-f:** Un parámetro muy peligroso. Borra el usuario aunque este se encuentre conectado al sistema. Además borra también el directorio raíz del usuario y todos los ficheros que hay en su interior. Borra también el directorio que contiene el mail del usuario aunque este no pertenezca exclusivamente al usuario. Si existe un grupo con el mismo nombre que el usuario este también es borrado incluso cuando existan otros usuarios que pertenezcan al grupo. Hay que ser muy cuidadosos con este parámetro para evitar desgracias en nuestros sistemas.
- **-r:** Borra además del usuario su directorio raíz y todo el contenido de este.

8.5 Ficheros de inicialización

Al trabajar con usuarios o grupos hay una gran cantidad de información que podemos decidir especificar o no. En caso de no hacerlo se cogen los valores por defecto de los que dispone el sistema. Estos valores no se encuentran en el código fuente del kernel o del programen uso, si no que se encuentran en ficheros de configuración. No solo eso, si no que además, tal y como veíamos en la sección 2.2 Jerarquía de directorios, podemos encontrar los ficheros con los valores por defecto en el directorio /etc. Debemos tener en cuenta que al tener la información en ficheros podemos modificarla según nos convenga. Debemos hacerlo, no obstante, con precaución. Los usuarios son los agentes a través de los cuales el sistema ejecuta los

procesos. Si configuramos de manera inadecuada nuestro sistema es posible que algunos programas dejen de funcionar. Si el error lo cometemos en determinados ficheros y para determinados usuarios, por ejemplo en `/etc/passwd` para el usuario `root`, es posible que nuestro Linux ni siquiera arranque.

Aparte de los ficheros que definen los valores por defecto que nos permiten trabajar con usuarios, hay diversos ficheros que nos permiten definir la configuración del entorno de trabajo y del sistema para los distintos usuarios. En el caso de la configuración propia a cada usuario encontraremos duplicidad en los ficheros de configuración: encontraremos en `/etc` aquellos que sean genéricos para todos los usuarios y en el directorio raíz del usuario aquellos que deban afectarle solo a él. Más adelante veremos los ficheros de configuración propios para cada usuario, de momento nos centraremos en los que nos especifican valores por defecto para trabajar con usuarios y grupos.

- **`/etc/default/useradd`:** Contiene los valores por defecto que se usan en la creación de usuarios. Dependiendo de la distribución de Linux los valores que podemos encontrar en este fichero pueden variar. Lo importante, sea cual sea la distribución que usemos, es tener claro que podemos modificar los valores por defecto que nos ofrece el sistema según nos convenga.
- **`/etc/login.defs`:** Contiene algunos de los valores por defecto para la creación de usuarios. Los valores que contiene son distintos a los que

contiene el fichero anterior. En este caso están más orientados a modificar el comportamiento del comando `useradd` en lugar de proveer valores. De la misma manera, este fichero está enfocado a la configuración de la suite `shadow` de contraseñas.

- **/etc/skel:** No se trata de un archivo regular si no de un directorio. En este directorio se encuentra los esqueletos de los ficheros que todo usuario debe tener. Es decir en este directorio se encuentran los ficheros por defecto que se copiarán en los directorios pertinentes del directorio raíz del usuario.

8.5.1 El fichero `/etc/default/useradd`

Se trata de un fichero de texto que contiene información por defecto usada por el programa `useradd` al añadir un usuario. Los valores que contiene el fichero pueden ser reemplazados por los que queramos mediante el uso de parámetros o mediante la edición de este fichero. Cada atributo se especifica en una línea diferente en el formato `Atributo=valor`

Los espacios en blanco importan, por lo que se debe ser cuidadoso. La mejor práctica es que ninguna de nuestras rutas tenga espacios en blanco ni comillas en alguno de los nombres, si los tienen deberemos ser cuidadosos con estos caracteres porque causarán problemas. Algunos de los atributos que podemos especificar en este fichero son:

- **HOME:** Especifica el directorio base que se usará

para crear el directorio raíz del usuario. Si este campo no existe se usará /home

- **EXPIRE:** Define la fecha de expiración de la cuenta de usuario. Si no existe este atributo o tiene un valor vacío las cuentas no expiran nunca.
- **GROUP:** Aunque el comportamiento depende de otras variables, en caso de que no se cree un nuevo grupo para el usuario, este atributo permite especificar que grupo debe ser el grupo primario de los nuevos usuarios. Muy útil si todos los nuevos usuarios deben caer dentro del mismo grupo. Si no se especifica ningún valor toma 100 por defecto
- **SHELL:** Especifica que shell usarán los nuevos usuarios. Si no se especifica valor, el usuario por defecto no tendrá shell por lo que no podrá acceder al sistema.

8.5.2 El fichero /etc/login.defs

Se trata de un fichero de texto que define el comportamiento de la suite de contraseñas de los sistemas Linux y define, por tanto, el comportamiento del sistema con los nuevos usuarios así como con los que ya existen. Este fichero debe existir en el sistema, si no existe el sistema arrancará igual pero el comportamiento que se obtendrá es impredecible.

A diferencia del fichero de /etc/defaults/useradd, este especifica sus atributos y valores de forma distinta. Cada atributo se presenta en una línea y seguido de un espacio en blanco se presenta

su valor. Las líneas en blanco o que empiezan por '#' no son tomadas en cuenta.

Atributo valor

Los atributos que puede tener este fichero son muchos y no los comentaremos todos. Como otras veces, una sencilla búsqueda en Internet puede arrojar luz sobre ellos. Algunos de los más interesantes se presentan a continuación:

- **CREATE_HOME:** Puede tomar los valores yes o no (Sí o No en inglés). Especifica si se debe crear el directorio raíz para los nuevos usuarios.
- **DEFAULT_HOME:** Puede tomar los valores yes o no (Sí o No en inglés). Especifica si el usuario debe entrar al sistema dentro de su directorio raíz o en caso de no poder entrar en el directorio raíz debe entrar en /. En caso de poner este valor a yes el usuario entrara al sistema en el directorio / en caso de que no pudiese hacerlo en el suyo propio.
- **ENV_PATH:** Toma como valor una cadena de caracteres. Define el PATH que se usará para los usuarios cuando estos entran al sistema. Se puede sobrescribir mediante los ficheros de configuración propios de los usuarios. Para saber más sobre el PATH podemos volver a la sección 5.3 Path.
- **GID_MAX, GID_MIN:** Ambos pueden recibir como valor un número entero. Especifica el ID mínimo y

máximo del grupo al crear un usuario o un grupo nuevo. Por defecto toman, respectivamente, los valores 1000 y 60000. Un ID por debajo del mínimo especificado se considerará un grupo del sistema.

- **LOG_OK_LOGINS:** Puede tomar los valores yes o no (Sí o No en inglés). Permite especificar que se guarde información de las entradas exitosas al sistema. Resulta muy interesante tener este parámetro a yes como política de seguridad.
- **MAX_MEMBERS_PER_GROUP:** Especifica el número máximo de usuarios que puede tener un grupo. Una vez se supera el tamaño máximo se duplica el grupo. Esto puede llevar en algunas situaciones a comportamientos inesperados. Para más información podemos revisar la sección 8.3.1 Trabajando con Grupos.
- **PASS_MAX_DAYS, PASS_MIN_DAYS:** Permiten especificar el máximo y mínimo número de días que una contraseña puede ser usada por los usuarios. Este comportamiento puede modificarse mediante comandos.
- **UMASK:** Define la mascara para la creación de nuevos ficheros. Por defecto toma el valor 022. La creación del directorio raíz de los nuevos usuarios tendrá en cuenta esta mascara para asignar los permisos. Para más información podemos volver a

la sección 2.3 Permisos y propiedades de ficheros.

- **USERGROUPS_ENAB:** Puede tomar los valores yes o no (Sí o No en inglés). Si se pone a yes, useradd creará por defecto un grupo con el nombre del usuario al que dicho usuario será asignado. Si se pone a no al usuario se le asignará al valor definido por el atributo GROUP del fichero /etc/default/useradd

Ya conocemos como y que ficheros podemos cambiar para modificar los valores por defecto en nuestro sistema Linux. Aún así debemos ser cuidadosos con la seguridad de nuestro sistema y con los permisos de los usuarios puesto que debemos recordar que es posible modificar estos valores por defecto desde los distintos comandos de manipulación de usuarios y grupos.

8.6 Root, sudo y el fichero /etc/sudoers

En los sistemas operativos Linux “root” es usualmente el nombre para aquel usuario que posee permisos para realizar cualquier acción en el sistema. Es decir, es el nombre de la cuenta de administración. Con el usuario root podemos modificar cualquier tipo de fichero aunque no seamos los propietarios o no tengamos permisos para ello. De la misma manera, podemos asignar o cambiar permisos o propietarios a cualquier fichero. Dicho de otra forma, es una cuenta de usuario que nos permite hacer cualquier cosa que se nos ocurra.

Una vez hemos entrado en un sistema Linux podemos cambiar de usuario con el comando su (del inglés substitute user). Si simplemente invocamos su cambiaremos al usuario root. Si después de su especificamos el nombre de usuario cambiaremos al usuario

especificado. Si la cuenta del usuario a la que queremos cambiar está protegida por contraseña, el sistema nos solicitará la contraseña del usuario correspondiente. Hay una excepción para esta regla: si el usuario con el que estamos trabajando es el usuario root no se nos pedirá ninguna contraseña para cambiar a otro usuario distinto.

su

su usuario

Esto significa que es muy peligroso que un usuario consiga acceso como root a un sistema puesto que dicho usuario podrá hacer todo aquello que se le antoje, sin limitaciones. Por ello es importante proteger la cuenta de súper usuario o root con una contraseña segura e intentar trabajar lo menos posible con esta cuenta.

Si tenemos la necesidad de trabajar como root o de permitir que algunos usuarios trabajen como usuario root podemos recurrir al comando sudo (del inglés substitute user do). Este comando nos permite ejecutar programas y comandos como otro usuario, normalmente como root. Es posible que sudo no venga instalado en algunas distribuciones de Linux, sin embargo no es difícil de instalar por los medios habituales.

Como ya hemos dicho, el programa sudo permite ejecutar programas y comandos como root siempre que el usuario sea capaz de autenticarse, normalmente introduciendo su propia contraseña. En ocasiones, es posible que no nos interese darles a todos los usuarios privilegios de superadministrador. En algunas ocasiones nos convendrá otorgarle a un usuario permiso para actuar en nombre de un grupo, de otro usuario o realizar algunas tareas como root. Esto no significa, pese a lo dicho, que haya que darle a dicho usuario permisos

totales sobre el sistema. Es más, eso podría ser un importante fallo de seguridad. Es por eso que podremos configurar el comportamiento de sudo mediante el fichero de configuración /etc/sudoers

El fichero /etc/sudoers es un fichero de texto plano por lo que puede ser modificado con cualquier editor que usemos habitualmente. Pese a la posibilidad de usar un editor común y corriente para modificar este fichero es recomendable hacer uso del comando **visudo**.

visudo

De esta manera no solo podremos modificar el fichero de sudoers sin problema si no que además, al cerrar el fichero el programa comprobará que la sintaxis es correcta. Una mala sintaxis en el fichero de sudoers podría llevar a dejar el comando sudo inservible.

El fichero de sudoers tiene un aspecto similar al que se muestra a continuación:

#Lineas por defecto:

Defaults env_reset

Defaults mail_badpass

Defaults timestamp_timeout=5

Defaults

secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

#Lineas de usuarios:

root ALL=(ALL:ALL) ALL

#Lineas de grupos:

Las líneas por defecto especifican comportamientos de sudo. Como podemos observar todas las líneas que modifican el comportamiento empiezan por la palabra clave Defaults seguida de un

tabulador y el valor que debe tomar. Estos valores afectarán a todos los usuarios del fichero sudoers. Algunos de los valores que podemos encontrar son:

- **authenticate:** Puede tomar los valores de on y off. Si está activo obliga a los usuarios a autenticarse antes de usar cualquier comando con sudo. Por defecto está activo.
- **env_reset:** Limpia cualquier variable de entorno seteada antes de ejecutar el comando.
- **timestamp_timeout:** Permite especificar el tiempo que debe durar la sesión de sudo. Es decir, durante cuantos minutos el comando sudo no volverá a pedir la contraseña al mismo usuario para autenticarse. Este parámetro solo tiene sentido si authenticate se encuentra activo.
- **insults:** Si está activo el comando insultará a los usuarios que se equivoquen al introducir la contraseña. Por defecto se encuentra inactivo. La verdad es que, a parte del detalle gracioso, no aporta mucho.
- **log_input:** Si se activa, todas las entradas del usuario cuando emplee sudo quedarán registradas. Por defecto quedarán registradas en /var/log/sudo-io. Por defecto log_input está desactivada.
- **mail_badpass:** Si el usuario que intenta ejecutar el

comando con sudo no introduce la contraseña correcta se manda un correo electrónico al correo especificado en mailto indicando el error.

- **mail_always:** Se manda un correo a la dirección especificada en mailto cada vez que un usuario ejecuta el comando sudo.
- **mailto:** Permite especificar la dirección de a quien se deben mandar correos de aviso.

Para las opciones por defecto que pueden tomar los valores on y off, el comando sudo únicamente requiere escribir el nombre de la opción. Aún así es posible escribir la opción larga. Para el resto de opciones se debe especificar el nombre de la opción, un igual y el valor que debe tomar la opción.

Acontinuación siguen un conjunto de líneas del estilo:

```
root    ALL=(ALL:ALL) ALL
```

Estas son las líneas que verdaderamente definen el comportamiento del comando sudo para cada usuario. Vamos a ver que significa cada uno de estos valores:

- **root ALL=(ALL:ALL) ALL:** Usuario sobre el que se aplica la regla que queremos configurar.
- **root ALL=(ALL:ALL) ALL:** Host, es decir, nombre de la máquina en el que se aplicará la regla. A menos que usemos el fichero de /etc/sudoers para diversas máquinas al mismo tiempo, este siempre tomará el valor ALL.

- **root ALL=(ALL:ALL) ALL:** Indica en lugar de que usuario y grupo, en este orden se pueden ejecutar comandos. Si ponemos ALL podremos ejecutar comandos en substitución de cualquier usuario o grupo.
- **root ALL=(ALL:ALL) ALL:** Indica que comandos o programas puede ejecutar el usuario o usuarios que cumplan la regla. Este último parámetro puede modificarse con ciertas etiquetas. Dos de las más útiles son NOPASSWD, NOEXEC.
- **NOEXEC:** Impide ejecutar determinados comandos a un determinado usuario.
- **NOPASSWD:** Permite ejecutar determinados comandos sin que el usuario tenga que autenticarse.

Sudo permite además la creación alias para trabajar de manera más cómoda. De esta manera podemos definir varios comandos o usuarios para aplicar más tarde a nuestras reglas. Todo esto dentro del fichero /etc/sudoers. Los alias deben especificarse antes de las reglas y cada uno debe tener su tipo:

- **Cmnd_Alias:** Para los comandos.
- **User_Alias:** Para los usuarios.
- **Runas_Alias:** Usuarios con privilegios o usuarios del sistema.
- **Host_Alias:** Para los nombres de equipos.

La mejor manera de entenderlo quizás sea con algunos ejemplos:

#Cmdnd_Alias, especifica una serie de comandos

#Permite ejecutar cualquier comando en /usr/bin

#excepto los comandos rm y mv

Cmdnd_Alias BIN=/usr/bin, !/usr/bin/rm, !/usr/bin/mv

#Comandos de apagado

Cmdnd_Alias POWER=/sbin/shutdown, /sbin/halt, /sbin/reboot,
/sbin/restart

#User_Alias, especifica grupos de usuarios

#Especifica un grupo de usuarios

User_Alias GRUPO1=usuario1,usuario2, usuario3

#Crea una regla para los alias especificados anteriormente.

#Permite ejecutar los comandos definidos en bin pero no los #definidos en power

GRUPO1 ALL=(ALL:ALL) NOEXEC:POWER, EXEC:BIN

Los alias no suponen más que una manera sencilla de organización, lo mismo que se logra con ellos es posible hacerlo substituyéndolos por su valor en la línea que se quiere modificar del fichero sudoers. Con o sin alias, sudoers permite otorgar privilegios a determinados usuarios para un conjunto limitado de acciones, frente al comando su que les permite actuar en nombre de otro usuario sin restricción alguna.

8.7 UIDs y cambios de usuario

Sabemos que los comandos `su` y `sudo` nos permiten cambiar de usuario o actuar en nombre de otros usuarios. Si revisamos la sección 2.3 Permisos y propiedades de ficheros, recordaremos que era posible también trabajar con determinados ficheros como si fuésemos el propietario o grupo propietario de esos ficheros. Esto se consigue mediante el uso de algunos de los bits de permisos. Ahora que tenemos una mejor visión de los usuarios quizás sea el momento de entender realmente como funciona el cambio de usuarios.

En los sistemas Linux, siempre es responsabilidad del Kernel manejar los cambios de usuario, para ello hace uso de la familia de llamadas al sistema `setuid`. `setuid` es también un programa existente en los sistemas Linux que permite cambiar el ID efectivo del proceso que lo invoca. Más adelante veremos que es el ID efectivo. Antes de eso vamos a ver que reglas define el kernel de Linux sobre lo que un proceso puede hacer y lo que no. Aquí mostramos solo tres de ellas puesto que son las que más directamente nos afectan en cuanto a UIDs.

1. Un proceso ejecutándose como `root`(UID 0) puede usar `setuid()` para convertirse en cualquier otro usuario.
2. Un proceso que no se ejecuta como `root` tendrá importantes restricciones para usar `setuid`, de hecho en la mayoría de ocasiones no podrá hacerlo.
3. Cualquier proceso puede ejecutar un programa con un bit `setuid` o `setgid` siempre que tenga los permisos necesarios para ello.

Después de estas tres reglas, es necesario hacer una distinción que no siempre es sencilla. Cuando se habla de cambio de

usuarios no estamos hablando de introducir contraseñas o de nada que tenga que ver con los nombres de usuario. Tanto los nombres de usuario como las contraseñas son elementos del espacio de usuario y por lo tanto no tienen relación con lo que haga el kernel por debajo de este nivel de abstracción. De hecho, para el kernel los usuarios son simplemente números, en la sección 8.1 el fichero `/etc/passwd`, hemos podido ver como cada nombre de usuario se vinculaba con un identificador numérico único, este es el que emplea el kernel para trabajar.

Llegados a este punto es importante ser conscientes que en los sistemas Linux modernos todos los procesos tienen más de un UID. El UID que hemos estado describiendo hasta ahora, aquel que ejecuta un proceso, es el UID efectivo. Aparte de este tenemos el UID real, que indica a que usuario pertenece el proceso, es decir, que usuario ha estado interactuando con él y puede enviarle señales. Esto puede parecer confuso y la documentación que hay al respecto no ayuda a hacerlo más claro. El hecho que para la mayoría de procesos el UID efectivo sea igual al real tampoco ayuda a aclarar su funcionamiento.

Podemos intentar comprenderlo mejor con un ejemplo: supongamos que usuario1 lanza un proceso que tiene el `setuid` puesto y que pertenece al usuario2. El usuario1 sería aquí el UID real porque es el que ha lanzado el proceso y el que puede interactuar con el. Sin embargo, el programa tenía el `setuid` puesto y pertenece al usuario2, por tanto el proceso se ejecutará como si se tratase del usuario2 y por tanto el UID efectivo correspondería al usuario2. Intentando hacer una analogía: el UID real equivaldría al sargento que da la orden y el UID efectivo sería el soldado que la ejecuta.

Los comandos o programas que hemos visto hasta ahora nos

mostraban únicamente el UID efectivo de los procesos en ejecución. En la mayoría de casos, es posible mostrar, además, el UID real si se le especifica al programa invocado las opciones adecuadas. Por ejemplo, en el caso del comando `ps`, podemos recurrir a la siguiente línea para ver el UID real y el efectivo. No debe sorprendernos si en la mayoría de casos ambos valores son el mismo.

```
ps -eo pid,euser,ruser,comm
```

Es importante observar que después de iniciar un programa como `sudo`, no nos será posible enviarle señales o interactuar con el proceso sin ser `sudo`. Esto es una consecuencia del comportamiento de `sudo`. Este programa lo que hace es cambiar el UID real del programa que ejecutamos además del efectivo con lo que la interacción con el proceso queda a cargo de `root` o del usuario que empleemos con `sudo` en este caso. Algunos programas, sobretodo por motivos de seguridad, es recomendable que no se ejecuten con una cuenta como la de `root` que tiene permisos casi ilimitados. Es por ello que en el fichero `sudoers` disponemos de la opción: `stay_setuid`. La opción, le indica a el programa `sudo` que no debe cambiar el UID real al ejecutar un comando.

Dado que es el kernel el que maneja los cambios de usuario, los administradores y desarrolladores de sistemas Linux deben ser especialmente cuidadosos con los cambios de usuario y `setuid` en sus programas y en los permisos de los ficheros. Muchas de las intrusiones que se producen en sistemas Linux se aprovechan de vulnerabilidades en los programas para tratar de conseguir una elevación de privilegios, así se conoce a poder ejecutar programas o abrir ficheros como `root`, para poder tener acceso total al sistema.

8.8 Identificación y Autenticación de usuarios

Los sistemas multiusuario como son los sistemas Linux, deben proveer de sistemas de identificación y autenticación de usuarios para garantizar la seguridad del sistema. Podemos entender la identificación como la pregunta: ¿Quién es el usuario? La autenticación trata de averiguar si el usuario realmente es quien dice ser. Ambos procesos sirven tanto para entrar al sistema como para trabajar con los distintos ficheros y programas.

Si recordamos como trabajaba el kernel de los sistemas Linux con los usuarios nos vendrá a la mente que este los identificaba únicamente con números, con identificadores únicos de usuario o con UID, mientras que los diferentes agentes se identifican a ellos mismos usando un nombre. Así pues el kernel conoce las reglas de como gestionar las llamadas para `setuid` y conoce que reglas aplicar a uno u otro usuario autenticado. El kernel, por contra, no sabe nada acerca de contraseñas o nombres de usuario, estas cosas son del ámbito exclusivo del espacio de usuario.

Supongamos, entonces, un caso en el que un proceso de usuario necesita conocer el nombre de usuario del que lo ejecuta para poder autenticarlo, igual porque estamos trabajando con `sudo`. El procedimiento que seguiría el proceso sería algo similar a lo que se presenta a continuación, aunque más complejo.

1. El proceso pide al kernel su UID efectivo mediante una llamada al sistema.
2. El proceso abre el fichero `/etc/passwd`
3. El proceso lee una línea. Si no quedan líneas por leer

termina y da un error. El nombre de usuario no existe.

4. El proceso separa cada línea en campos. El tercer campo corresponde con el UID. Recordemos la sección 8.1 el fichero `/etc/passwd`.
5. Si el ID obtenido del paso 4 se corresponde con el del paso 1 termina y devuelve el campo 1. Es decir, el nombre de usuario.
6. El proceso avanza a la siguiente línea. Vuelve al paso 3.

Este proceso esconde más complejidad de la mostrada aquí, pero estos 6 puntos nos sirven para hacernos una idea aproximada de lo que sucede. Precisamente debido a la complejidad, los sistemas Linux no dejan a cada desarrollador la tarea de implementar la identificación, eso sería confuso y daría lugar a mayores problemas de seguridad de los que solucionaría. Es por ello que se hace uso de librerías que permiten llevar a cabo estas tareas. Además se dispone de un conjunto de llamadas al sistema como `geteuid()` o `getpwuid()` que permiten hacer de las tareas que se presentan un proceso más o menos homogéneo para cualquier desarrollador.

Igual que para obtener los nombres de usuario se dan estas facilidades, no sucede lo mismo con las contraseñas. En el caso de las contraseñas, según los mecanismos tradicionales, es el desarrollador el que debe encriptar debidamente el valor que le proporcione el usuario para comparar posteriormente el valor cifrado con el valor que se encuentre en el fichero de configuración.

8.9 PAM (Pluggable Authentication

Modules)

Adiferencia de la identificación la autenticación no gozaba de un sistema centralizado en los antiguos sistemas Linux. Para solventar el problema, aparece en 1995 de mano de Sun Microsystems lo que se conoce como PAM por las siglas en inglés de Pluggable Authentication Mode. Este sistema provee un mecanismo flexible y centralizado para la autenticación de los usuarios. Aunque a día de hoy, no todas las distribuciones de Linux implementan por defecto este mecanismo si que lo hacen la mayoría de ellas.

La idea de PAM es que las diferentes aplicaciones puedan delegar la autenticación de usuarios a este mecanismo para tratar de identificar y autenticar al usuario. El diseño centralizado que tiene y el sistema modular permiten añadir de manera sencilla segundos factores de autenticación, la autenticación mediante dispositivos físicos, un usb por ejemplo, o casi cualquier otro sistema que se nos ocurra.

Para entender como funciona PAM lo mejor es echar un vistazo a sus ficheros de configuración. Los mayores problemas que nos encontraremos son:

- Cada distribución gestiona de forma distinta los ficheros de configuración de PAM
- Es posible encontrar la configuración de PAM en `/etc/pam.conf` o bien en distintos ficheros dentro del directorio `/etc/pam.d/`
- Los ficheros de configuración permiten la inclusión, mediante líneas de texto en los ficheros de configuración de otros ficheros de configuración completos, cosa que dificulta la tarea de

entenderlos.

- No todos los módulos de PAM tienen porque estar siempre disponibles, es posible que sea necesario habilitarlos.

Teniendo claras las limitaciones que existen podemos abordar los ficheros de configuración de PAM. Estos ficheros, independientemente de la distribución, son ficheros de texto con una estructura similar. Empecemos por el caso en el que disponemos del fichero `/etc/pam.conf`. En este caso, cada línea del fichero representa una regla y debe tener una estructura como la que se presenta a continuación:

servicio tipo control ruta [argumentos]

- **servicio:** Nombre de la aplicación o el fichero sobre el que se aplicará la regla.
- **Tipo:** Especifica el tipo de acción que deberá hacer PAM. Esto se hace mediante la especificación de uno de los 4 grupos de gestión que se presentan a continuación:
- **account:** Permite realizar tareas de gestión de la cuenta de usuario sin realizar ninguna autenticación, por ejemplo validar el estado de una cuenta de usuario.
- **auth:** Permite autenticar a un usuario en dos pasos. El primero de ellos es asegurarse que el

usuario es quien dice ser, mediante contraseña o cualquier otro mecanismo. El segundo es comprobar la pertenencia del usuario a un grupo o los privilegios de este en el sistema.

- **password:** Permite cambiar las credenciales de un usuario, bien se trate de una contraseña o de cualquier otro tipo de credencial.
- **session:** Permite realizar una acción exclusivamente para la sesión del usuario. Mostrar el mensaje del día por ejemplo o montar un directorio.

Si el valor del tipo está precedido por un carácter '-', no quedará constancia en ningún archivo del sistema en caso que el módulo no pueda ser cargado. Si no se especifica el carácter '-' y el modulo no se puede cargar se generará la anotación correspondiente.

- **Control:** Especifica que debe hacer PAM después de que la acción falle o no. Puede tomar valores complejos o valores simples. En seguida entraremos en más detalle acerca de los valores de control.
- **Ruta:** El módulo de PAM que se ejecutará para esta línea. Lo que hace cada módulo puede encontrarse en el manual de dicho módulo. Dado que es posible programar nuestros propios

módulos para PAM es necesario revisar la documentación de cada uno. La ruta del módulo debe ser absoluta. Si no lo es, debemos tener en cuenta que será relativa a `/lib/security` en la mayoría de las distribuciones. Podríamos encontrarnos por ejemplo con: `pam_shells.so`

- **Argumentos:** Algunos módulos aceptan argumentos propios al módulo para modificar su comportamiento. En caso que existan deben ir en este lugar. Los argumentos de cada módulo deberán consultarse en la documentación de este, por ejemplo:

```
auth sufficient pam_unix.so nullok
```

Especifica que el usuario podría no tener contraseña sin que ello devuelva un error.

Antes de entrar de lleno con los valores de control, debemos hacer una última consideración. Las reglas que definamos en los ficheros de configuración de PAM se interpretan secuencialmente. Esto significa que la regla que se encuentra más arriba es la que primero se evalúa. No solo eso, si no que además las reglas pueden acumularse. Se puede definir para un mismo programa más de una regla de manera que será la combinación de todas ellas la que defina si PAM puede o no realizar la acción. En seguida veremos un ejemplo para tenerlo más claro. Veamos antes algunos de los valores que puede tomar el campo control y que significa cada uno de ellos.

- **required:** En caso de fallo PAM devolverá un fallo, pero solo después de haber evaluado el resto de

módulos que hacen referencia al programa o fichero.

- **requisite:** En caso de fallo PAM devolverá un fallo. Sin embargo, el control se devuelve inmediatamente a la aplicación sin ejecutar ninguno de los módulos restantes.
- **sufficient:** En caso de éxito PAM devolverá un éxito si ninguna línea anterior ha fallado. Ningún módulo más será ejecutado puesto que el control se devuelve a la aplicación inmediatamente.
- **optional:** El éxito o fracaso de este módulo solo se tiene en cuenta si es el único asociado al servicio.
- **include:** Incluye todas las líneas del fichero especificado como valores de control.

Para que dos reglas sean acumulables deben tener el mismo servicio y el mismo tipo. En ese caso se acumularán las reglas tal como se muestra en el ejemplo a continuación.

servicio	auth	sufficient	pam_rootok.so
servicio	auth	sufficient	pam_unix.so
servicio	auth	required	pam_deny.so

En el ejemplo mostrado PAM seguirá el procedimiento siguiente dado que todas las reglas son del mismo tipo para el mismo servicio:

1. El módulo pam_rootok.so comprueba que es el usuario root el que trata de autenticarse. Si devuelve un éxito,

dado que el valor de control es sufficient, PAM devuelve un éxito y se devuelve el control a servicio. En caso de fallo pasará a la siguiente regla acumulada, mismo servicio y mismo tipo.

2. El módulo pam_unix.so pide la contraseña al usuario, en caso de que la contraseña que ofrece el usuario sea correcta se devuelve un éxito. Dado que el valor del campo control está puesto a sufficient, PAM devolverá un éxito y se devolverá el control a la aplicación. Si da un error se procederá con la siguiente línea.
3. El módulo pam_deny.so siempre devuelve un fallo. Dado que el campo control toma el valor required PAM devolverá un error. Antes de devolver el error, sin embargo, ejecutará el resto de líneas. Si en lugar de required tuviésemos requisite aquí terminaría el trabajo de PAM. En este caso no hay más líneas, por tanto aquí terminaríamos sucediese lo que sucediese.

A partir de la configuración que hemos visto podemos hacernos una idea de como funciona PAM. El funcionamiento concreto queda “ofuscado” por los diferentes módulos que se usan. Además, esta configuración es únicamente para el caso de disponer del fichero /etc/pam.conf. ¿Que sucede si en nuestro sistemas disponemos en su lugar del directorio /etc/pam.d/?

El funcionamiento de PAM en esta situación es muy similar al que ya se ha mostrado. La principal diferencia estriba en que en el directorio /etc/pam.d encontraremos un fichero para cada servicio al

que queremos definirle alguna regla. Así pues, en lugar de tener un fichero monstruoso con líneas de la forma:

`servicio tipo control ruta [argumentos]`

Tendremos el servicio definido por el nombre del fichero dentro del directorio `/etc/pam.d/` mientras que dentro de cada fichero tendremos, en texto, una línea por cada una de las reglas para ese servicio siguiendo la forma:

`tipo control ruta [argumentos]`

Aunque los módulos de PAM que tengamos y la propia configuración dependen en gran medida de la distribución de Linux que usemos y del software que tengamos instalado, tenemos las referencias anteriores sobre su configuración para entender el funcionamiento y, por que no, para atrevernos a configurar los módulos que más nos convenga. Sea como sea, hay algunos consejos que serán aplicables, si no para todas, para la gran mayoría de distribuciones.

- Podemos saber que módulos de PAM hay cargados en nuestro sistemas con `man -k pam_` (importante no dejarse la barra baja).
- Una vez sabemos que módulos tenemos cargados podemos tratar de localizarlos mediante el comando `locate`.

`locate pam_unix.so`

- Los argumentos de los módulos y su comportamiento siempre está definido en los

manuales.

- En el caso que exista, el fichero `/etc/pam.d/other` define la configuración por defecto para cualquier aplicación que no disponga del fichero de configuración para PAM correspondiente.
- Algunas distribuciones de Linux generan ficheros de configuración para PAM de forma automática, en estos casos, no suele ser muy recomendable cambiar dichos ficheros de configuración a menos que se tenga muy claro lo que se está haciendo. En estos casos, siempre encontraremos algunas comentarios en el fichero, líneas que empiezan por `#` para avisarnos de que los ficheros han sido generados automáticamente.

Al trabajar con los ficheros de configuración de PAM debemos hacer siempre una copia de seguridad. Si por algún casual cometiésemos un error en los ficheros de configuración podríamos llevarnos una sorpresa muy desagradable. En el peor de los casos podríamos ser incapaces incluso de entrar al sistema. Si esto sucede y hemos sido precavidos, lo único que debemos hacer es restaurar los ficheros de configuración desde la copia de seguridad.

8.9.1 Limitando recursos con PAM

Como hemos visto en PAM (Pluggable Authentication Modules), PAM sirve para un gran abanico de cosas. Además de todo lo visto, PAM nos ofrece una funcionalidad muy interesante. Nos permite limitar los recursos para los distintos usuarios. Esta es una

funcionalidad que nos resultará interesante tanto a nivel de administración como a nivel de seguridad. Por desgracia, no vamos a encontrar esta funcionalidad en todas las distribuciones ni en todas las versiones de PAM. Sea como sea, la mayoría de sistemas Linux y BSD, estos últimos tienen los ficheros de configuración en otras rutas, aceptan este tipo de limitaciones.

Igual que sucede con las otras funcionalidades de PAM, podemos encontrar alguna discrepancia sobre la limitación de recursos según la distribución de Linux que usemos. Esto no supone un obstáculo para que veamos como funciona. En caso de encontrar una discrepancia siempre podemos recurrir a los manuales que el mismo sistema provee. Normalmente encontraremos el fichero de configuración con el que vamos a trabajar en: `/etc/security/limits.conf`

El fichero de configuración es un fichero de texto con una línea para cada una de las reglas. Cada línea se haya representada en el siguiente formato:

usuario	tipo	recurso	limite
---------	------	---------	--------

- **usuario:** Define a que usuario se le aplica la regla. Debe existir en el sistema. Las reglas pueden definirse para usuarios. Para grupos si empiezan por '@' o como reglas por defecto si se llena el campo usuario con el carácter '*'.
- **tipo:** Podemos especificar las reglas como hard o soft, duras o blandas. Las reglas definidas como hard solo podrán ser cambiadas por el usuario root. Las reglas soft pueden llegar a ser cambiadas por el propio usuario.

- **recurso:** El recurso que se quiere limitar. Algunos de los valores que puede tomar son los siguientes:
 - **core:** El tamaño máximo de los ficheros de volcado en kilobytes.
 - **fsize:** El tamaño máximo de los ficheros en kilobytes.
 - **nofile:** El número máximo de ficheros que el usuario puede tener abiertos al mismo tiempo.
 - **rss:** Cantidad máxima de memoria que pueden usar los procesos de la sesión en kilobytes
 - **cpu:** Especifica la cantidad de minutos que la sesión puede hacer uso de la CPU.
 - **nproc:** Especifica la cantidad máxima de procesos que pueden ejecutarse en una sesión.
- **límite:** Dependiendo del valor que tome el recurso tomará un valor u otro para indicar el límite correspondiente.

Las reglas en el fichero `/etc/security/limits.conf` son leídas de arriba abajo. Así pues si hay dos reglas que se contradicen la que aparece más abajo en el fichero es la que se aplicará. Hay que tener en cuenta además que una regla definida para todos los usuarios, con un asterisco en el campo usuario, siempre tendrá menos peso que una regla específica. Lo mismo sucede para los grupos. Es decir, una regla definida de forma estricta para un único usuario siempre prevalecerá sobre las reglas definidas para el grupo o por defecto. De la misma

manera una regla definida para un grupo prevalecerá sobre las reglas que se definen por defecto.

9. Arranque del sistema

Ahora que conocemos los sistemas de ficheros, los procesos y los usuarios llega el momento de saber que sucede al arrancar un sistema Linux: como arranca el Kernel y que sucede hasta que se ejecutan los procesos de usuario.

Comprender como arranca un sistema Linux nos ayudará a comprender el sistema en todo su conjunto en lugar de hacerlo por sus distintas partes o de una manera modular. Además puede ayudarnos a solucionar diversos problemas que pueden aparecer mientras el sistema se inicia. No debe preocuparnos reconocer las etapas en las que nuestro sistema arranca dado que no es una tarea sencilla, y menos en los sistemas modernos. Es muy probable que muchas ocasiones seamos incapaces de interactuar con el sistema de arranque hasta que se ejecute el primer proceso de usuario. Sin embargo, seremos capaces de ver a posteriori los mensajes que se han generado.

De forma simplista podríamos decir que todos los sistemas arrancan de una forma similar a la siguiente:

1. La BIOS o el software de arranque cargan y ejecutan el cargador de arranque, en inglés boot loader.
Recordemos que, como comentábamos en la sección 4.1 Tablas de particiones y particiones, es posible que en los discos nos encontremos con un MBR (master boot record) que contendrá el software de arranque o puede ser que este proceso dependa completamente de la BIOS y los sistemas UEFI, como sucede en los

sistemas modernos.

2. El cargador de arranque, boot loader, busca la imagen del kernel en el disco y la carga en memoria, una vez cargada en memoria la ejecuta.
3. El kernel inicializa los dispositivos y sus drivers correspondientes. Podemos refrescar algunos detalles si volvemos al capítulo 3. Dispositivos o a la sección 3.5 Udev. Es en este momento que kernel ya conoce los dispositivos conectados y es capaz de proporcionar esa información para que nosotros trabajemos más adelante, aún así aun no podremos interactuar con esta información.
4. El kernel monta el sistema de ficheros raíz. Es decir, el sistema de ficheros principal. Si recordamos la sección 4.9 Montando un sistema de ficheros y, en concreto como funciona el fichero `/etc/fstab`, podemos darnos cuenta que este proceso es algo complejo y que implica comprobaciones y otras sub tareas.
5. El kernel inicia un programa llamado `init` con PID 1. Es el primer programa del espacio de usuario. Es decir, el programa `init` inicia el espacio de usuario.
6. `Init` inicia los servicios de bajo nivel como `udev` y `syslogd`. Esto significa que hasta este momento no somos capaces de interactuar con los dispositivos o de generar entradas de log en el sistema.
7. Se configura y se levanta la red.

8. Se ejecutan los servicios de nivel intermedio y de alto nivel como cron. Podemos recordar que era cron en la sección 6.4 Planificando tareas.
9. Se lanzan los procesos de alto nivel como la interfaz gráfica o GUI, si la hubiese, o los procesos de entrada al sistema para los usuarios.

Como podemos ver, el arranque del sistema es un proceso complejo que involucra muchos pasos. Intentaremos obtener una visión completa de estos pasos para ser capaces de entender el sistema en su conjunto. Aeste proceso hay que sumar otro proceso similar, aunque más sencillo que es realizado antes de la carga del sistema.

Al encender una máquina, antes de empezar todo este procedimiento, se produce una comprobación del hardware de la máquina. Si faltase algún componente importante, si la fuente de alimentación no suministrase suficiente potencia, o si un elemento estuviese conectado allí donde no debe, es posible que ni siquiera lleguemos a ver arrancar la BIOS. En este caso, si hemos montado nosotros mismos la máquina o si nos la han montado por piezas, antes de entrar en pánico deberíamos revisar que hemos montado correctamente cada uno de los componentes.

9.1 Los mensajes de inicio

Todos los sistemas Unix, esto incluye también a los sistemas Linux, producen una gran cantidad de mensajes de diagnostico a medida que arrancan. Desgraciadamente para nosotros la cantidad de estos mensajes es enorme y en la mayoría de los casos muy difícil de comprender. Al principio, es decir en las primeras etapas del sistema,

estos mensajes son generados por el kernel. En las últimas etapas de arranque algunos servicios o procesos pueden generar también este tipo de mensajes de diagnostico, haciendo que sea más difícil de comprender.

Incluso una vez el sistema ya ha arrancado el kernel sigue generando mensajes de diagnostico ante algunos eventos. Por ejemplo, puede generar mensajes de diagnostico al conectar un nuevo dispositivo. Eso hace que la cantidad de mensajes final que podemos ver sea muchas veces desalentadora.

Gracias a la evolución del hardware y del propio software es posible que el usuario ni siquiera sea capaz de ver estos mensajes. Hay distribuciones de Linux o sistemas que se han configurado para mostrar una pantalla de carga que oculta todos estos mensajes. A veces, ni tan siquiera es una pantalla de carga y se trata únicamente de una pantalla en negro. En estos casos, apretar la tecla ESC (o tecla escape) suele ser suficiente para desactivar la pantalla de carga y ver los mensajes. Aún así, la evolución del hardware ha llevado a que estos mensajes aparezcan muy rápidamente en la pantalla, siendo casi imposible leerlos. Es por ello que tenemos dos maneras prácticas de ver estos mensajes.

- **Mirar en los logs del sistema.** Estos suelen encontrarse en el directorio `/var/log`. En el caso de los mensajes del kernel los encontraremos, dependiendo de la distribución que usemos, en `kern.log`, `messages` o `messages.log`. Sea cual sea nuestra distribución y nuestro fichero, un vistazo a los ficheros en `/var/log` nos dará una pista bastante buena puesto que los nombres son bastante

descriptivos.

- **El comando dmesg.** Este comando nos devolverá el contenido de los mensajes del kernel desde el último arranque. La salida de dmesg puede ser extremadamente grande por lo que no se recomienda ejecutar el comando dmesg sin más. En su lugar es recomendable redirigir la salida a otro programa como less, more o tail.

```
dmesg | tail
```

```
dmesg | less
```

Aunque no podemos entrar a explicar todos los mensajes que arroja dmesg a continuación se muestra una posible salida de este comando:

```
[ 0.000000] Initializing cgroup subsys cpuset
[ 0.000000] Initializing cgroup subsys cpu
[ 0.000000] Initializing cgroup subsys cpuacct
[ 0.000000] Linux version 3.11.0-13-generic (buildd@aatxe) (gcc
version 4.8.1 (Ubuntu/Linaro 4.8.1-10ubuntu8) ) #20-Ubuntu SMP Wed
Oct 23 17:26:33 UTC 2013
(Ubuntu 3.11.0-13.20-generic 3.11.6)
[ 0.000000] KERNEL supported cpus:
[ 0.000000] Intel GenuineIntel
[ 0.000000] AMD AuthenticAMD
[ 0.000000] NSC Geode by NSC
[ 0.000000] Cyrix CyrixInstead
[ 0.000000] Centaur CentaurHauls
```

```
[ 0.000000] Transmeta GenuineTMx86
[ 0.000000] Transmeta TransmetaCPU
[ 0.000000] UMC UMC UMC UMC
[ 0.000000] e820: BIOS-provided physical RAM map:
[ 0.000000] BIOS-e820: [mem0x0000000000000000-
0x0000000000009fbff] usable
[ 0.000000] BIOS-e820: [mem0x000000000000f0000-
0x000000000000fffff] reserved
[ 0.000000] BIOS-e820: [mem0x0000000000100000-
0x000000007dc08bff] usable
```

Si nos fijamos, aunque los mensajes son muchos y muy variados, todos siguen una estructura, más o menos, común. Al modo de guía podemos observar que todos los mensajes empiezan con un número decimal entre corchetes '[0.000000]'. Este número indica el tiempo transcurrido desde el arranque del sistema. Es decir, si vemos todo ceros, se trata de un evento que ha sucedido en el segundo 0 de arranque del kernel. Si por el contrario viésemos un mensaje que empezase por [3.432100] estaríamos ante un mensaje que se ha generado en el segundo 3 desde el arranque del sistema. Esta marca temporal puede sernos muy útil para tratar de detectar errores. Se puede observar también que en ocasiones los mensajes empiezan por unas letras o número seguidos del carácter ':' y a continuación continua el mensaje. En estos casos, lo que encontramos antes del carácter ':' indica quien está produciendo el mensaje. Lo que encontramos después de los dos puntos no es otra cosa que el mensaje generado.

Otro comando muy útil para trabajar con la salida de dmsg o

con los logs del kernel es el comando grep. Como ya hemos visto en la sección 5.5 Algunos comando básicos, el comando grep es una herramienta que permite buscar texto plano que coincida con una expresión dada. Así pues podemos hacer uso de expresiones como:

#Buscar mensajes sobre los dispositivos SCSI disk

```
dmesg | grep -i "sd"
```

#Buscar mensajes sobre dispositivos usb

```
dmesg | grep -i "usb"
```

Al igual que muchos otros de los comando de sistemas Linux, dmesg tiene sus propias opciones y argumentos. La lista de argumentos para dmesg es considerable y nos permite una gran flexibilidad. Vamos a ver algunos parámetros que resultan interesantes. Para más información sobre los parámetros de dmesg es posible recurrir a la entrada del manual:

```
man dmesg
```

- **-u:** Muestra los errores del espacio de usuario.
- **-t:** No muestra las marcas temporales del kernel.
- **-C o --clear:** Elimina el buffer de mensajes del kernel. Debemos ser cuidadosos con esta opción.
- **-c:** Elimina el buffer con los mensajes del kernel después de mostrarlos por pantalla
- **-L:** Muestra la salida de dmesg en distintos colores. Puede ayudar a comprender los mensajes que vemos.

- **-l o -level <lista>**: Limita la salida de dmesg a los niveles de error especificados. Recibe como parámetro la lista de niveles separados por comas. Podemos ver todos los niveles especificando a dmesg la opción -h. Algunos de ellos son:
 - **err**: Error
 - **warn**: Warning o advertencia. Son errores que no impiden seguir con la ejecución del programa que los genera.
 - **crit**: Se trata de errores críticos

El tamaño del buffer que almacena los mensajes del kernel es limitado. Normalmente es suficientemente grande como para poder guardar un gran número de mensajes, sin embargo hay que tener en cuenta que si el sistema lleva mucho tiempo en funcionamiento y se han ido generando anotaciones por parte del kernel, iremos perdiendo las más antiguas. Normalmente esto no suele ser un problema, puesto que aquellas anotaciones que se borran de los ficheros de logs del sistema son suficientemente antiguas para que no debemos preocuparnos por ellas.

Ampliación

Los logs o anotaciones del sistema se almacenan en ficheros. Para evitar que estos ficheros crezcan indefinidamente, existe un mecanismo conocido como log rotate, rotación de anotaciones. Cada vez que un fichero alcanza un tamaño determinado, el fichero se comprime y se inicia un nuevo fichero de logs. El número de

ficheros comprimidos que se almacenan es configurable por el administrador de la máquina. Una vez se sobrepasa el número máximo de ficheros comprimidos guardados se empiezan a borrar los más antiguos antes de guardar nuevos ficheros.

9.2 El bootloader

El boot loader es un pequeño programa que arranca después de las comprobaciones iniciales y de la interfaz de entrada/salida básica, es decir después de la BIOS. El arranque del sistema depende en gran medida de nuestros discos y de la BIOS, el boot loader, por tanto, deberá ser acorde a la configuración de nuestra máquina. En función de si disponemos de una máquina que permite el arranque desde UEFI o si tenemos un disco con un MBR, ver la sección 4.1 Tabla de particiones y particiones, deberemos escoger el boot loader en consecuencia.

La tarea del boot loader es sencilla, al menos en apariencia. Carga el kernel del sistema operativo en memoria y lo ejecuta. Aunque enunciado parece sencillo es una tarea que a medida que la miramos detalladamente se revela complicada. En primer lugar debemos saber donde se encuentra la imagen del kernel. En segundo lugar debemos conocer con que parámetros se debe llamar al kernel.

Parece que la respuesta es sencilla: tanto el kernel del sistema operativo como los parámetros con los que se debe llamar están en algún lugar del sistema de ficheros. La conclusión sencilla es que el boot loader únicamente debe buscar en el sistema de ficheros para encontrar la imagen del kernel y el fichero de configuración con los parámetros. Aunque se trata de la conclusión más fácil, se nos

presenta un gran problema. Debemos recordar que es el kernel el que monta el sistema de ficheros. No solo eso, también es el kernel el que carga los drivers para entenderse con los dispositivos de almacenamiento.

Aunque parecen problemas insalvables los boot loaders recurren a la BIOS o a UEFI para cargar unos drivers muy sencillos que les permiten acceder a cualquier tipo de dispositivo de almacenamiento. El acceso no es rápido ni el más eficiente, pero se emplea únicamente hasta que el momento en que se carga el kernel. El método que suelen usar tanto BIOS como UEFI para el acceso a los discos es el direccionamiento por bloque lógico o LBA. El acceso es secuencial según la numeración de los bloques, pero como se ha comentado, cuando el kernel este cargado ya se ocupará de mejorar la velocidad. De la misma manera, ambos sistemas, al menos actualmente, pues no ha sido siempre así, soportan el montaje de sistemas de ficheros en modo de solo lectura, permitiendo de esta manera localizar los ficheros.

Debemos tener en cuenta que cuando no era posible montar los sistemas de ficheros en modo lectura programar los boot loaders era más complejo que en la actualidad puesto que debían lidiar con el problema. Así pues, una vez solucionada la principal problemática podemos decir que además de lo descrito un boot loader debe ser capaz de:

- Permitir seleccionar entre distintos kernels.
- Permitir escoger entre distintos parámetros para iniciar el kernel. Aunque esto dependerá de como configuremos el cargador de arranque o

boot loader. Hay sistemas operativos como Windows que en un funcionamiento normal no dan esta opción. Por suerte la mayoría de bootloaders que cargan sistemas si que la permiten Linux.

- Permiten cargar distintos sistemas operativos. Hay que remarcar que esto es distinto de la primera opción. Como veíamos en la sección 1.2 Una visión general de los sistemas Linux, un sistema Linux no se compone únicamente por el kernel. Esto significa que podemos tener un mismo sistema operativo con dos kernels distintos.

Aunque algunas distribuciones incluyen uno u otro, escoger el boot loader es una tarea que depende de cada usuario. Cada uno tiene sus propias características, algunos están pensados únicamente para sistemas UEFI, otros para sistemas con uso de MBR y otros pueden trabajar con ambos. Estos son algunos de los más conocidos:

- **GRUB:** Puede trabajar con escenarios basados en UEFI o BIOS+MBR. Es uno de los más extendidos y más usados independientemente de la distribución de Linux que utilicemos y en el que nos centraremos. Su configuración es bastante intuitiva puesto que sigue el formato de muchos scripts de sh, el terminal del que evoluciona bash.
- **Syslinux:** Está pensado para cargar el kernel

desde una gran variedad de sistemas de ficheros, sin embargo permite cargar únicamente ficheros que se encuentren en la partición donde se encuentra instalado.

- **LILLO y ELILO:** El primero para trabajar únicamente con arranque desde la BIOS, el segundo para arrancar únicamente desde UEFI. Es uno de los primeros boot loaders de Linux. El soporte para ELILO ha sido discontinuado por lo que no se recomienda su uso. En cualquier caso es decisión del usuario. El soporte para LILLO también se ha visto discontinuado debido a la amplia aceptación de GRUB.
- **systemd-boot:** Es un boot loader que trabaja únicamente con UEFI permitiendo seleccionar entre varias imágenes que cargar según su configuración o según un menú en pantalla. Se incluye en los sistemas junto con systemd. Más adelante hablaremos de systemd. Pese a disponer de este cargador en el sistema sigue siendo posible recurrir a otros como GRUB.

Aunque la variedad de boot loaders es, a día de hoy, bastante considerable, GRUB es casi universal por la sencillez de configuración y su rendimiento. Normalmente cuando se recurre a otro cargador de arranque es porque se busca una mayor velocidad, menor tiempo de respuesta o una configuración más sencilla aunque se pierda cierta flexibilidad. Sea como sea, la elección queda en manos del usuario.

9.3 Parámetros del kernel

Muchos de los programas en los sistemas Linux disponen de parámetros que modifican su comportamiento. Lo hemos estado observando a lo largo de todos los comandos mostrados. El kernel o núcleo del sistema no es distinto. No solo eso si no que además, el kernel suele ser bastante parecido, si no igual, entre todas las distribuciones. La discusión de si es igual o no o de donde termina el kernel y empiezan los módulos que ya no forman parte de este la dejaremos para otros.

Una vez hemos arrancado el sistema y hemos entrado podemos ver que parámetros ha pasado el boot loader al kernel. Para ello recurrimos a un comando y a un directorio que ya conocemos:

```
cat /proc/cmdline
```

Los parámetros que nos muestra pueden parecer confusos pero solo podemos encontrar parámetros de dos tipos. Palabras sueltas que especifican un comportamiento concreto o parejas de clave=valor. En estos casos el kernel especifica que una misma clave puede tener n valores separados por comas. Es decir: clave=valor1, valor2, ..., valorN. En ocasiones esto no será útil, en otras puede llegar a serlo mucho. De la misma manera hay parámetros que resultan críticos para que todo funcione mientras que hay otros que son superfluos.

La lista de parámetros del kernel es bastante larga, podemos encontrar por Internet algunas listas de estos parámetros. En la gran mayoría de ocasiones encontrar la lista completa no nos aportará nada

que necesitamos. En ocasiones, encontrarla puede ayudarnos a entender mejor el sistema en su conjunto y el núcleo de los sistemas Linux en particular.

A la hora de trabajar con los parámetros el kernel buscará primero las claves siguientes:

- root=<valor>
- nfsroot=<valor>
- nfsaddrs=<valor>
- ro
- rw
- debug
- init=<valor>

A continuación el kernel revisará el resto de parámetros. Si el parámetro es conocido se interpreta según su significado. Si el kernel encuentra una pareja de clave=valor donde la clave no es ninguno de los parámetros que el kernel puede aceptar, entonces se considera como una variable de entorno. Es decir si encontrase *internals='mi_valor'* consideraría que internals es una variable de entorno. Así pues, si alguno de los parámetros no es soportado por el kernel esto no causará un error crítico.

Algunos de los parámetros que se le pueden pasar al kernel son los siguientes:

- **root=** : Este parámetro le dice al kernel que dispositivo usar como sistema de ficheros raíz en el momento de arrancar. Recibe un parámetro

del tipo `/dev/sda3`. Debemos tener en cuenta que esta forma no es equivalente a la que se usa más adelante para representar los dispositivos en el kernel. De hecho para un dispositivo que tiene como valor de mayor 8 y valor menor 3. El kernel podría recibir un parámetro del estilo `0x803` en lugar de `/dev/sda3`.

- **nfsroot=** : Asigna el valor de la raíz del nfs (Network FileSystem o en castellano Sistema de ficheros en red), cuando es necesario. Si la ruta especificada no empieza por '/', ',' ó un número se añade a los especificado el prefijo `'/tftpboot'`.
- **nfsaddr=** : Permite especificar la dirección de la máquina remota en caso de un arranque en red. Es importante recalcar que con la bajada de precio de las unidades de almacenamiento, casi siempre discos duros, cada vez es menos usual cargar el sistema operativo en red, aunque con el uso de chromebooks quizás este repuntando un poco. Sea como sea, si se especifica la opción `'root='` esta tendrá prioridad sobre el arranque de red.
- **init=** : Define el primer comando que debe ejecutar el kernel. Si no se especifica ningún valor el kernel probará `/sbin/init`, `/bin/init` y finalmente `/bin/sh`. Si no encuentra ninguno de estos programas y no se ha especificado valor

nos dará un error de pánico. Para los habituados a Windows: el equivalente a un pantallazo azul. Este comando es muy importante. Quizás resulta interesante, ahora o después de terminar de leer lo referente al arranque del sistema, reflexionar sobre que pasaría si el primer comando que ejecuta el kernel es `/bin/bash`.

- **ro, rw:** Le dicen al kernel que debe montar el sistema de ficheros raíz en modo de solo lectura o en modo de lectura y escritura respectivamente. Suele ser habitual encontrar la opción de solo lectura para el sistema de ficheros raíz.
- **debug:** Los mensajes del kernel que se muestran por pantalla son delegados también a un demonio, veamos en el capítulo 6. Procesos y threads, para recordar que es un demonio. Este demonio se encarga de almacenar los mensajes generados por el kernel en un fichero. Como hemos visto anteriormente los mensajes del kernel ya se guardan en un fichero, debug lo único que permite es guardar más información si cabe, demasiada en la mayoría de los casos.
- **rootdelay= :** Especifica el número de segundos que el kernel debe esperar antes de tratar de montar el sistema de ficheros raíz. Puede resultar

bastante útil en momentos en los que montamos un sistema de ficheros en red o cuando sabemos de la lentitud del dispositivo de almacenamiento.

- **rootfstype=** : Especifica al kernel que monte el sistema de ficheros raíz como si se tratase de un sistema de ficheros del tipo indicado. Normalmente se usa para garantizar compatibilidades hacia atrás, por ejemplo montar un sistema de ficheros ext3 como si se tratase de un ext2. De otra manera el propio kernel suele ser bastante bueno para detectar por él mismo el sistema de ficheros adecuado.
- **resume=** : Bastante importante, le indica al kernel donde localizar la información que se guarda después de suspender el sistema. Suele ser la misma partición donde se encuentra la swap, podemos volver a la sección 4.3 La partición SWAP o espacio SWAP, para refrescar la memoria. Suele ser útil para recuperarse de la hibernación.
- **panic=** : Por defecto después de un mensaje de pánico, un error irrecuperable del kernel, algo muy malo, el sistema no se reinicia. panic permite obligar al reinicio del sistema después de esperar los segundos especificados tras un error de pánico.

Los parámetros del kernel están divididos en categorías: según sean generales, afecten a discos, a memoria, etc. Los que se han visto aquí son todos de carácter general. Según lo que se quiera conseguir será necesario buscar parámetros más específicos.

9.4 Arranques MBR y UEFI

En todo momento se ha hablado de dos maneras distintas de gestionar el arranque del sistema: mediante master boot record (MBR) y mediante UEFI. El primero de ellos emplea la BIOS, el sistema básico de entrada/salida, para ser capaz de cargar el gestor de arranque del sistema operativo. Si recordamos la sección 4.1 Tablas de particiones y particiones, el gestor de arranque se almacenaba en un espacio de poco más de 400 bytes en el primer sector del disco duro. Esto, a día de hoy es insuficiente para el cargador de arranque. Es por ello que realmente lo que se almacena en este espacio es un cargador del cargador de arranque. Actualmente, si se debe trabajar con la BIOS para el arranque se trabaja con cargadores en múltiples etapas de manera que se almacena una primera etapa en el espacio MBR mientras que el resto del cargador de arranque o bootloader se almacena en el espacio que queda entre MBR y la primera partición. El almacenamiento del cargador de arranque antes de la primera partición no es muy seguro, puesto que puede ser sobrescrito con otra información, conscientemente o por error, impidiendo el correcto arranque del sistema. Hay tablas de particiones, como GPT, que no son soportadas por este método.

Por otro lado, el sistema UEFI está basado en un nuevo mecanismo que reemplaza la BIOS. Los sistemas UEFI están cada vez más extendidos porque ofrecen funcionalidades muy útiles como son la

posibilidad de navegar por sistemas de ficheros, o de disponer de un terminal propio antes del arranque del sistema operativo. Además ofrecen una interfaz mucho más sencilla de cara al usuario para interactuar con UEFI. Aunque muchas placas actuales ya trabajan con UEFI es habitual que se siga conociendo a la interfaz que ofrecen como BIOS.

Para almacenar el cargador de arranque, UEFI no emplea, como hemos visto, un registro de arranque primario o MBR. En su lugar, y dadas sus capacidades para trabajar con sistemas de ficheros, UEFI almacena el cargador de arranque en un sistema de ficheros conocido como ESP (EFI System Partition). Este sistema de ficheros contiene un directorio llamado `efi` que contiene a su vez un subdirectorio por cada cargador de arranque disponible. Por ejemplo: `efi/microsoft`, `efi/grub` o `efi/systemd-boot`.

En la actualidad, muchos sistemas UEFI, disponen de una opción conocida como arranque seguro (*secure boot*), que requiere que el cargador de arranque se encuentre firmado digitalmente, si no el sistema es incapaz de arrancar. Muchos de los cargadores de arranque de sistemas Linux no se encuentran firmados, mientras que las versiones de Windows más recientes como Windows 8, no arrancan si no es haciendo uso de esta característica. Para solucionar el problema, si tan solo vamos a usar Linux podemos deshabilitar el arranque seguro. Si necesitamos poder trabajar con ambos sistemas operativos al mismo tiempo deberemos buscar un cargador de arranque que se encuentre firmado digitalmente. Hay algunas versiones de GRUB que lo están.

Tanto al trabajar con BIOS como con UEFI, se debe proveer un cargador de arranque adecuado para el sistema con el que se trabaja.

Los cargadores de arranque de BIOS no funcionan para sistemas UEFI y viceversa. Al hablar de cargadores como GRUB que soportan ambos tipos, en realidad se dispone de un tipo distinto para cada uno de ellos.


9.5 GRUB, un primer vistazo

GRUB proviene de las siglas en inglés Grand Unified Boot Loader. Una de las funcionalidades más importantes que nos ofrece GRUB es la posibilidad de navegar por los sistemas de ficheros, permitiendo de esta manera un acceso sencillo a las imágenes de los distintos kernel y a las configuraciones disponibles. Podremos configurar GRUB desde nuestro sistema Linux como si de un elemento más del sistema se tratase, es decir, a través de un fichero de configuración.

Habitualmente los sistemas Linux y en general todos los sistemas operativos, hacen lo posible por esconder el cargador de arranque, sea GRUB o cualquier otro. Es posible, por tanto que nunca hayamos visto el cargador de arranque pese a que sabemos que siempre es el encargado de arrancar el kernel. Lo primero que deberíamos probar es de ver el cargador de arranque. Para ello deberemos presionar, en la mayoría de ocasiones, la tecla *shift*, para poder ver la pantalla de selección del kernel.

Una vez aparezca la pantalla de selección del kernel, es habitual que si estamos unos segundos sin interactuar de ninguna manera con ella, se lance el kernel por defecto. Si queremos disponer de tiempo para observar las opciones con tranquilidad será suficiente con apretar la tecla ESC (escape) para cancelar la cuenta atrás.

Una vez hayamos accedido al gestor que nos ofrece GRUB deberíamos ver algo parecido a lo siguiente:



```
Ubuntu 8.04, kernel 2.6.24-16-generic
Ubuntu 8.04, kernel 2.6.24-16-generic (recovery mode)
Ubuntu 8.04, memtest86+
```

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the
commands before booting, or 'c' for a command-line.

Si presionamos la tecla 'e' en cualquiera de las opciones veremos una serie de comandos y los parámetros con los que inicia el kernel, son los mismos que en la sección 9.3 Parámetros del kernel. De un primer vistazo resulta bastante confuso entender que está pasando con las opciones a la hora de arrancar el Kernel. Es fácil que veamos más de una vez la palabra root, por ejemplo. Debemos tener en cuenta que GRUB tiene sus propios módulos y comandos, la mayoría de ellos con la misma sintaxis que emplean los comandos UNIX. Además los comandos del propio kernel también cuentan con una sintaxis muy parecida.

El único root que equivale a la raíz del sistema es el que pertenece a la línea que lanza el kernel. De la misma manera los parámetros que recibe el kernel estarán siempre en la línea que empieza por "linux". En cualquier caso, y atendiendo a posibles cambios, debemos ser cuidadosos a la hora de distinguir los comandos de GRUB de la línea que lanza el kernel.

A grandes rasgos GRUB funciona de la siguiente manera:

1. Se carga el kernel de GRUB. Cuidado, el kernel de GRUB no es el kernel de Linux, se trata de cosas distintas.
2. GRUB puede acceder a las particiones y sistemas de ficheros después de la inicialización de su kernel. De nuevo no se trata todavía del kernel de Linux.
3. GRUB identifica sus particiones de arranque y carga los ficheros de configuración que corresponda.
4. GRUB ofrece al usuario una pantalla como la que hemos visto antes que permite al usuario editar los

parámetros, abrir un terminal o seleccionar que desea lanzar.

5. Después de un tiempo de espera o de la selección del usuario, GRUB lanza los comandos obtenidos. En este caso sí que se trata del kernel del sistema operativo.
6. Finalmente GRUB carga y ejecuta el kernel del sistema operativo según la localización y parámetros que ha obtenido.

En el funcionamiento de GRUB intervienen más variables y módulos, sin embargo estos seis pasos nos permiten hacernos una idea bastante buena de como trabaja GRUB.

9.6 El terminal GRUB

Una vez estamos en la pantalla de selección de GRUB se nos da la opción de abrir un terminal presionando la tecla 'c'. Este terminal es muy parecido al que manejamos en los sistemas Linux. No obstante, debemos tener en cuenta que en el punto en el que se nos ofrece el terminal el kernel de Linux todavía no ha cargado. Los comandos que tendremos disponibles son los de GRUB no los del sistema Linux, aunque realmente la sintaxis es muy parecida o igual.

Una de las cosas que podemos hacer es listar los dispositivos que tenemos conectados al sistema. Más concretamente, dispositivos de almacenamiento como discos duros que es con lo que trabaja GRUB. Recurriremos al comando ls. A diferencia del comando ls en Linux, en GRUB y sin parámetros, ls listará dispositivos en lugar de ficheros o directorios.

```
grub>ls
```

(hd0)

(hd0,msdos1)

(hd0,msdos5)

En este caso la salida del comando nos muestra que disponemos de un disco duro (hd0) que contiene dos particiones. Además sabemos que las particiones son de tipo msdos, es decir que contienen una tabla de particiones de tipos dos, con espacio MBR. Si queremos información más detallada, igual que para el comando del sistema Linux, podemos emplear la opción -l.

```
grub> ls -l
```

En este caso, veremos que la salida es más detallada pero esencialmente distinta de la que muestra el mismo comando cuando es capaz de mostrar ficheros o directorios.

También disponemos de la funcionalidad para ls de listar ficheros y directorios siempre que especifiquemos un sistema de ficheros válido. Para ello, podemos recordar las particiones que acabamos de listar o podemos recordar simplemente la variable \$root. Los dos comandos siguientes son equivalentes:

```
grub> ls (hd0,msdos1)/
```

```
grub> ls ($root)/
```

La manera de trabajar con GRUB y con el terminal de Linux, sea cual sea el que usemos, es bastante similar. En muchas ocasiones podremos usar las funciones de autocompletado. Podremos usar también las flechas del teclado para acceder al historial o para editar el comando en que nos encontramos. En la página del proyecto gnu (<http://gnu.org>) es posible encontrar un manual de referencia completo sobre GRUB que nos puede ayudar si necesitamos realizar tareas más complejas. Aunque, en la mayoría de ocasiones, no debería ser

necesario.

9.7 GRUB, configuración y gestión

La instalación de GRUB suele ser más complicada que su configuración. Por suerte al instalar nuestra distribución de Linux esta se hará cargo, en la gran mayoría de ocasiones, de instalar el cargador de arranque, sea GRUB o cualquier otro. Es por ello, que debe preocuparnos más la configuración que la propia instalación.

La configuración de GRUB se encuentra en un directorio, normalmente en el fichero `grub.cfg` y varios ficheros más que sirven para la configuración de cada uno de los módulos y que suelen contener una extensión `.mod`. A diferencia de otros ficheros de configuración, no encontraremos el directorio de configuración de GRUB en `/etc`. En primer lugar no se trata de un fichero de configuración del sistema operativo, puesto que es anterior a este. En segundo lugar y no menos importante, en tiempo en que GRUB necesite su configuración todavía no disponemos de la jerarquía de directorios del sistema operativo puesto que no hay sistema operativo. Por tanto, el directorio de configuración de GRUB se encuentra en `/boot/` y suele ser `/boot/grub` o `/boot/grub2`.

Igual que sucedía con el fichero `sudoers` en la sección 8.6, no es recomendable modificar los ficheros de configuración de GRUB manualmente puesto que cualquier error puede ser crítico para el arranque del sistema. Suele usarse para modificar los ficheros de configuración el comando `grub-mkconfig`, o en algunas distribuciones `grub2-mkconfig`.

Si nos fijamos en el fichero `grub.cfg` veremos que se trata de un fichero de texto que contiene una ristra de comandos de GRUB.

Estos comandos definen el comportamiento inicial, antes de mostrar opciones al usuario. A continuación, podremos observar una serie de líneas que empiezan con la palabra clave: *menuentry*. Estas líneas definen cada una de las líneas que nos ofrece GRUB en el menú y los parámetros, comandos y opciones que carga cada una de ellas. Así pues, podemos configurar cada opción de arranque nosotros mismos de forma permanente realizando los cambios sobre la entrada de menú correspondiente.

```
menuentry "Install on sdb1" {  
    set root=(hd1,1)  
    Linux /vmlinuz root=/dev/sdb1 ro quiet splash  
    initrd /initrd.img  
}
```

La entrada de arriba es solo un ejemplo de lo que seria una entrada del fichero de configuración. Estas entradas suelen ser bastante más complejas, sin embargo, es suficiente para observar como se estructura. Los parámetros a los que invocamos para la llamada al kernel (Linux) son los mismos que se han comentado en la sección 9.3 Parámetros del kernel. Sobre *initrd* hablaremos más adelante. Para el resto de comandos de GRUB, si requerimos de una configuración compleja podemos recurrir al manual ofrecido por gnu.

Para añadir nuevas entradas al menú de GRUB o hacer cambios significativos en la configuración de GRUB es recomendable, además de hacer copias de seguridad de los ficheros existentes, realizar los cambios en un fichero aparte. Una vez hemos terminado con los ficheros de configuración deberemos añadirlo al directorio */boot/grub/miconfiguracion.cfg*. Otra opción de la que disponemos es

usar los scripts llamados *custom* que encontramos en `/etc/grub.d`. Estos scripts nos permiten copiar y registrar nuestros propios ficheros de configuración para GRUB. La verdad es que no es preferible un método sobre otro, simplemente se trata de una cuestión de costumbres.

Para generar o instalar un nuevo fichero de configuración de GRUB, una vez creado este, deberemos hacer uso de:

```
grub-mkconfig -o /boot/grub/fichero_de_configuración.cfg  
grub-mkconfig -o /boot/grub/grub.cfg
```

Si el fichero no existe, este se creará de nuevo con los parámetros por defecto. Estos parámetros por defecto son muy dependientes de la distribución de Linux que haya instalado el cargador, es por eso que no entraremos en detalles. No debería ser difícil, sin embargo, obtener información al respecto.

9.8 El sistema de ficheros inicial en RAM

Al arrancar el kernel de Linux, hemos visto que debe montarse el sistema de ficheros raíz. Para ello, se requieren una serie de drivers y de ficheros de configuración. Normalmente los drivers se incluyen en módulos del kernel en lugar de incluirse en el propio núcleo del kernel para evitar que el tiempo de carga sea excesivo. Los discos en RAID o cifrados complican todavía más el trabajo de poder montar el sistema de ficheros raíz. Además, los sistemas modernos dan soporte a la opción de hibernación que requiere recuperar el fichero en el que se realizó el volcado de los datos. Para evitar que el kernel considere una variedad inmanejable de casos particulares se hace uso de un sistema de ficheros raíz temporal. Este sistema de ficheros se

conoce como `initramfs` o sistema de ficheros inicial en RAM.

Aclaración

RAID significa conjunto redundante de discos independientes, por sus siglas en inglés. A grandes rasgos, un sistema de RAID es un mecanismo de almacenamiento empleando varios discos que permite redundar o repartir la información entre los discos disponibles. Dependiendo del nivel de RAID que se emplee las ventajas o desventajas serán unas u otras. No resulta complicado encontrar información al respecto en Internet.

Si recordamos las opciones del fichero de configuración de GRUB o las líneas que nos permite configurar desde el menú, recordaremos que había una línea:

`initrd`

Esta es la línea encargada de montar el sistema de ficheros en RAM. El sistema de ficheros, en las versiones actuales, conocido como `initramfs` no es más que un fichero comprimido. Este fichero se descomprime y se monta en memoria RAM en lugar de hacerlo en un dispositivo de almacenamiento permanente. A partir de ese momento substituye al sistema de ficheros raíz hasta que el programa `init` es capaz de montar el sistema de ficheros en disco.

Hay que tener en cuenta que el contenido de este sistema de ficheros cambia en gran medida según la distribución, por lo que el trabajo que debe realizar el proceso de inicio también cambiará. Mientras algunas distribuciones como Debian tienen una información bastante concreta sobre módulos y drivers necesarios, otras

distribuciones como Fedora, generan un sistema de ficheros raíz temporal mucho más genérico, que luego debe ser capaz de buscar y encontrar algunos de los datos necesarios.

En cualquier caso, conociendo la distribución con la que trabaja cada uno y la existencia de initrd, debería ser sencillo encontrar información concreta a la distribución en Internet.

9.9 El programa init

Init es un programa corriente y como tal se encuentra en /sbin. Lo que diferencia a init del resto de programas es que es el primer proceso que se ejecuta después de iniciar el kernel. Init es el responsable de encender, parar y gestionar los servicios esenciales de cualquier sistema. O al menos antes era responsable de estas tareas. Los init modernos tienen un número mayor de responsabilidades. Adía de hoy hay tres implementaciones mayoritarias de este programa:

- **System V init:** Es un sistema secuencial de arranque. En esta implementación se ejecuta o se lanza solo un servicio en cada instante de tiempo. Secuencialmente es muy sencillo resolver dependencias pero el rendimiento puede verse enormemente resentido. Es el que usan distribuciones como Red Hat u otras.
- **Systemd:** Se trata del estándar emergente para el programa Init. Confía en resolver varios servicios al mismo tiempo con una resolución de las dependencias algo más compleja que su predecesor pero con un mejor rendimiento.
- **Upstart:** Upstart es un estándar para Init basado

en eventos, es similar en concepto a `systemd`. Aunque su adopción se dio en varias distribuciones de Linux, muchas de las que lo usaban han migrado en versiones más modernas a `systemd`.

Junto con las implementaciones más conocidas de `Init` aparecen muchas más, por ejemplo Android tiene su propio estándar para `Init`. Sí, android es un sistema Linux y por tanto son aplicables los conceptos aquí descritos.

A diferencia de los estándares tradicionales como `system V` `init`, `systemd` es un estándar orientado a objetivos, requiere que se le especifique el servicio que debe ejecutar y las dependencias que requiere y `systemd` se encarga de solucionar las dependencias en el momento de lanzar el servicio. Además, `systemd` es un estándar de evaluación perezosa o bajo demanda. Es decir, contempla la opción de no lanzar los servicios hasta que estos son necesarios, a diferencia de otros estándares como `system V` que lanzan todos los servicios programados de inicio. `Upstart` también considera el tipo de evaluación perezosa para ejecutar los distintos servicios.

Otra de las diferencias principales entre `systemd`, `Upstart` y otros estándares anteriores como `system V` es la centralización en torno a scripts. `System V` lanza los demonios o servicios mediante scripts que ejecutan el programa que corresponde, el programa se desvincula del script y permanece en ejecución. Usando este mecanismo se suele guardar un fichero con el PID del servicio que se ha lanzado para poder interactuar con él más tarde dado que ha sido desvinculado del script. `Systemd` y `Upstart` por contra no están centrados en scripts si no

que permiten la manipulación de los scripts directamente.

Para identificar que Init tiene nuestro sistema podemos seguir los pasos siguientes:

1. Si existe el directorio `/usr/lib/systemd` y `/etc/systemd` nuestro Linux trabaja con systemd
2. Si existe el directorio `/etc/init` con varios ficheros `.conf` en su interior nuestro sistema Linux tiene Upstart o en el caso de Debian 7.0 o sus derivados es posible que se trate de System V.
3. Si no se cumple ninguno de los anteriores pero existe `/etc/inittab` seguro que tenemos System V.
4. Una manera de averiguar que estándar de init tenemos es recurrir al manual:

`man init`

9.10 Runlevels

Un sistema Linux se puede encontrar en diversos estados en cada momento de su ejecución. Los estados de un sistema operativo complejo se definen por los procesos que se están ejecutando o que se pueden ejecutar en un determinado momento. Este conjunto de estados se conoce como runlevels en las implementaciones System V de Init. Lo comentamos de manera global dado que Upstart y Systemd mantienen compatibilidad con los runlevels para dar soporte a aquellos scripts o servicios que están pensados para ejecutarse únicamente con System V. Sin embargo, tanto Upstart como Systemd consideran el sistema de runlevels obsoleto y las implementaciones difieren bastante entre uno y otro estándar.

Los estados o runlevels se numeran con los números del 0 al 6. El runlevel 0 equivale al sistema apagado y el 6 al reinicio del sistema. El usuario trabaja usualmente en un único runlevel aunque el sistema se encarga de cambiar entre runlevels según convenga. Hay que entender que cada distribución ha usado los runlevels como más le ha convenido, sin embargo, más o menos los significados de cada uno son los que se muestran a continuación. El 0, el 1 y el 6 respetan el significado entre distintas distribuciones.

Número	Nombre	Descripción
0	Halt	Apaga el sistema. El hardware debe soportarlo, aunque hoy en día casi todos lo soportan.
1	Single-user	Modo de recuperación. No se ejecuta ningún demonio ni se configura la red. Únicamente está disponible el usuario root y los sistemas de ficheros. GRUB suele dar la opción de iniciar en este runlevel.
2	Multi-user	Multiusuario sin soporte de red. Lanza los demonios pero no configura ni exporta configuración de red.
3	Multi-user + networking	Inicia el sistema normalmente. Es igual que el runlevel 2 pero con

		servicios de red
4	En desuso	Queda libre para que cada distribución haga con él lo que quiera. Normalmente suele ser igual que el 3.
5	Multi-user + GUI	Inicia el sistema normalmente igual que el runlevel 3 pero además carga el entorno gráfico de usuario. Los usuarios suelen trabajar en este runlevel.
6	Reboot	Reinicia el sistema.

Hasta ahora todo esto parece muy abstracto. Tenemos estados, en cada estado se ejecutan unos programas o servicios y el kernel cambia entre distintos runlevels según le convenga en cada momento. ¿Donde encontramos todo esto en nuestro sistema Linux? La información acerca de los runlevels forma parte de la configuración del sistema y en `/etc/rc1.d/` encontraremos todo los programas o servicios que se ejecutarán para el runlevel1. Si cambiamos el 1 por el número que nos interese veremos los programas correspondientes al runlevel especificado.

Si observamos los directorios del interior de cualquiera de los runlevels observaremos dos detalles cuanto menos curiosos. En primer lugar, todo lo que encontramos es o debería ser un enlace simbólico a scripts que se encuentran, posiblemente, en `/etc/init.d`. Esta es la manera de trabajar de System V Init con los runlevels. En el caso de `systemd` o `Upstart` la manera de trabajar depende de cada

implementación. La segunda cosa que observamos es que todos los nombres dentro de los directorios de los runlevels siguen el mismo formato:

[K o S] + nn + [nombre] #Por ejemplo S01motd

Este formato está justificado y es el que definirá el comportamiento de cada script al entrar al runlevel junto con el orden en que ese programa o servicio se ejecutará o dejará de hacerlo.

- **K (kill):** Indica que el proceso se parará al entrar al runlevel.
- **S (start):** Indica que el programa se ejecutará al entrar al runlevel.
- **nn:** Es un número de dos dígitos que marca la prioridad de ejecución. Mientras más bajo sea el número mayor prioridad.

nombre es simplemente indicativo para que se pueda entender que hace el script. Hay que tener en cuenta que en caso que dos scripts o servicios tengan el mismo número la prioridad será definida según el orden alfabético de nombre.

Si un fichero no empieza con la letra S o K será ignorado por init al arrancar. Si queremos realizar copias de seguridad de nuestros ficheros o queremos cambiar el orden en la secuencia de arranque deberemos procurar que las copias de los ficheros empiecen por un carácter inocuo, por ejemplo '_ '.

Una vez nos encontramos dentro del sistema, o siempre que tengamos acceso a un terminal podemos saber en que runlevel nos encontramos podemos recurrir al comando runlevel o who -r.

runlevel

N 5

who -r

run-level 5 2015-09-06 8:37

En el primer caso la salida nos indica el runlevel con el valor numérico mientras que la N nos indica que este no ha cambiado desde el inicio del sistema. En el segundo caso, who -r, nos devuelve el runlevel y la fecha en que este fue establecido.

Finalmente, para añadir un script a un determinado runlevel, podremos recurrir al comando update-rc.d.

```
update-rc.d servicio start 90 3 5 #Añade S90servicio a los runlevels 3 y 5
```

```
update-rc.d servicio stop 90 0 1 6 #Añade K90servicio a los runlevels 0,1 y 6
```

```
update-rc.d servicio defaults
```

update-rc.d realiza un trabajo que puede ser realizado a mano, si se crean los links simbólicos en los directorios apropiados con los nombre apropiados y los permisos que corresponden nuestro servicio se ejecutará de la misma manera que se ejecutaría cualquier otro servicio. Si no disponemos de update-rc.d y nuestro sistema funciona mediante runlevels, no hay problema en realizar el proceso manualmente.

9.11 System V Init

El estándar de System V para init es uno de los más antiguos de los sistemas Linux. Su uso ha ido desapareciendo de las distintas distribuciones Linux a medida que han sacado versiones modernas,

sin embargo, todavía hay una cantidad importante de distribuciones que usan este estándar. Una instalación típica del programa `init` de System V se compone de un fichero de configuración, usualmente `/etc/inittab`, y de un conjunto de scripts o de links simbólicos a scripts.

El fichero `/etc/inittab` es un fichero de texto donde encontramos una regla por línea. Las líneas que empiezan por el carácter '#' no se toman en cuenta, son comentarios. Cada una de las reglas tiene el formato siguiente:

`uid:rl:acción:comando`

`15:5:wait:/etc/rc.d/rc 5`

- **uid:** Un identificador único. No tiene que ser necesariamente un número, vale con que sea una cadena corta y única, por convención resulta más sencillo emplear números, puesto que si lo hemos hecho bien basta con incrementar en 1 el último valor que veamos.
- **rl:** El runlevel en el que se aplicará la acción. Por comodidad a la hora de trabajar con el fichero de configuración `/etc/inittab` es recomendable agrupar las líneas aplicables a un mismo runlevel.
- **acción:** La acción que `init` deberá realizar. Por ejemplo, para la acción `wait`, `init` esperará a haber completado la ejecución del comando antes de realizar nada más. Algunas de las acciones que puede tomar y sus repercusiones

son:

- **wait:** El proceso se lanzará al entrar en el runlevel e init se esperará a que termine el proceso antes de realizar ninguna otra acción.
- **respawn:** El proceso se volverá a lanzar una vez termine.
- **once:** El proceso se lanzará al entrar al runlevel, init no esperará a que termine el proceso para lanzar otros programas del runlevel.
- **powerwait:** Si init recibe una señal de que hay problemas de corriente o alimentación, se de el apagado, por ejemplo, ejecutará el proceso y esperará a que termine antes de hacer nada más.
- **boot:** El proceso se ejecutará durante el arranque del sistema. En este caso se ignorará el runlevel. Init no esperará a que termine el proceso para lanzar otros.
- **bootwait:** El proceso se ejecutará durante el arranque del sistema. En este caso se ignorará el runlevel y se esperará a que termine el proceso antes de realizar ninguna otra acción.

- **ctralt+del:** Este proceso se ejecuta cuando init recibe la señal de que se ha presionado Ctrl+Alt+del. En la mayoría de sistemas Linux esta combinación de teclas provoca el reinicio del sistema. En servidores que deben estar encendidos 24/7 y que no están en lugares con acceso restringido puede resultar muy interesante deshabilitar Ctrl+Alt+supr como combinación de teclas para apagar o reiniciar el sistema.
- **comando:** Equivale al comando que se ejecutará para la regla. En el caso del ejemplo la acción especificada es realmente importante dado que `/etc/rc.d/rc 5` ejecutará todos los programas que se encuentren en el directorio `/etc/rc5.d/`.

Existen más acciones que permiten matizar como queremos que init interprete los servicios a lanzar. Podemos encontrar estos matices o acciones recurriendo al manual: `man inittab`.

Inittab junto con los runlevels y los links simbólicos que encontramos en `/etc/init.d` y que se han comentado en la sección 9.10 Runlevels, nos permite entender como arranca el espacio de usuario en System V. Al ejecutarse el servicio init se lee el fichero `/etc/inittab` y se van ejecutando cada una de las reglas y acciones definidas según los runlevels.

System V utiliza un mecanismo conocido como run-parts para ejecutar los scripts que se encuentran en `/etc/init.d/`. Este programa está

pensado para ejecutar todos o algunos de los programas en un directorio concreto siguiendo un orden. No es necesario conocer mucho acerca de run-parts, pero un buen símil para entenderlo es que es similar a ejecutar ls sobre un directorio y ejecutar después todos los programas que nos devuelva ls ordenándolos, por ejemplo, por orden alfabético. Este es el trabajo de run-parts i es lo que permite especificar unas prioridades en /etc/rc*.d/.

9.11.1 Cambiar de run level

System V permite cambiar el runlevel una vez nos encontramos dentro del sistema. El cambiar de runlevel puede ser útil para forzar a leer algunos ficheros de configuración o incluso para cargar dependencias que no se han satisfecho correctamente. Para poder realizar el cambio recurriremos al comando telinit.

telinit 3

El comando anterior cambiará al runlevel 3. Además de cambiar entre runlevels el comando telinit nos permite cargar de nuevo la configuración en /etc/inittab. Por defecto la configuración en este fichero se carga únicamente cuando el kernel lanza el programa init. Si hemos estado trabajando con el fichero /etc/inittab y necesitamos aplicar los cambios inmediatamente por que queramos realizar una acción, al apagar el sistema por ejemplo, podemos forzar la carga de las reglas en /etc/inittab de nuevo.

telinit q #Recarga las reglas de /etc/inittab

9.11.2 Interactuar con los servicios y demonios

Aunque System V trabaja con granjas de links que se ejecutan

mediante las acciones especificadas en `/etc/inittab`, es posible interactuar con los servicios o demonios que son ejecutados por el programa `init`. Para ello simplemente debemos conocer el nombre del script o programa que lanza el servicio o demonio en el directorio `/etc/init.d`. Después podremos interactuar con él mediante las opciones: `start`, `stop`, `status` y en algunos casos `restart`.

```
/etc/init.d/apache2 start
```

```
/etc/init.d/apache2 stop
```

```
/etc/init.d/apache2 restart
```

```
/etc/init.d/apache2 status
```

La opción `status` pasada a un script o servicio como se muestra, devolverá un mensaje informando si el servicio se encuentra en ejecución o se encuentra parado. Normalmente `status` basa su información en un conjunto de ficheros que guardan el PID de los servicios que ha ejecutado `init` o el usuario manualmente. En algunos casos es posible que no se haya creado correctamente el fichero con el PID o que no se haya borrado correctamente al terminar el proceso. En cualquier caso, aunque `status` es muy útil en la mayoría de ocasiones, puede ser necesario comprobar el estado del proceso mediante el comando `ps`.

9.12 Systemd

`Systemd` es una de las implementaciones más nuevas para el programa `init`. A diferencia del `init` de `System V`, `systemd` engloba muchas más tareas y responsabilidades. `Systemd` además de manejar los servicios de nuestro sistema Linux es capaz de montar sistemas de ficheros, monitorizar la red, lanzar temporizadores y otra gran miriada

de tareas.

Este conjunto de funciones que ahora es capaz de realizar systemd no vienen solas. La complejidad tanto en el procesado como en la configuración del sistema ha crecido muchísimo al implementar este Init. Pese a la complejidad, parece que cada vez son más las distribuciones que incluyen systemd. A diferencia de sus competidores, systemd hizo pública una interfaz de programación para que los desarrolladores que quieran puedan trabajar con ella. Quizás este es uno de los motivos de su amplia aceptación.

Antes de adentrarnos un poco en la complejidad de systemd vamos a ver, a grandes rasgos, que sucede después de iniciar el kernel.

1. Systemd carga su configuración
2. Systemd determina su objetivo de arranque.
Recordemos que systemd a diferencia de otros como systemV, trabaja por objetivos. El objetivo por defecto de systemd suele estar definido en el fichero `default.target`.
3. systemd determina todas las dependencias que debe satisfacer para lograr el objetivo. En otros Init como el de SystemV esto no era necesario porque el lanzamiento de los servicios era secuencial. Systemd permite el lanzamiento de los distintos servicios en paralelo.
4. Systemd activa las dependencias y el objetivo de arranque.

5. Después del arranque `systemd` se queda a la espera de eventos del sistema, `uevent` por ejemplo, para recordar la utilidad de `uevent` podemos volver a la sección 3.5. Esto significa que `Udev` pasa a estar integrado dentro de `systemd`.

Como hemos comentado, `systemd` asume más responsabilidades además de lanzar los servicios al iniciarse el sistema. Cada una de las capacidades de `systemd`, es decir, cada una de las cosas que puede hacer se conoce como un tipo de unidad. Así pues, encontraremos entre otros, los siguientes tipos:

- **Unidades de servicios:** Controlan los servicios.
- **Unidades de montaje:** Controlan el montaje de sistemas de ficheros
- **Unidades objetivo:** Controlan otras unidades. Es el caso de la unidad objetivo que se trata de alcanzar después de arrancar el kernel. Normalmente, `default.target` es la encargada de agrupar el resto de las unidades necesarias para poner el sistema en funcionamiento.

9.12.1 Systemd y las dependencias

En sistemas como `systemV` `init` las dependencias no suponían un problema mayor que el de ordenar la secuencia de arranque para que los distintos servicios se lanzasen en el orden adecuado. En el caso de `systemd` las dependencias se diferencian por tipos, pudiendo usar un tipo u otro según lo grave que sea un fallo en la inicialización de estas. Ver algunos de los tipos que podemos encontrarnos nos

ayudará a comprender mejor que significa el tipo de la dependencia:

- **Requires:** Se trata de una dependencia estricta. Si `systemd` no es capaz de activar la dependencia desactivará la unidad.
- **Wants:** `Systemd` activará la dependencia antes que la unidad, el servicio por ejemplo, a continuación activará la unidad sin preocuparse si la ejecución de la dependencia se ha podido llevar a cabo o no.
- **Requisite:** La dependencia debe estar activa en el momento de activar la unidad. Si no lo está `systemd` no probará de activarla, simplemente fallará y desactivará la unidad.

Hay otros tipos de dependencias que podremos definir en los distintos ficheros de configuración. Más adelante, veremos como trabajar con los ficheros de `systemd` y como configurar el inicio. Las mismas dependencias que pueden ser especificadas para una unidad pueden ser especificadas a la inversa. Por ejemplo: supongamos que el servicio A depende del servicio B. En este caso, en la configuración de la unidad A podemos especificar una dependencia de tipo *Requires* o bien, en la configuración de la unidad B podremos especificar *RequiredBy* haciendo referencia al servicio A.

Para ver las dependencias de un tipo determinado de una unidad podemos recurrir al comando `systemctl` (`system control`). Este comando es muy útil a la hora de interactuar con `systemd`, dado que es la principal herramienta para controlarlo. En el caso que nos ocupa, para las dependencias, deberíamos usar:

systemctl show -p Wants unidad

Donde Wants puede ser cualquiera de los tipos de dependencia válido mientras que unidad es la unidad (siempre refiriéndonos a unidades de systemctl) de la que queremos conocer las dependencias.

Además de las dependencias, systemd permite especificar prioridades en la ejecución de las distintas unidades. Para ello no se fija en el nombre de los ficheros a ejecutar como sucedía con system V. En este caso las prioridades se definen al igual que las dependencias: en la configuración de las unidades. Tenemos dos palabras clave que permiten controlar el orden: *before* y *after*, antes y después en inglés. Así pues si queremos que un servicio se ejecute forzosamente antes que otro, además de las dependencias podremos especificarlo con la propiedad `before=<programa antes del que debe ejecutarse>`.

9.12.2 Configurar Systemd

Lo primero que debemos hacer para poder configurar systemd a nuestro gusto es localizar donde se encuentran los ficheros de configuración. Estos pueden encontrarse distribuidos por todo el sistema, aunque suele ser habitual que se encuentre en:

- **/usr/lib/systemd/system:** Las configuración de las distintas unidades de systemd
- **/etc/systemd/system:** La configuración del propio systemd

Como norma general es recomendable no realizar ningún cambio en los ficheros que se encuentran en /usr, siempre es preferible realizar los cambios en /etc. La distribución o nuestro sistema ya se

encargarán de mantener la integridad de los datos en el resto de directorios. Si sabemos que es lo que estamos haciendo siempre se puede modificar cualquier fichero, realizando, eso sí, copias de seguridad previamente.

Si somos incapaces de encontrar los ficheros de configuración podemos recurrir de nuevo a `systemctl`:

```
systemctl -p UnitPath show
```

Una vez hemos localizado donde se hallan los ficheros de configuración de las distintas unidades podremos trabajar con ellos. Los ficheros de configuración de estas unidades son algo distintos a los ficheros de configuración que hemos visto hasta ahora. En primer lugar tienen secciones. Cada sección empieza con el nombre de la sección entre los caracteres []. En segundo lugar, cada sección tiene sus valores y atributos, estos no se representan en una sola línea sino que cada atributo u opción se escribe en una línea en el formato:

palabra clave=valor

Algunas de las palabras clave ya las hemos visto, puesto que corresponden a las opciones de orden y a de dependencias. Las palabras clave que existen son muchas y dependen en gran medida del tipo de sección. Las secciones que podemos tener son también limitadas y dependen en gran medida del tipo de unidad. Todo fichero debe contener al menos la sección "Unit". La sección "Install" aunque opcional, también es común a todos los tipos de unidad. Además de estas dos secciones cada unidad debe contener una sección con el nombre del tipo de unidad. Algunos de estos tipos son:

- **socket**: Para elementos de red.

- **device:** En el caso de dispositivos.
- **mount:** Para juntar sistemas de ficheros con directorios en el sistema. Podemos revisar la sección 4.9 Montando un sistema de ficheros para más información.
- **swap:** Permite trabajar con particiones y espacios de swap. Podemos revisar la sección 4.3 La partición SWAP o espacio de SWAP para más información.
- **service:** Permite trabajar con los servicios.
- **target:** Permite trabajar con otras unidades.

Hay bastantes más tipos, de hecho uno por cada tipo de unidad que acepta systemd. Es posible encontrar más información al respecto recurriendo al manual:

`man systemd`

`man systemd.unit`

Cada uno tiene sus propios atributos y opciones, obligatorios u opcionales. Para conocer como configurar cada tipo de unidad es recomendable recurrir al manual para estar seguros de que se realizará la configuración correctamente. En el caso que nos ocupa, comprender el funcionamiento de systemd, tendremos suficiente con conocer las secciones comunes y entender que no son las únicas existentes. Antes de entrar en más detalles vamos a ver un ejemplo del fichero de configuración de una unidad.

[Unit]

Description=Some HTTP server
After=remote-fs.target sqlldb.service
Requires=sqlldb.service
AssertPathExists=/srv/webserver

[Service]

Type=notify
ExecStart=/usr/sbin/some-fancy-httpd-server
Nice=5

[Install]

WantedBy=multi-user.target

Como podemos observar, las secciones Unit e Install están presentes. La tercera sección, Service, nos indica que estamos viendo el fichero de configuración de una unidad de tipo servicio.

La sección Unit describe cuestiones genéricas de la unidad y es donde se expresan tanto las prioridades como las dependencias. Las dependencias que se deben expresar en esta sección son únicamente las directas. Es decir, aquellas que se expresan como RequiredBy no entran dentro de la sección Unit. Algunas de las opciones que podemos encontrar en Unit además de las ya comentadas son:

- **Description:** Una descripción de la unidad. No es obligatorio y el único propósito es mostrar al usuario una descripción comprensible del tipo de unidad de que se trata.
- **OnFailure:** Especifica una lista, separada por

espacios, de unidades que deben ejecutarse en caso de que la unidad donde se especifica la opción falle.

- **IgnoreOnSnapshot:** Toma como valor true o false, verdadero o falso en inglés. En caso de estar a false, la unidad se ignorará al hacer un snapshot de systemd. No recomendable deshabilitar esta opción si no se tiene muy claro lo que se quiere hacer. Por defecto toma el valor cierto.
- **AssertPathExists:** Comprueba que una ruta existe. En caso de que no exista devuelve un error y la ejecución de la unidad se considera fallida.
- **ConditionPathExists:** Igual que la anterior. La diferencia radica en que la ejecución de la unidad no se considera fallida en caso que el directorio no exista. El servicio, por ejemplo, no se ejecuta, pero se satisfacen las dependencias y la unidad se considerará ejecutada correctamente.

Hay varias opciones que incluyen condiciones o Asserts de manera que es posible incluso comprobar la arquitectura del sistema antes de realizar una u otra acción. Es posible encontrarlas todas en las páginas del manual.

La sección Install a diferencia de la sección Unit es opcional.

En la sección Install se definen aquellas opciones que únicamente se tomarán en cuenta al habilitar o instalar la unidad, pero no en cada una de las ejecuciones. Install si que acepta las dependencias del tipo RequiredBy. Además acepta algunas otras opciones como:

- **Alias:** Una lista de nombres separada por espacios que la unidad debe recibir al ser instalada.
- **Also:** Una lista de unidades adicionales, separadas por espacios, que deben ser instaladas conjuntamente con la unidad donde se especifica la opción.

En los distintos parámetros de configuración se pueden emplear el uso de especificadores. Los especificadores son variables que toman valores concretos que permiten concretar alguno de los valores de una unidad. No todos los especificadores son aceptados en todas las secciones. Sin embargo, los manuales de cada una de las secciones deberían concretar que especificadores aceptan. Algunos de ellos son:

- **%I:** Nombre de la instancia. Permite definir una única unidad para varias instancias. Para ello se emplea el carácter @. Si el nombre de una unidad es getty@ y se lanza getty@tty1, %I será substituido por tty1
- **%u:** Nombre de usuario que está configurado en la Unidad. Si no hay ningún usuario configurado para la unidad toma el valor del usuario que ejecuta systemd, normalmente root.

- **%n:** Nombre completo de la unidad.
- **%H:** el nombre de red de la máquina en el momento en que systemd lanza la unidad. Se debe ser cuidadoso %H porque el nombre de red podría ser inexistente o uno no deseado hasta que se cargan los módulos de red.
- **%v:** La versión del kernel. Equivalente a la salida del comando *uname -r*.

9.12.3 Systemctl y journalctl

Systemctl es la aplicación o el comando que gobierna la mayoría de interacciones con systemd. Es por ello que es importante tenerla en cuenta. Vamos a ver algunas de las tareas que podemos llevar a cabo mediante esta herramienta. Seguiremos la misma convención que al revisar los parámetros de bash. Los valores entre [] serán valores opcionales mientras que los especificados entre < > serán parámetros obligatorios.

systemctl list-units [patrón]

Permite listar todas las unidades conocidas por systemd. Si se especifica un patrón solo aquellas unidades que coincidan con el patrón serán listadas. Por defecto no se considera ningún patrón.

systemctl start <unit1 unit2 ...>

Inicia una unidad. La acción de iniciarla no es la misma que la de habilitarla. Las unidades que no se encuentran habilitadas o en un estado de fallo no coincidirán con ningún nombre que se especifique

dado que systemd no las conoce. Antes de iniciarlas será necesario instalar o habilitar las unidades.

systemctl stop <unit1 unit2 ...>

Para una unidad. En este caso la unidad debe encontrarse iniciada, en caso contrario systemd será incapaz de pararla.

systemctl reload <unit1 unit2 ...>

Fuerza que la unidad vuelva a cargar la configuración. Debemos ser cuidadosos, porque fuerza a que la unidad cargue la configuración del servicio, o el tipo correspondiente, no a que systemd recargue su configuración sobre la unidad. Si queremos que systemd recargue su configuración sobre las unidades deberemos hacerlo mediante:

```
systemctl daemon-reload <unit1 unit2 ...>
```

systemctl restart <unit1 unit2 ...>

Permite reiniciar una unidad. No reinicia únicamente la configuración, si no la unidad completa. Si la unidad no se encuentra iniciada la iniciará.

systemctl enable <unit1 unit2 ...>

Instala una unidad según se especifica en la sección Install. Si la sección install no se encuentra en el fichero de configuración de la unidad no se considera ningún link relativo a las dependencias. Después de ejecutar este comando se ejecuta

```
systemctl daemon-reload
```

para asegurar así que systemd carga correctamente la configuración. El hecho de instalar una Unidad no la ejecuta. Para ello debemos recurrir al comando start.

systemctl disable <unit1 unit2 ...>

Desinstala una unidad. Después recarga la configuración de systemd para asegurarse que se ha desinstalado la unidad correctamente. Hay que diferenciar desinstalar de parar. Desinstalar una unidad no la para, por lo que deberemos detenerla antes de poder deshabilitarla.

9.12.3.1 journalctl

Una de las características de systemd es que guarda los logs en formato binario, es decir, no comprensible por humanos. Para poder ver entonces los logs de una unidad o de systemd se debe recurrir a otra herramienta: journalctl.

Por defecto journalctl nos muestra los logs del sistema. Esto es el equivalente a inspeccionar los logs del kernel durante el arranque leyendo el archivo */var/log/messages*, podemos recordar algunas cosas volviendo a la sección 9.1 Los mensajes de Inicio.

journalctl permite especificar una unidad concreta de la que obtener el log, un programa o incluso un nombre de usuario. Esto es porque systemd pasa a gestionar los logs del sistema, cosa que no hace SystemV.

La mejor manera de conocer todas las opciones de journalctl es recurrir a las páginas del manual, sin embargo vamos a tratar de ver a través de algunos ejemplos, algunas de las opciones que nos ofrece journalctl

```
# Permite obtener los logs referentes al usuario con UID 1001
journalctl _UID=1001
# Permite obtener los logs para una unidad concreta
journalctl _SYSTEMD_UNIT=unidad
#Permite filtrar los logs por un rango de fechas
journalctl --since='2015-02-29 00:01' --until='2015-03-29 00:01'
#Permite mostrar los mensajes que se corresponden con un PID
journalctl _PID=123
```

9.12.4 La polémica tras systemd

Systemd ha levantado, desde su salida, una gran polémica. Aunque no voy a posicionarme, intentaré exponer los argumentos de aquellos que se oponen a systemd. Luego es responsabilidad de cada uno valorar estos argumentos y decidir que opinión le merece este sistema.

Systemd como primer programa que se ejecuta en el sistema toma el PID=1. Esto sucedía igualmente con el init de System V. El PID 1 tiene una característica especial y es que el sistema no puede funcionar si no hay un proceso que tome este PID. Como hemos visto cualquier actualización en las unidades de systemd requiere un reinicio del demonio. Si el demonio de systemd es incapaz de arrancar nuestro sistema Linux se apagará. Algunos de los detractores de systemd alegan que los sistemas Linux no pueden permitirse el tener que reiniciar el sistema por cada actualización que se realice.

Otro de los motivos que plantean los detractores de systemd es que este sistema rompe con la mentalidad Unix. Unix siempre ha intentado caracterizarse por el lema: "Haz una cosa y hazla bien" para venir a defender un sistema modular, poco acoplado y donde la

robustez de los programas estuviese demostrada. En el caso de systemd defienden que ha aglutinado demasiadas tareas, rompiendo completamente el principio de diseño de Unix. Es cierto que Linux no es Unix pero desciende de Unix y son muchos los detractores que presentan este argumento.

La comunidad en torno a systemd es otro de los motivos en contra de systemd. Linux y sus derivados son open source, de código abierto, esto significa que la comunidad tiene acceso a su código y a trabajar con él. El tratamiento de los responsables de systemd con la comunidad ha dejado muchos episodios, algunos soberbios, que han enfadado a la comunidad implicada con Linux. El argumento de la comunidad se ha vuelto muy potente en contra de systemd ya que se dice de Linux que debe ser un proyecto Community Driven. Es decir, llevado por la comunidad, sin aceptar imposiciones de gobiernos ni de grupos de presión. Esto en un mundo como el de hoy es muy difícil a todos los niveles. De una u otra forma, el trato de los responsables de systemd con la comunidad Linuxera ha levantado algunas ampollas.

Otro punto que no ha gustado nada es que systemd guarde los ficheros en binario en lugar de hacerlo en texto plano. Esta queja responde igualmente a la filosofía de transparencia de los sistemas opensource y de la sencillez de comprender lo que se está configurando sin que se requiera un programa que hable en nombre del usuario. Por otro lado, es cierto que los ficheros binarios ofrecen un mejor rendimiento que los ficheros de texto.

Finalmente, otro de los puntos a considerar son las dependencias que esta generando systemd y que no gustan. En general el acoplamiento de las funciones de systemd es un problema para muchos de sus detractores. Systemd no funciona en sistemas no

Linux, es decir no funciona en derivados de Unix como FreeBSD, esto provoca que muchos creen también que traiciona la filosofía de los sistemas operativos Linux. Todo nos conduce a que aparecen programas como gnome3.8, que es un entorno gráfico de escritorio para usuarios Linux, que empiezan a depender de systemd. Esto provoca que estos programas dejen de funcionar en sistemas no Linux, levantando más ampollas todavía en los defensores de la filosofía Unix.

Estos y algunos más son los argumentos presentados. En Internet han corrido ríos de tinta o de bytes sobre el asunto así que no será difícil encontrar más información. Dejo a la responsabilidad del lector evaluar unos y otros argumentos y decidir que hacer. Lamentablemente la elección del primer programa que se inicia, va ligado a la distribución y systemd no se deja desinstalar fácilmente, con lo que muchas veces no se tendrá elección.

9.13 Upstart

Upstart es un programa de inicio que adoptaron Ubuntu y otras distribuciones. Adía de hoy ha perdido mucho terreno respecto a systemd que es el que se está imponiendo como opción mayoritaria para substituir a system V. Upstart a diferencia de systemd es orientado a eventos. Al lanzarse el programa se genera el primero de los eventos por parte de Upstart. El primer evento desencadena la ejecución de algunos programas de inicio que a su vez desencadenan nuevos eventos. Una vez el sistema esta ha arrancado completamente, Upstart queda a la espera de nuevos eventos y es capaz de gestionarlos.

Además las “unidades” con las que trabaja Upstart son conocidas como trabajos. Los trabajos son lanzados por los eventos y

se conocen principalmente dos tipos de trabajo:

- **Trabajos de tipo tarea:** Tienen un final definido. Empiezan y acaban. Un ejemplo muy claro sería la tarea encargada de montar todos los sistemas de ficheros. Una vez montados todos los sistemas de fichero conocidos la tarea termina y el proceso asociado deja de existir.
- **Trabajos de tipo servicio:** Son los servicios tradicionales de Systemd y System V. En este caso el programa no debe tener un final definido, puede permanecer por un tiempo indeterminado a la espera de eventos o procesando a intervalos. Se trataría por ejemplo de un servidor web. Es decir, el servidor web y las páginas web que ofrece están siempre disponibles hasta el momento en que suceda el evento de apagado.

El proceso de inicialización que sigue Upstart, sería a grandes rasgos, el siguiente:

1. Carga su fichero de configuración y los ficheros de configuración de los trabajos
2. Lanza el primer evento, se conoce como startup event. A la acción de lanzar un evento se la conoce como emitir un evento.
3. Lanza los trabajos que deben ser lanzados al recibir el primer evento.
4. Los nuevos trabajos generan nuevos eventos. Los

nuevos eventos lanzan nuevos trabajos. Es una ejecución en cascada. Cuando no hay nuevos eventos Upstart queda a la espera.

La mayoría de instalaciones de Upstart tienen un mecanismo de ejecución similar al que se presenta a continuación:

1. Se lanza mountall y otros procesos con la señal startup. Mountall se encarga de montar todos los sistemas de ficheros conocidos por el sistema y de generar las señales apropiadas.
2. Se ejecuta, entre otros, Udev que se encarga de gestionar los dispositivos conectados.
3. Una vez hecho esto se lanzan los trabajos que configuran las interfaces de red.
4. Sigue con la ejecución del resto de servicios.

Aunque el orden que se presenta es bastante superficial, lo importante es el concepto subyacente. Upstart se encarga en primer lugar de los sistemas de ficheros, los dispositivos y la red y en última instancia del resto de servicios. Las señales generadas entre estos trabajos pueden lanzar otros trabajos, pero estos sean quizás los más importantes.

9.13.1 Configuración de Upstart

Upstart mantiene la configuración de todos sus trabajos en /etc/init. Dentro de este directorio todos los ficheros que tengan la extensión .conf son tratados como ficheros de configuración de trabajos. A diferencia de System V, el nombre en Upstart no define la prioridad en la que los trabajos son lanzados. Al ser un sistema

orientado a eventos, serán estos los que marquen el orden en el que los distintos trabajos se irán ejecutando.

Los ficheros de configuración de Upstart son ficheros de texto plano en que cada línea o párrafo presenta una opción. Todos los ficheros de configuración de trabajos deben tener al menos un párrafo `exec` o `script`.

```
exec /bin/foo --opt -xyz foo bar
```

```
script
  # hace cosas
  if [ ... ]; then
    ...
  fi
end script
```

Exec proporciona la ruta a un binario ejecutable dentro del sistema y sus posibles argumentos para que sea ejecutado. Por contra `script` proporciona un shell script que será ejecutado. Dentro de un párrafo de `script` podemos encontrar una o varias especificaciones `exec`. Todo el código de `script` que se proporcione a Upstart se ejecutará empleando el terminal `/bin/sh` con lo que se debe cuidar que la sintaxis sea adecuada para este terminal.

Además, los ficheros pueden contar con otro tipo de párrafos o líneas. Lo más sencillo para conocerlos todos es recurrir a las páginas del manual. Sin embargo, hay algunos que resultan especialmente interesantes:

```
start on startup
start on runlevel [23]
```

emit startup

Es el caso de “start on” o de emit. “start on” o “stop on” permiten definir cuando el trabajo debe lanzarse o pararse. Ambos pueden recibir como parámetros el nombre de un evento o un runlevel. En el caso del nombre del evento estaremos especificando que el trabajo debe lanzarse en cuanto suceda el evento. En el caso de especificar un runlevel el trabajo se lanzará o parará en cuanto se cambie a dicho runlevel. En el caso del ejemplo, el trabajo se lanzaría al entrar en el runlevel 2 o en el 3. El funcionamiento de los runlevels es el que se ha descrito en la sección 9.10 Runlevels.

emit permite lanzar un evento. Esta funcionalidad puede ser muy útil para ejecutar otros trabajos según un orden de prioridad que nos convenga. Los eventos pueden ser definidos por el usuario. Además, hay una serie de eventos conocidos que pueden ser usados para poder realizar una planificación medida y ajustada. Algunos de ellos son:

- **control-alt-del:** Evento que se produce cuando se da la pulsación de las tres teclas mencionadas al mismo tiempo.
- **startup:** Es el primer evento que lanza Upstart al iniciarse.
- **power-status-changed:** Es un evento que se genera cuando cambia la alimentación de corriente de la máquina. Por ejemplo si se mantiene apretado durante varios segundos el botón de apagado y el sistema se va a apagar.

- **local-filesystems**: Se lanza este evento cuando los sistemas de ficheros locales han sido montados.
- **session-end**: Se lanza este evento cuando un usuario termina su sesión.

9.13.2 initctl

De la misma manera que en systemd teníamos systemctl, Upstart ofrece la herramienta initctl para interactuar con él. initctl nos permite diversas tareas, algunas de ellas tan útiles como la posibilidad de enviar señales en cualquier momento.

initctl start <trabajo> [clave=valor, ...]

Permite lanzar un trabajo, independientemente de si se ha satisfecho o no el evento que lo lanza. La lista de parejas clave=valor permiten pasar parámetros al trabajo que se está intentando lanzar. Si el programa ya se encuentra en ejecución, este comando lanzará una nueva instancia. Si no se quiere que un programa o servicio pueda tener múltiples instancias, por ejemplo no nos interesa tener dos servidores web, es posible especificar en el fichero de configuración la opción

instance

Si se intenta lanzar un trabajo configurado como instancia y este ya se está ejecutando se obtendrá un error.

initctl stop <trabajo> [clave=valor,...]

Permite parar un trabajo. De nuevo la lista de parejas

clave=valor permiten pasar parámetros al trabajo a parar.

initctl restart <trabajo> [clave=valor,...]

Permite relanzar un trabajo. Hay que tener en cuenta que el trabajo que se relance retendrá su antigua configuración. Si se quiere que el trabajo cargue una nueva configuración debe pararse primero con el comando stop y lanzarlo posteriormente con start.

initctl stop trabajo

initctl start trabajo

initctl status <trabajo> [clave=valor,...]

Permite conocer el estado de uno de los trabajos. La salida de este comando se muestra en la salida estándar de una forma similar a la siguiente:

job start/running, process 1234

initctl emit <evento>[clave=valor,...]

Permite generar un evento. El conjunto de parejas clave=valor son parámetros que son pasados como parte del evento, no al propio evento. El usuario o administrador es libre de inventarse los nombres de evento que quiera. Para que un evento sea útil, algún fichero de configuración debe recogerlo. Lo mismo sucede con los eventos en el fichero de configuración de los trabajos.

initctl reload-configuration

Solicita al programa init que recargue su configuración. Aunque pueda parecer un comando útil, la verdad es que Upstart ya cuida de cargar la configuración automáticamente mediante métodos

de vigilancia de cambios en los directorios donde se encuentran los ficheros de configuración.

9.14 Apagado del sistema

El programa `init` es también el encargado del apagado del sistema. Si no el responsable directo, si en gran medida. En los sistemas Linux disponemos del comando `shutdown`. Este comando dependiendo de las opciones se encarga de apagar el sistema de una forma controlada y segura. Su funcionamiento a grandes rasgos es el siguiente:

1. Se notifica a todos los usuarios conectados al sistema que este va a apagarse.
2. Se bloquean todas las operaciones de entrada al sistema.
3. Se solicita a todos los procesos que terminen su ejecución. Si recordamos la sección 6.2 Señales, esto se hace mediante la señal `SIGTERM`.
4. Si la señal `SIGTERM` no permite terminar el proceso correctamente, pasado un tiempo `init` manda la señal `SIGKILL` a los procesos que no han terminado.
5. Se bloquean todos los sistemas de ficheros en uso.
6. Se desmontan los sistemas de ficheros.
7. Se monta el sistema de ficheros raíz en modo de solo lectura.
8. Se escriben los datos pendientes a disco.
9. Se le indica al kernel la acción a realizar.

El último paso depende en gran medida de las opciones de shutdown. Básicamente este comando lanza los eventos correspondientes en Upstart, activa las unidades que corresponde en systemd o cambia el runlevel en System V. Independientemente de cual sea la manera de invocarlo el procedimiento suele ser el mismo.

Aunque una de las opciones que recibe shutdown, permite especificar el tiempo que tardará en iniciarse el programa de apagado del sistema, los dos primeros puntos se realizan en el momento de ejecutar el comando. Esto significa que si se pretende empezar el proceso de apagado 1 hora después de ejecutar el comando, los usuarios no podrán entrar al sistema durante 1 hora. Hay que ser pues, cuidadosos con el comando shutdown.

Normalmente el apagado del sistema solo puede ser llevado a cabo por el administrador de la máquina o por usuarios autorizados, por lo que si obtenemos un error indicándonos que el comando no existe, es posible que no tengamos permisos para ejecutarlo. Sea como sea, siempre puede ser útil recurrir al manual en caso de duda. Estas son algunas de las opciones que nos ofrece el comando shutdown:

- **-k:** No apaga el sistema. Pero envía los mensajes de advertencia como si realmente se fuese a iniciar el proceso de apagado.
- **-r:** Reinicia el sistema.
- **-h:** Realiza un halt del sistema después del shutdown. El shutdown implica parar el kernel. Una vez parado el kernel el sistema ya no está en funcionamiento. Un halt implica, además,

parar todas las CPUs de la máquina. La alternativa al halt implica un poweroff. Poweroff a diferencia de halt lo que hace es enviar una señal para cortar la alimentación de las CPUs.

- **-P:** Pide realizar un PowerDown del sistema después del shutdown. Se puede consultar la opción `-r` para entender más sobre esta opción.
- **-c:** Cancela un apagado que se encuentre pendiente. Muy útil si hemos programado un shutdown para dentro de 15 minutos y nos damos cuenta que hemos olvidado algo importante.
- **tiempo:** El tiempo debe especificarse tras las opciones, sin guión delante. Suele expresarse en formato `+m` donde `m` es el número de minutos a esperar antes de realizar el apagado. Si se quiere realizar el apagado en el momento de ejecutar el comando debe introducirse como tiempo la palabra clave `"now"`, ahora en inglés.

Para hacernos una idea exacta de la sintaxis de shutdown veamos un par de ejemplos:

Reinicia el sistema sin demora

shutdown -r now

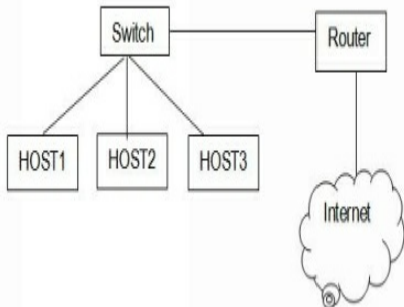
#Apaga el sistema y las CPUS pasados 10 minutos

shutdown -h +10

10. Red

Aunque tener el sistema conectado en red no es imprescindible para su funcionamiento, hace ya unos años que un sistema que no está conectado a otro, aunque no este conectado a Internet, sirve de bien poco. En el mundo de la informática se dispone ya de servidores de disco, redes internas y externas y de una de las redes globales más conocidas por todos: Internet. Así pues, la configuración de la red y sus características se han convertido en una de las cosas esenciales para cualquier máquina y también para los sistemas Linux.

10.1 Breve introducción a las redes de computadores



Las redes de computadores o las redes a partir de ahora, únicamente

buscan la manera de poder enviar información de una máquina a otra. Después será responsabilidad de cada programa el tratar esta información según le convenga. Dado que el envío de información es un trabajo complejo, en computación se resuelve mediante un modelo de capas. Cada capa es responsable de una parte del proceso y es independiente del resto de las capas. A día de hoy existen dos modelos de capas: el modelo OSI y el modelo TCP/IP. El primero de ellos es principalmente teórico, mientras que el segundo es el más extendido y el que se usa hoy en día. Antes de adentrarnos más, vamos a ver una primera fotografía de lo que suele ser una red domestica o de oficina. En muchos de los casos, sobre todo en redes domesticas el elemento que en la imagen aparece como switch es el propio router, que realiza ambas funciones.

Este tipo de red se conoce como LAN (Local Area Network o red de área local). Lo que en el dibujo aparece como host equivale a las máquinas de la red. Ahora podemos entender que el nombre de una máquina en el sistema Linux se conozca por *hostname*, este es el nombre con el que la máquina se identificará en la red.

Seguro que el esquema presentado es conocido por todos: varios ordenadores o dispositivos, móviles por ejemplo, conectados a un mismo router que nos da acceso a Internet o a cualquier otra red. Este ejemplo representa a la mayoría de las viviendas actuales.

Un sistema de red funcional basado en el modelo TCP/IP, que es el modelo con el que trabajaremos en los sistemas Linux, también Windows y MAC, está basado en 5 capas. Cada capa tiene un conjunto de protocolos que realizan las tareas necesarias para cumplir las tareas de la capa. Estos protocolos se definen en unos documentos públicos conocidos como RFC (Request for Comments), así si uno

quiere profundizar en cualquiera de los protocolos de red tan solo tiene que realizar una búsqueda con las palabras clave: protocolo RFC. Donde protocolo es el nombre del protocolo por el que sienta una curiosidad. La torre TCP/IP es la siguiente:



El uso de internet con minúsculas es expreso. De esta manera se pretende diferenciar internet de Internet, la red global a la que accedemos cada día para consultar el correo o leer los periódicos. Cada una de estas capas tiene sus propios mensajes con sus propias cabeceras y cada una de estas capas tiene sus propias direcciones.

- **Capa de aplicación:** Es el nivel más alto, normalmente no comprendido por ninguno de los actores intermedios de la red. Los protocolos que se definen en este nivel especifican como se comunican las aplicaciones entre ellas. Un

ejemplo muy claro y seguro conocido por todos es http.

- **Capa de transporte:** La capa de transporte tiene asociada una “dirección” conocida como puerto. La capa de transporte se encarga de las comunicaciones extremo a extremo, de una aplicación a la otra. Esta capa es capaz de fraccionar los mensajes en trozos más pequeños si así se requiere. En destino será la responsable de recomponer los mensajes fraccionados. Los dos protocolos más conocidos de la capa de transporte son TCP y UDP.
- **Capa de internet:** Se encarga de la interconexión de máquinas. La “dirección” que define cada uno de los extremos se conoce como dirección IP y debería ser única en la red. Este nivel es todavía independiente de la infraestructura que la máquina tenga por debajo. Es decir, tenga el hardware que tenga nuestra máquina dispondrá de dirección IP. En esta capa los protocolos más conocidos son: IPv4, IPv6 e ICMP
- **Capa de red:** Se encarga de gestionar el acceso a la red, de la manera en que la máquina puede encaminar los datos y de como hacer llegar los datos a otros elementos de la propia red. Se asocia habitualmente con la dirección MAC que

es única para cada dispositivo.

- **Capa física:** Sería propiamente el firmware del dispositivo de red que estemos empleando. Esta capa se encarga de gestionar frecuencias, intensidades, voltajes y otros detalles de bajo nivel en los que no vamos a entrar.

Todos los elementos de una red de computadores implementan diversas capas de este modelo de red. Dependiendo del trabajo del dispositivo implementaran más o menos capas. Así por ejemplo, cuando se habla de un router se habla de un dispositivo de red de nivel 3. Hablar de un dispositivo de red de nivel 3 implica hablar de un dispositivo que implementa las 3 primeras capas: es decir Física, Red e internet. Así pues, en principio un router de nivel 3 no debería entender de puertos. A día de hoy es habitual encontrar routers de nivel 4 o switches, que actúan como distribuidores de datos, de nivel 3. Un servidor con toda seguridad implementará las 5 capas de este modelo de red. De otra manera sería incapaz de proveer u obtener ningún servicio a través de la red.

10.2 Un símil de las comunicaciones entre máquinas

Vamos a intentar ver como funciona una comunicación de red mediante una analogía con el servicio de correos. En algunos momentos será necesario ponerse las gafas de la imaginación para que el efecto sea el esperado, pero esperemos que ayude a ver las cosas más claramente.

Supongamos que queremos enviar una carta. Al escribir la carta debemos escribirla en un lenguaje que estamos seguros que la

persona que la reciba va a entender. No solo eso si no que además deberemos escoger en que forma y que nivel de educación debe tener el texto. No es lo mismo una carta para nuestro jefe que una carta para un amigo. Supongamos que es una carta formal, sería entonces algo como lo que sigue:

Ala atención del señor Rodríguez.

Como trabajador de Empresa.SAme gustaría invitarle a Ud. y a su señora a asistir a la inauguración de nuestra nueva planta en Ciudad.

[...]

Esperamos su grata asistencia.

Saludos Cordiales

Nombre

Una vez hemos decidido forma, modales e idiomas la carta está lista para meterla en el sobre. Acabamos de escoger el protocolo de la capa de aplicación. Una vez la carta se ha escrito, la introduciremos en un sobre y anotaremos la dirección y el remitente. Lo primero que escribiremos es a donde debe llegar la carta, en este caso a que despacho y desde donde:

A: Despacho del Director General, planta 5, Edificio Alpha

De: Cubículo 3D, Planta 2, Edificio Omega

Con esto acabamos de especificar la dirección de la capa de transporte. Especificamos el destino final dentro del edificio al que debe llegar. Al poner así la dirección ya hemos escogido el protocolo, aún sin saberlo. Cada edificio tiene su propio repartidor de correo que va repartiendo la correspondencia del día entre las mesas y despachos. En el caso de que el mensaje vaya destinado a alguien en

otro edificio deberemos especificar una nueva dirección, puesto que seguramente a la gente de Correos no sepa que hacer con una carta que tenga como dirección y remitente lo que le hemos escrito hasta ahora. Así pues, cogeremos el sobre y lo introduciremos dentro de otro sobre con la dirección del edificio y el remitente correspondiente. Esto es también lo que sucede en el caso de la red. Al delegar de una capa a la otra se encapsula el mensaje con nuevas cabeceras especificando las direcciones adecuadas.

A: Av. La avenida 49, La ciudad, 00000, Provincia

De: C\La calle 4, Otra ciudad, 00001, Provincia

Acabamos de pasar a la capa de internet y hemos escrito la dirección IP. Una vez tenemos los dos sobres, metemos el paquete resultante en otro sobre. En el sobre escribimos:

Chico del correo

Con esto escogemos la capa de red y escribimos la dirección única, la MAC. Luego se lo damos a nuestra secretaria. Ella conoce al chico del correo y se lo dará en mano. Hemos recorrido todas las capas de protocolos que nosotros teníamos disponibles. Ahora queda en manos de otro.

El chico del correo abrirá el sobre y dentro encontrará otro sobre con las direcciones del edificio. Como claramente son direcciones enviadas a otro edificio, introducirá el paquete en otro sobre y escribirá como dirección:

A: El cartero

De: El chico del correo

El chico del correo de nuestra empresa solo ha podido entender las dos primeras direcciones. Al ver que la segunda no era su responsabilidad ha vuelto a encapsular el mensaje en un nuevo sobre y se lo ha pasado a alguien que si que entiende la siguiente dirección. El chico del correo ha actuado como un switch.

El cartero realizará el mismo proceso que ha realizado el chico del correo, con la diferencia que el cartero si que entenderá la dirección Av\..., la dirección IP, Así pues, la irán intercambiando entre oficinas de correos y carteros hasta que llegue a su destino. Los carteros y las distintas oficinas realizan el papel de todos los routers entre un origen y un destino.

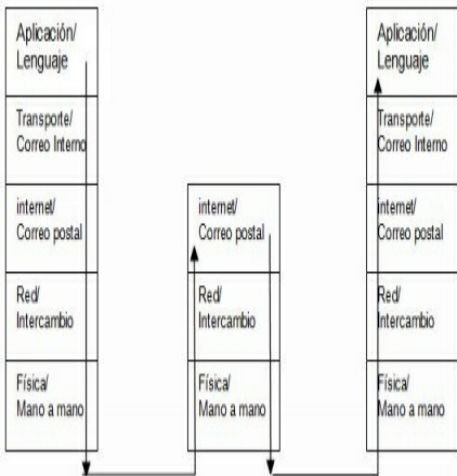
Una vez en su destino, el cartero le entregará al chico del correo del Edificio Alpha una carta que va destinada a él. Al abrirla el chico del correo segundo, verá que dentro el destino es

Av. La avenida 49, La ciudad, 00000, Provincia

Esta, efectivamente, es la dirección del edificio donde se encuentra, su dirección IP, por tanto también abrirá este sobre. Dentro del sobre encontrará otro destinado a:

Despacho del Director General, planta 5, Edificio Alpha

Esta vez la carta no es para él, por tanto no la abrirá. Sin embargo, la carta va destinada a una aplicación de esa máquina. Es decir, conoce donde está el despacho del Director General. Por tanto se la hará llegar. Acabamos de interpretar el puerto y estamos delegando el mensaje a la aplicación.



La aplicación, es decir el Director General o señor Rodríguez, afortunadamente habla el mismo protocolo de aplicación que nosotros. Por tanto al abrir el último sobre entenderá exactamente lo que pone.

10.3 La capa de red o acceso a la red

En los sistemas Linux no tenemos excesivo control sobre la capa de red. Sin embargo, cada uno de los dispositivos de red que tengamos en la máquina tiene una dirección única que es conocida como dirección MAC, del inglés *Media Access Control*. La MAC permite, en principio, identificar de forma única cada dispositivo. Si nuestra

máquina dispone de de n tomas Ethernet o de varios pinchos wi-fi cada uno de ellos tendrá su propia dirección MAC. De la misma manera, cada uno de ellos se corresponderá con una interfaz de red distinta.

Es habitual que un router tenga más de una interfaz de red, puesto que suele estar conectado a múltiples redes simultáneamente. En el caso de los routers domésticos actuales, suelen tener al menos cuatro. La configuración de los routers de las distintas compañías ya es un tema aparte.

Una dirección MAC tiene la forma:

01:23:a2:df:00:ff

En principio esta dirección es única y vinculada al dispositivo de red que estemos usando para conectarnos, sin embargo, en los sistemas Linux es posible cambiarla. Para ello debemos recurrir al comando `ifconfig`. Antes de ver exactamente este comando debemos tener en cuenta que en diversas distribuciones está siendo substituido por una familia de programas más nuevos conocida como `ip`.

`ifconfig` nos permite trabajar con las interfaces de red de nuestro sistemas. Para ello, como muchos otros comandos se vale de parámetros y de diversas opciones. A medida que vayamos adentrándonos en la configuración de red del sistema iremos viendo más utilidades de `ifconfig`.

Si invocamos `ifconfig` sin parámetros nos mostrará una lista de las interfaces disponibles y diversa información sobre estas interfaces. La salida de `ifconfig` debería ser algo similar a lo siguiente:

```
eth0  Link encap:Ethernet HWaddr 09:00:12:90:e3:e5
      inet addr:192.168.1.29 Bcast:192.168.1.255 Mask:255.255.255.0
      inet6 addr: fe80::a00:27ff:fe70:e3f5/64 Scope:Link
```

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:54071 errors:1 dropped:0 overruns:0 frame:0
TX packets:48515 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:22009423 (20.9 MiB) TX bytes:25690847 (24.5 MiB)
Interrupt:10 Base address:0xd020

A la izquierda de todo nos indica el nombre de la interfaz, en este caso eth0. Marcado en gris, podemos ver la actual dirección MAC. Para poder cambiarla, lo que deberemos hacer es:

1. Deshabilitar la interfaz que corresponda.
2. Cambiar la MAC.
3. Habilitar de nuevo la interfaz.

Es recomendable antes de realizar ningún cambio sobre la MAC apuntarla en algún lugar seguro por si luego necesitásemos recuperarla. Los pasos anteriores, con ifconfig serían de la siguiente manera:

```
ifconfig eth0 down  
ifconfig eth0 hw ether 01:02:03:04:05:06  
ifconfig eth0 up
```

Substituyendo la dirección MAC por aquella que nos interese. Es posible que dependiendo del entorno en el que nos encontremos una MAC alterada nos impida hacer cosas que de otra manera podríamos hacer. También podría suceder a la inversa. Después de un cambio de MAC es posible que se requiera reiniciar los servicios de red del sistema Linux para que el cambio tenga efecto.

10.4 Direcciones IP

Aunque la idea es que el protocolo usado en cada una de las capas sea transparente al resto de capas, la verdad es que el protocolo IP y su compañero ICMP de la capa de internet, forman parte casi inseparable de las implementaciones de protocolos de red de cualquier sistema moderno. El protocolo IP pretende que sea posible enviar mensajes de una máquina a la otra independientemente del hardware que haya debajo, independientemente de la topología de red. La idea detrás de IP es una red descentralizada en que existen subredes y una manera de conectar estas subredes entre si. De ahí el nombre de internet: la capa permite la conexión entre redes, en inglés net.

El protocolo IPv4 que es a día de hoy el más extendido, está asociado con las direcciones IPs que son direcciones de aspecto:

a.b.c.d #Es un ejemplo. Una IPv4 real no tendrá letras
127.0.0.1

Estas direcciones son una tupla de 4 números separados por puntos. Cada uno de los 4 números es la representación en base decimal de 8 bits, la dirección completa son por tanto 32 bits. Esto significa que ninguno de los números podrá ser nunca superior a 255. Estamos limitados también, a un total de 4 mil millones de direcciones. Las direcciones IP deben ser únicas en la red donde se encuentran. Aunque esto es una simplificación, un cálculo rápido nos lleva a darnos cuenta que los dispositivos conectados a día de hoy superan con creces los 4 mil millones. Con el propósito de suplir la carencia de direcciones IP, además de otras necesidades, nace IPv6. Este nuevo protocolo permite representar las direcciones IP con 128 bits en lugar

de los 32 de IPv4. IPv6 solucionaría los problemas de direccionamiento del Internet actual. Sin embargo, la implementación de IPv6 todavía no se ha aceptado completamente, seguramente por el miedo a los problemas de compatibilidad que esto pudiese ocasionar. Pese a que la implementación no es completa los sistemas Linux implementan lo que se conoce como doble pila: implementan tanto IPv4 como IPv6.

ifconfig

```
eth0  Link encap:Ethernet HWaddr 09:00:12:90:e3:e5
      inet addr:192.168.1.29 Bcast:192.168.1.255 Mask:255.255.255.0
      inet6 addr: fe80::a00:27ff:fe70:e3f5/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:54071 errors:1 dropped:0 overruns:0 frame:0
      TX packets:48515 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:22009423 (20.9 MiB) TX bytes:25690847 (24.5 MB)
      Interrupt:10 Base address:0xd020
```

Si ejecutamos de nuevo ifconfig, vemos que este nos da, en gris, ambas direcciones. Marcadas como inet e inet6 para IPv4 e IPv6 respectivamente. Si nos fijamos el separador para las direcciones IPv6 es el carácter ':' en lugar de '.'. No solo eso si no que además vemos que faltan números. Esto es debido a que las direcciones de IPv6 se expresan en hexadecimal y permiten la supresión de los 0 intermedios indicándolo con un campo vacío, es decir '::'.

Para ver la dirección ip de nuestra máquina, como ya hemos visto, podemos recurrir a ifconfig o bien, podemos recurrir al nuevo conjunto de comandos.

ip a s # Del inglés ip address show

10.4.1 Subredes

Comprender las subredes y las direcciones IP, así como su funcionamiento, es el primero de los pasos para comprender como funciona internet. Aunque no es el objetivo de este texto profundizar en las subredes, también son uno de los bloques imprescindibles para comprender como funciona cualquier red de computadores y por tanto se intentará dar una visión global.

Una subred es un conjunto de máquinas, cada una con su dirección IP, de tal manera que la dirección IP de estas máquinas, está de alguna manera agrupada u ordenada. Así por ejemplo, todas las máquinas que tengan una IP entre 192.168.10.0 y 192.168.10.255 pueden pertenecer a la misma subred. Todas las direcciones IP están definidas por dos elementos: la dirección IP y la mascara de red. La dirección IP ya la hemos visto. La mascara de red también nos la ha mostrado ifconfig. Dicha mascara nos está indicando que parte de la dirección IP nos indica la red y que parte identifica al host. Tal como sucede con la dirección de las cartas en las que se especifica calle y número de vivienda.

Para saber como relacionar la mascara de red con la IP debemos pasar a notación binaria:

IP: 10.23.2.0 --> 11000000 10101000 00000001 00011101

Mascara: 255.255.255.0 --> 11111111 11111111 1111 1111 00000000

Lo que indica la mascara son los bits de la dirección IP que no pueden ser modificados. Dicho de otro modo, que bits de la dirección IP sirven para especificar la subred a que pertenece la máquina.

10101000 00000001 00011101 00011101

La parte en **negrita** representa la subred. Mientras que la parte no resaltada representa el host perteneciente a la subred indicada. Es importante resaltar que las direcciones de host dentro de una subred con todos los bits a 1 o todos los bits a 0 están reservadas. Si nos fijamos en la salida de ifconfig

ifconfig

```
eth0  Link encap:Ethernet HWaddr 09:00:12:90:e3:e5
      inet addr:192.168.1.29 Bcast:192.168.1.255 Mask:255.255.255.0
      inet6 addr: fe80::a00:27ff:fe70:e3f5/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:54071 errors:1 dropped:0 overruns:0 frame:0
      TX packets:48515 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:22009423 (20.9 MiB) TX bytes:25690847 (24.5 MiB)
      Interrupt:10 Base address:0xd020
```

Una de estas direcciones, aquella que tiene en la parte del host todos los bits a 1, nos la está marcando como una dirección especial. Concretamente nos la está marcando como Bcast, de Broadcast en inglés. Esta dirección permite mandar un mensaje a todos los hosts o servidores de la subred a la que pertenece.

La notación de la máscara de red puede expresarse tal como marca el comando ifconfig o bien nos la podemos encontrar en notación CIDR (Classless Inter-Domain Routing). Esta notación es más sencilla de entender y lo que hace es indicarnos, en notación decimal cuantos bits de la dirección IP se emplean para identificar la subred. Así

por ejemplo:

IP: 192.168.2.1 Mask: 255.255.255.0

Es equivalente a

IP: 192.168.2.1/24

Si nos fijamos en la notación en binario será más sencillo comprender el porque:

11111111 11111111 1111 1111 00000000

Los primeros 24 bits son 1, mientras que el resto son 0s.

Una IP con una mascara con todo 1s está representando de forma unívoca un host. Si los 32 bits de una dirección IP sirven para identificar una subred, significa que la subred está compuesta por una única máquina con la IP expresada.

10.4.2 Convertir una netmask a notación CIDR

Para pasar de la notación de netmask a la notación a CIDR deberemos trabajar en notación binaria. Si tenemos la mascara:

255.240.0.0

11111111 11110000 00000000 00000000

Para saber como sería en notación CIDR únicamente debemos contar el número de 1s. Todos los bits que se encuentren a 1 en la máscara serán bits de la IP que no podrán alterarse puesto que representan la subred a la que pertenece.

Lo mismo sucede a la inversa. Si tenemos una mascara /12 lo único que debemos hacer es escribir 12 1s.

1111111111

y completar con 0s hasta completar las 32 cifras

11111111111000000000000000000000

Luego cada 8 cifras introducimos un espacio y convertimos los 4 números binarios que obtengamos a decimal. Eso nos dará la máscara.

11111111 11110000 00000000 00000000
255 240 0 0

Debemos tomar en cuenta que una máscara tendrá los 1's de forma consecutiva. No debemos considerar máscaras que tengan 0s intercalados entre los 1s.

Juntando conceptos

Si la máscara de red representa bits invariables de la dirección IP, los bits puestos a cero representan la parte variable de la IP, la que nos permite identificar a un host. Esta parte variable, permite representar un número limitado de hosts. Dicho de otro modo, la máscara de red, además de indicar cuál es la subred, limitará el número de IPs, de dispositivos, que pueden estar conectados a una misma subred. Con una máscara /24 nos quedarán 8 bits libres. Estos 8 bits nos permitirían representar un total de 256 IPs distintas. Lamentablemente la primera IP y la última de cualquier subred están reservadas. Una máscara /24 permite, entonces, disponer de 254 IPs distintas en la subred.

10.4.3 Subredes Privadas

Como hemos visto, el número de dispositivos que se pueden conectar a la red está limitado en por el número de bits con que se representan las direcciones. Aunque en IPv6 las posibles direcciones son virtualmente infinitas, si queremos una IP pública, accesible por todo el mundo deberemos pagar por ella a nuestro ISP, nuestro proveedor de servicios, Telefónica por ejemplo.

Para montar una red privada, igual no es necesario disponer de una ip que sea accesible por todo el mundo. Es por ello que en IPv4 hay diversos rangos de direcciones IP que están reservados para direcciones privadas. Estas subredes no son enrutables, es decir no se pueden conectar a otras redes. De esta manera dos organizaciones distintas pueden compartir la misma IP en dos máquinas distintas siempre que estas direcciones IP se encuentren dentro de los rangos reservados para direcciones privadas.

Los rangos reservados son:

- 10.0.0.0/8 o 10.0.0.0 con netmask 255.0.0.0
- 192.168.0.0/16 o 192.168.0.0 con netmask 255.255.0.0
- 172.16.0.0/12 o 172.16.0.0 con netmask 255.240.0.0

10.4.4 Cambiando o asignando una IP a una interfaz

Los sistemas Linux nos dan la posibilidad de especificar manualmente que IP y mascara queremos usar además de darnos la opción de obtenerla automáticamente. Para ello podemos recurrir al comando `ifconfig` o a los comandos de la nueva familia de comandos `ip`.

```
#añade una IP a la interfaz eth0  
ifconfig eth0 127.0.0.1 netmask 255.255.255.0  
ip a a 127.0.0.1/24 dev eth0
```

En el caso de ifconfig tan solo debemos especificar la interfaz y a continuación la dirección IP que queramos junto con la mascara de red. Hay que tener en cuenta que si no se especifica una mascara de red se tomará por defecto una mascara /32 o 255.255.255.255.

Para la familia de comandos ip, se debe especificar la acción: ip a a (ip address add) seguida de la IP, la mascara en notación CIDR y finalmente la palabra clave “dev” seguida de la interfaz a la que queramos añadir la IP.

Juntando conceptos

Cada interfaz de red se corresponde a un dispositivo en los sistemas Linux. Esto significa, por tanto, que cada dispositivo de red tiene también un fichero asociado.

Si recordamos el capítulo3. Dispositivos, recordaremos que se hablaba de unos dispositivos llamados sockets. De una manera bastante simplificada, podríamos decir que los sockets nos permiten enviar mensajes del espacio de usuario a los dispositivos de red. A parte de esto también permiten la comunicación entre procesos igual que los Pipes, pero una de las funcionalidades más importantes es la de permitir la comunicación con los dispositivos de red.

Si queremos borrar las direcciones IP que tenemos en una determinada interfaz podemos recurrir de nuevo a los dos comandos

anteriores:

```
ifconfig eth0 0.0.0.0
```

```
#ip address delete
```

```
ip a d 127.0.0.1/24 dev eth0
```

```
#si tuviésemos más de una IP y quisiésemos borrarlas todas
```

```
ip a flush dev eth0
```

10.4.5 Resolución de nombres de Dominio (DNS)

Recordar una dirección IP es complicado. Recordar diez direcciones IP empieza a ser bastante complicado. Imaginad que cada vez que queréis entrar a Facebook fuese necesario introducir su dirección IP, imaginad ahora que hubiese que hacerlo para todas las páginas web a las que entramos a diario. Para que las direcciones de las distintas máquinas sean más accesibles se emplea un protocolo de resolución de nombres de dominios conocido como DNS. Este protocolo permite que usemos direcciones de máquinas tales como facebook.com. Es el sistema el que se encarga de contactar con un servidor de DNS para resolver el nombre de dominio en una dirección IP que pueda comprender el kernel.

En Linux, podemos ver que IP corresponde a un dominio ejecutando el comando host

```
host <nombre de dominio>
```

DNS funciona como una especie de páginas amarillas en las que se realiza la equivalencia de nombre de dominio con una IP. Almacenan, además, otra información sobre los dominios. Podemos acceder a la información a través de las distintas opciones del

comando host. Algunas de ellas se listan a continuación:

- **-4:** Únicamente muestra la IPv4 de la máquina.
- **-6:** Únicamente muestra la Ipv6 de la máquina.
- **-t <tipo>:** Especifica el tipo de entrada que se quiere recuperar sobre el dominio. Algunos de los tipos son:
 - **A:** IP versión 4.
 - **AAAA:** IP versión 6.
 - **CNAME:** Canonical Domain Name. Sirve para especificar que un dominio es un alias de otro dominio.
 - **MX:** Mail Excahnge. Permite asociar una dirección de correo al dominio.
 - **SOA:** Proporciona información sobre el servidor DNS primario de la zona.
 - **TXT:** Permite asociar un texto cualquiera a un dominio.
- **-a:** Permite obtener toda la información disponible en el registro DNS.

Es posible que no todos los campos del registro DNS para un dominio estén completos. De hecho hay servicios que permiten al poseedor de un dominio pagar para ocultar alguna de la información relacionada con sus dominios. En cualquier caso el comando hosts proveerá lo que pueda conseguir. Una vez conseguida la IP tanto da

acceder a un servicio por su IP o por su nombre de dominio.

10.4.5.1 Como funciona la resolución de nombres de dominio

Hemos visto como host nos puede devolver las IPs de un dominio concreto u otra información de interés. Considerando que hemos hablado de servidores DNS, llegamos a la conclusión que con los servicios adecuados, podemos convertir una máquina Linux en un servidor de DNS. No entra dentro del propósito de este texto explicar como convertir nuestra máquina Linux en un servidor de DNS puesto que es un proceso que variará enormemente entre distribuciones. Sin embargo, si que puede resultar interesante hacernos una idea global de como se resuelve una petición de DNS. Para aquellos que necesiten instalar un servidor de DNS en su máquina, no será difícil encontrar manuales de como hacerlo.

Hay que tener en cuenta que la resolución de DNS se produce en el espacio de usuario, a diferencia de los otros procesos que involucran la red que se resuelven en espacio de kernel. El funcionamiento típico de la resolución DNS es el siguiente:

1. La aplicación llama a una función de una de las librerías compartidas, para conocer que IP hay detrás de un nombre de dominio.
2. Se consulta como actuar en el fichero `/etc/nsswitch.conf`. Suele ser habitual que antes de consultar ningún servidor de DNS se consulte el fichero `/etc/hosts`. En los sistemas Linux es posible cambiar este comportamiento.
3. Una vez se llega al punto que es necesario consultar

un servidor de DNS se consulta el fichero `/etc/resolv.conf` para conocer la dirección IP del servidor a consultar.

4. Se manda una solicitud de DNS al servidor. Si el servidor conoce la IP enviará está de vuelta a la aplicación. Si no lo conoce, suele ser habitual, que el propio servidor de DNS genere una cascada de consultas a otros servidores DNS hasta que se obtiene la dirección buscada.
5. La dirección deshace el camino desde el servidor de DNS y se entrega a la aplicación.

10.4.5.2 El fichero `/etc/nsswitch.conf`

El fichero `/etc/nsswitch.conf` es un fichero de configuración usado por los sistemas Linux para conocer la procedencia de muchos servicios asociados a nombres. Así por ejemplo, una de sus funciones originales, aunque ya en desuso, es indicarle a las librerías del sistema de donde debe obtener las contraseñas asociadas a los usuarios.

El fichero `/etc/nsswitch.conf` es un fichero de texto plano en que cada línea representa una base de datos y seguido de un espacio el lugar donde esa base de datos debe obtener los valores. Así por ejemplo es habitual encontrar una línea:

```
passwd: compat
```

Esta línea activa el modo de compatibilidad para la base de datos de contraseñas. En este caso no será `nsswitch.conf` quien se haga cargo de proporcionar los valores si no que lo hará PAM, podemos revisar el capítulo 8.Usuarios para ver en que fichero

podemos encontrar las contraseñas y como funciona PAM.

El fichero `/etc/nsswitch.conf` permite la especificación de varias fuentes para una misma base de datos. En el caso de que se especifique más de una fuente, el orden en el que aparecen marca la prioridad de las fuentes especificadas. Así, en el caso que nos ocupa, el de `dns`, deberemos buscar la base de datos nombrada como `hosts`.

Esta línea indica donde debe buscar para resolver los nombres de dominio, lo habitual sería ver una línea como la siguiente:

```
hosts: files dns
```

`files` indica que debe buscar en los ficheros de configuración del sistema, más concretamente en el fichero `/etc/hosts`. La segunda fuente nos indica que debe enviar una solicitud de resolución a un servidor de DNS. El servidor de DNS en el que debe buscar se especificará en otro fichero de configuración.

Atención

Para poder poner en primer lugar la consulta al servidor de `dns` y en segundo lugar el fichero de configuración debe hacerse siguiendo la siguiente línea:

```
hosts:      dns [!UNAVAIL=return] files
```

De otra manera, si la consulta al servidor `dns` falla nunca se buscará la resolución en los ficheros de configuración.

10.4.5.3 El fichero `/etc/hosts`

El fichero `/etc/hosts` es un fichero de configuración de texto plano que permite especificar relaciones entre nombres de dominio y direcciones IP. Cada línea del fichero tiene una de estas relaciones y consta de 3 columnas, aunque la tercera es opcional. En primer lugar, se especifica la dirección IP, a continuación el nombre de dominio y finalmente, si se desea, un alias. El formato sería el siguiente:

```
127.0.0.1 localhost miPC
208.164.186.1      ip1.com      aliasIP1
```

La IP 127.0.0.1 es una IP especial que representa siempre a la propia máquina. Así pues, si nos conectamos a la IP 127.0.0.1, estemos en la máquina que estemos nos conectaremos a la propia máquina.

Después de modificar el fichero de hosts es necesario reiniciar los servicios de red para que los cambios sean tenidos en cuenta. Es recomendable antes de modificar el fichero realizar una copia de seguridad del mismo.

10.4.5.4 El fichero `/etc/resolv.conf`

El fichero `/etc/resolv.conf` es tradicionalmente el fichero que permite configurar los servidores de dns a los que solicitar la resolución de nombres de dominio. Aunque actualmente este fichero sigue existiendo, en muchas ocasiones no es manejado por el usuario si no por otros programas o procesos que se encargan de manejar los datos que hay en él. Es el caso, por ejemplo, del network-manager en aquellas distribuciones que cuentan con él. La tarea de configuración rehace entonces sobre los programas que manejan este fichero y no sobre el fichero propiamente.

Si mantuviésemos la configuración manual más tradicional, el fichero `/etc/resolv.conf` presentaría un aspecto como el siguiente:

```
search ejemplo.com
nameserver 8.8.8.8
nameserver 8.8.4.4
```

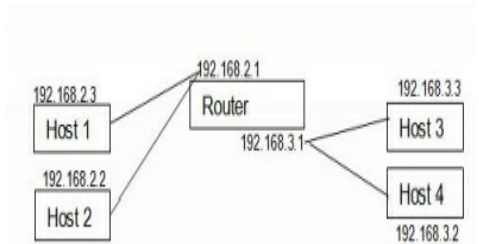
En el ejemplo, el `search` indica que en caso de intentar conectar a un nombre de dominio incompleto, por ejemplo Facebook en lugar de Facebook.com, se debe completar la cadena introducida con `ejemplo.com`. Si se diese el caso, Facebook quedaría como: Facebook.ejemplo.com.

Las líneas siguientes indican los servidores de DNS a los que hay que contactar. Estos se consultan según el orden de aparición en el fichero. Al contrario de lo que pueda parecer, si uno de ellos devuelve un valor informando que no se ha podido consultar el nombre de dominio no se consultará al siguiente. Únicamente en caso de que un servidor no este disponible se consultará al siguiente.

Si al abrir el fichero `/etc/resolv.conf` nos encontramos una dirección 127.0.0.1, significa que el archivo está gestionado por otro programa. En estos casos es muy recomendable no modificar el fichero manualmente.

10.5 Rutas y tablas de rutas

Una vez conocemos las subredes y las direcciones IP que identifican a las máquinas de estas subredes, necesitamos mecanismos que nos permitan conectar las subredes entre ellas. Este es el trabajo de los enrutadores, o como son más conocidos de los routers.



Un router no es más que una máquina que se encuentra en más de una red. Es decir, dispone de más de una IP. Esto permite que, llegado el momento, pueda recibir mensajes de una subred y enviarlos a otra. Al enviar un mensaje a través de la red nuestra máquina decide donde enviarla según una tabla de enrutamiento. En la tabla de enrutamiento se especifica que hacer con los mensajes según el origen del que provengan. Las máquinas que se encuentran en la misma red se pueden alcanzar directamente, es lo que se conoce como entrega directa. Para entregar los mensajes a máquinas que se encuentran en otra red se debe recurrir a un router. Un router tiene “conciencia” de las redes a las que pertenece y de las máquinas que puede encontrar en dichas redes. También tiene conocimientos sobre otros routers a los que debe enviar los mensajes en caso de no poder entregarlos en ninguna de las redes a las que está conectado.

Los sistemas Linux actúen o no como enrutadores, disponen

de una tabla de enrutamiento que les permite decidir por que interfaz deben entregar los mensajes. Si deben realizar entrega directa o si por el contrario deben entregar el mensaje a un router. Incluso pueden enviar los mensajes que van a una determinada dirección IP con la dirección de origen que se especifique. Para ver la tabla de enrutamiento de nuestro sistema podemos emplear

route -n

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
0.0.0.0	192.168.1.10	0.0.0.0	UG	0	0	0	eth0

ip r s # ip route show

192.168.99.0/24 dev eth0 scope link

127.0.0.0/8 dev lo scope link

default via 192.168.99.254 dev eth0

Aunque la salida de ambos comandos es bastante distinta, en esencia nos muestran la misma información. Fijémonos primero en la salida del comando route -n.

Vemos que la primera columna nos indica la destinación. En la primera línea una IP que termina en 0. Si recordamos el apartado de subredes, hemos comentado que una la parte que indica el host de una Ip no podía ser todo 0s o todo 1s. Estas direcciones están reservadas. La dirección con todo 0s en la parte que no representa la subred, se emplea junto con la mascara de red, la tercera columna, para indicar que la regla afecta a la subred al completo. La segunda columna nos indica el gateway o puerta de enlace, es decir, a donde deben enviarse los mensajes que lleguen a la máquina, o sean

generados por ella, y que vayan destinados a la subred 192.168.1.0/24. En el caso del comando *route -n* un 0.0.0.0 nos indica que deben entregarse mediante entrega directa.

Juntando Conceptos

Si los mensajes deben entregarse mediante entrega directa, podemos deducir que la máquina se encuentra en la subred 192.168.1.0/24, al menos en la interfaz en la que esta regla está especificada.

La última columna de la salida de *route -n* nos indica la interfaz de red por la que deben ser entregados los mensajes. La elección de la interfaz puede ser tanto por que la máquina tenga interfaz en redes distintas como por que se quiera cambiar el medio. Es decir, enviar unos mensajes de forma inalámbrica y otros a través de la interfaz con cable.

Finalmente nos quedan 3 columnas, estas nos indican diversos parámetros sobre la ruta:

- **Flags:** Indican propiedades sobre la regla de enrutamiento definida, puede tomar, entre otros, los valores siguientes:
 - **U:** La ruta está activa.
 - **H:** La destinación es una sola máquina.
 - **G:** La ruta implica un router.
 - **!:** Descarta todos los mensajes que cumplen las condiciones de la regla.

- **Metric:** La distancia hasta la destinación. La distancia se cuenta en routers que atraviesa el mensaje. Así pues, si definimos una regla con una métrica de 3 y debemos atravesar más de 3 subredes para poder alcanzar el host de la destinación el mensaje no llegará nunca. Actualmente el parámetro de las métricas está en desuso y de hecho en el nuevo comando `ip route` ya no se emplea por defecto. Esta métrica sigue empleándose en otros protocolos, pero asociada a cada mensaje concreto.
- **Ref:** Un parámetro en desuso. Se mantiene únicamente por compatibilidad con los kernels antiguos.

Si nos fijamos ahora en la salida de `ip route`, vemos que es algo distinta. Sin embargo, la información es la misma. Igual que en el caso del comando `route`, cada línea se corresponde con una regla de enrutamiento.

```
192.168.99.0/24 dev eth0 scope link
```

En primer lugar, nos indica IP y máscara, en notación CIDR, a la que se aplicará la regla. Es decir, todos los mensajes que tengan como destinación la IP y la máscara especificada cumplirán con la regla. A continuación nos indica el dispositivo, es decir “dev eth0”. Con esto conocemos que interfaz usar para mandar los mensajes cuando se cumpla la regla. Finalmente nos indica el ámbito del envío.

Encontramos dos opciones: la del ejemplo, “scope link” que nos indica que se trata de entrega directa, por tanto que la destinación se encuentra en la misma red, o bien “via 192.168.99.254” que indica que se debe enviar a través de un router y la IP de dicho router.

10.5.1 Ruta por defecto

En las tablas de enrutamiento encontramos rutas por defecto. En el caso de route -n era una regla que tenía como ip de destinación la dirección IP 0.0.0.0 con la máscara 0.0.0.0 y en el caso de ip route se trata de la regla que empieza por la palabra clave default. Las reglas en las tablas de enrutamiento se evalúan por orden. Es decir, las reglas se evalúan de las que aparecen más arriba a las que aparecen más abajo. Si una IP coincidiese con más de una regla, únicamente se aplicaría la que se encontrase en primer lugar.

La ruta por defecto es una regla que coincide con cualquier IP y debe ser siempre la última regla de nuestra tabla de enrutamiento. En caso de que no fuese la última, dado que todas las posibles IPs caen dentro de esta regla, las que encontrásemos detrás no serían evaluadas. Normalmente la ruta por defecto tiene como destino un router que es el que dará la conexión a redes que no son conocidas por la máquina.

Supongamos que queremos llegar a google.com. Es evidente que la máquina con la página web del famoso buscador no está en nuestra oficina, a menos que trabajemos en Google, en cuyo caso existirá una oportunidad. Tampoco sabemos la IP de la máquina, no podemos pretender que una web tan importante tenga únicamente una dirección IP, seguramente tendrá varias. Además, debemos tener en cuenta que no solo existe la web de google.com, existen muchas más a

las que queremos acceder. Así pues, después de configurar las rutas para todas las IPs que nos son conocidas, le indicaremos a nuestra máquina que envíe el resto de mensajes a un router, que permite conectar diversas subredes, con la esperanza de que el router conozca la subred donde se encuentra la máquina de Google. En caso contrario confiaremos en que ese router dirija el tráfico hacia otro router que si que pueda conocerla.

Esta es una simplificación muy generosa del funcionamiento de los routers y del encaminamiento. La realidad es bastante más compleja, sin embargo el principio subyacente detrás de las comunicaciones de red es esta idea. La complejidad aparece por la necesidad de conocer otra información y de evitar algunos de los problemas que genera la idea presentada.

Para añadir una ruta por defecto, o una ruta cualquiera, podemos recurrir a los mismos comandos que nos muestran la información al respecto.

#Para una ruta cualquiera

```
route add -net 192.56.76.0 netmask 255.255.255.0 dev eth0
```

#Para la ruta por defecto

```
route add default gw <IP del router>
```

#Para una ruta cualquiera

```
ip r a 192.168.1.0/24 dev eth0 scope link
```

#Para una ruta por defecto

```
ip r a default via <IP del router> dev eth0
```

De la misma manera, es posible eliminar las rutas que

hayamos definido mediante estos comandos:

```
route del -net 192.56.76.0 netmask 255.255.255.0  
ip r d 192.168.1.0/24
```

Trabajar con rutas y con las tablas de rutas puede llegar a ser bastante complejo, por lo que a menos que sea realmente necesario, se recomienda seguir el principio de “keep it clean, keep it easy”: Manténlo limpio, mantelo fácil. Mientras menos rutas tengamos en la tabla de rutas y más “inteligencia” podamos delegar a los routers más sencilla será la tarea de administrar la red en las máquinas. Hay que tener en cuenta que a día de hoy muchas de estas configuraciones son gestionadas por gestores de red. La nueva familia de comandos ip, incluso añade una ruta de entrega directa al añadir una ip a una interfaz. Así pues, al ejecutar

```
ip a a 192.168.1.10/24 dev eth0  
#Tambien se está ejecutando  
ip r a 192.168.1.0/24 scope link dev eth0
```

10.6 Configurar un sistema Linux como router

Un router no es más que una máquina capaz de conectar diversas interfaces físicas de red. Aunque a grandes rasgos esto es verdad, necesita cumplir alguna característica más. Por ejemplo, debe dejar pasar el tráfico de red a través de él. Esto quiere decir que un router debe aceptar tráfico que vaya destinado a una IP distinta de la que tiene y debe ser capaz de dirigirlo a alguna de sus subredes según conveniencia y según la tabla de enrutamiento.

Una tabla de enrutamiento de una máquina que actúe como

router se vería de forma similar a la siguiente:

```
ip r s # ip route show
192.168.99.0/24 dev eth0 scope link
192.168.100.0/24 dev eth1 scope link
127.0.0.0/8 dev lo scope link
default via 192.168.99.254 dev eth0
```

Con esta tabla de enrutamiento los paquetes que lleguen a la máquina con destino a 192.168.99.0/24 serán enviados por una interfaz del router en entrega directa, mientras que los que vayan a 192.168.100.0/24 serán enviados a través de la otra interfaz, también mediante entrega directa. Es decir, este router conocería dos subredes y el tráfico que no conociese lo enviaría a otro router dentro de una de las subredes. Sin embargo, como hemos dicho se debe permitir dirigir el tráfico de una interfaz a otra, que es algo que los sistemas Linux no permiten por defecto. Tenemos, no obstante la opción de habilitar este comportamiento. Para ello debemos ejecutar:

```
sysctl -w net.ipv4.ip_forward
```

Con el comando que se muestra más arriba el cambio no será permanente. Si queremos que el cambio realizado sea permanente deberemos añadir el comando en el fichero /etc/sysctl.conf. Es posible que en algunas distribuciones se siga el esquema que ya hemos visto en otras ocasiones de disponer de diversos ficheros .conf dentro del directorio /etc/sysctl.d/, en este caso simplemente será necesario crear el fichero para habilitar esta característica.

Si la distribución con la que trabajamos no nos pone las cosas

fáciles, una búsqueda con el concepto clave “ip forwarding” nos dirá que hacer para la distribución con la que trabajamos.

10.7 DHCP

Actualmente las IPs fijas y la configuración de servidores de resolución de nombres de dominio o de las tablas de rutas no suele hacerse manualmente. Las redes han crecido mucho y en muchas ocasiones, las IPs que son accesibles para todo el mundo, las IPs públicas esconden detrás redes privadas con sus propias IPs.

Para realizar estas configuraciones de forma automática el protocolo DHCP (Dynamic Host Configuration Protocol) permite “alquilar” una dirección IP, y obtener la ruta por defecto y los servidores de DNS de un servidor. El servidor de DHCP debe existir en la propia red local, puesto que en el momento de recurrir a él, todavía no disponemos de dirección IP. Esto hace que en la mayoría de ocasiones, sea el propio router de la red local el que ofrezca también el servicio de DHCP.

DHCP funciona como un cliente que se conecta a un servidor. En los sistemas Linux este cliente es un programa conocido como `dhclient`. Al iniciar el sistema se lanza el servicio de `dhcp` para obtener la configuración de red. Si así se ha especificado, una vez obtenida la configuración del servidor de DHCP, suele almacenarse en `/var/state/dhclient.leases`. Vamos a ver, a grandes rasgos como funciona DHCP.

1. La máquina que quiere obtener una IP y la correspondiente configuración (servidores DNS, ruta por defecto,...) manda a toda la red local una petición de DHCP.

2. Aquellos en la red local que entienden la petición, es decir, aquellos sistemas que cuentan con un servidor de DHCP, usualmente los routers, contestan con una oferta de DHCP.
3. La máquina que envió la petición escoge una oferta de entre todas las que le hayan llegado y envía un mensaje al origen de esa oferta con una solicitud de DHCP.
4. El servidor le asigna la oferta que le había hecho durante un tiempo limitado. Normalmente 48 o 72 horas. Pasado ese tiempo, la asignación de DHCP caducará y será responsabilidad de la máquina solicitante negociar de nuevo una configuración DHCP.

La duración de las asignaciones de DHCP corren a cuenta del servidor, por lo que, a menos que administremos un servidor de DHCP, no está en nuestras manos cambiar el tiempo de dicha asignación. Lo que si que nos permite un sistema Linux es solicitar a través de un comando una IP y configuración mediante DHCP. También, de la misma manera nos permite renunciar a una oferta que ya hayamos aceptado y que se encuentre en uso. Para ambas tareas recurriremos al comando `dhclient`.

`dhclient -r eth0` # Libera la IP actual para la interfaz `eth0`, así como la configuración

`dhclient eth0` #Realiza una nueva petición DHCP para la interfaz `eth0`

Antes de trabajar con estos comandos debemos recordar que para cualquier configuración de red que queramos realizar deberemos

tener permisos de administración.

10.8 La capa de transporte

La capa de internet y las direcciones IP permiten la comunicación entre máquinas. Pero, una vez llega el mensaje a la máquina debemos saber que hacer con él. Para identificar a quién se le debe entregar el mensaje, a que aplicación, se emplean los puertos. Los puertos son un número entre 0 y 65535 que permiten identificar a que aplicación deben entregarse los mensajes que llegan a la máquina. Las parejas IP, puerto suelen estar asociadas a un socket Berkley. Los sockets son la interfaz que ofrece el kernel para interactuar con los dispositivos de red. Los programas trabajan con los sockets a partir de un descriptor de fichero que obtienen en el momento de la creación del socket.

Los dos protocolos a nivel de aplicación más extendidos y usados son:

- **TCP:** Es un protocolo de la capa de transporte que trabaja con conexiones extremo a extremo. TCP garantiza la integridad de los datos extremo a extremo, también garantiza que el orden de llegada de los mensajes sea el mismo que el de salida. Además TCP implementa mecanismos de control de la congestión en la red.
- **UDP:** Es también, un protocolo de la capa de transporte que permite mandar mensajes extremo a extremo sin necesidad de establecer una conexión. A diferencia de TCP

no implementa mecanismos de control de la congestión, no asegura que lleguen todos los mensajes ni el orden de llegada de estos en caso que sea necesario fragmentarlos. Parece que todo son desventajas para UDP, sin embargo su uso está muy extendido en aplicaciones de streaming o de telefonía IP como Skype donde perder un paquete no es crítico pero la velocidad en la conexión si que lo es.

10.8.1 TCP

Aunque parezca una sutileza debemos notar, que es la primera vez, al hablar de TCP, en la que se habla de conexión. Esto se debe a que las conexiones en la capa de internet son máquina a máquina no extremo a extremo. Únicamente en la capa de transporte, el protocolo TCP establece una conexión punto a punto para la transmisión de datos. Una conexión de dos máquinas se define como el conjunto:

$(IP_{origen}, Puerto_{origen}), (IP_{destino}, Puerto_{destino})$

Si no se conocen estos datos la conexión no está establecida y no es hasta la capa de transporte que los puertos no entran en juego. Es por tanto, trabajo de TCP proveer los mecanismos para establecer una conexión y mantenerla abierta el tiempo necesario para enviar los datos. El protocolo TCP es bastante complejo, si se quiere conocer en profundidad, lo mejor es recurrir al RFC donde se define el protocolo.

Para ver las conexiones de nuestro sistema podemos recurrir al comando `netstat`. Este comando nos ofrece información sobre todos los puertos en nuestro sistema y las aplicaciones relacionadas con estos puertos. Se debe ser cuidadoso puesto que la información que mostrará `netstat` varía si el comando se ejecuta con permisos de administración o no. Para ver los puertos numéricamente y limitar el resultado únicamente a las conexiones TCP usaremos la opción `-t`, si se especifica también una `u` nos mostrará además, información sobre los puertos `udp` en la máquina.

```
netstat -nt
```

```
Active Internet connections (w/o servers)
```

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	127.0.0.1:34105	127.0.0.1:389	ESTABLISHED
tcp	0	0	127.0.0.1:389	127.0.0.1:54529	ESTABLISHED
tcp	0	0	127.0.0.1:389	127.0.0.1:56327	ESTABLISHED
tcp	0	0	172.16.4.5:110	172.25.47.30:1680	TIME_WAIT
tcp	0	0	172.16.4.5:110	172.25.131.212:2313	TIME_WAIT

- **Proto:** Muestra el protocolo de la capa de transporte. Si hemos especificado la opción `-t` este siempre será TCP.
- **Recv-Q Send-Q:** Indican el número de bytes que han sido enviados o recibidos por el programa, sin embargo una vez los mensajes ya se han entregado al nivel de aplicación para los recibidos o al socket para los

enviados, dejan de reflejarse en netstat, así que lo más probable es ver un 0 en estos valores.

- **Estado:** El estado en que se encuentra el socket. Dado que los estados se asocian normalmente con las conexiones, con el protocolo TCP, para otros protocolos es posible que encontremos el estado vacío. Puede tomar, entre otros, algunos de los siguientes valores:
 - **ESTABLISHED:** El socket mantiene una conexión activa y establecida.
 - **TIME_WAIT:** El socket está esperando a la otra parte para cerrarse
 - **LISTEN:** El socket está escuchando conexiones entrantes. Los sockets en este estado no son listados por netstat a menos que se especifique la opción -l o bien la opción -a.
 - **UNKNOWN:** El estado del socket es desconocido.
- **Local Address:** Especifica la dirección IP y el puerto local, normalmente separados por el carácter ':'
- **Foreign Address:** Especifica la dirección IP y

el puerto de la otra máquina. En las conexiones con estado LISTENING es habitual encontrar cosas como *:80 o *:.* lo que significa que espera una conexión de cualquier dirección IP por el puerto 80 o bien que espera una conexión de cualquier dirección IP por cualquier puerto. La verdad es que el último caso suele ser poco corriente, pero si lo vemos no debemos asustarnos.

10.8.2 Puertos

Los puertos son valores enteros que representan las distintas aplicaciones. El número que puede tomar un puerto va desde 1 hasta 65535. Los primeros 1024 puertos son conocidos como well known ports y los usuarios sin permisos no pueden cambiar la utilidad de estos puertos. Tampoco los programas que no se ejecutan como súper usuario pueden hacer uso de estos puertos. Son puertos asociados a servicios o protocolos bien conocidos. Es el caso de http, el protocolo de hipertexto que emplean las páginas web. Este se encuentra en el puerto 80 por defecto. Los puertos por encima del 1024 se conocen como puertos efímeros y están a disposición de los usuarios.

Habitualmente los servidores suelen escuchar por un puerto a la espera de conexiones. Una vez han recibido la conexión, para no ocupar el puerto conocido, derivan la conexión a un puerto efímero y siguen a la espera de conexiones por el puerto conocido. Así trabajan por ejemplo los servidores FTP. FTP suele escuchar por el puerto 20 pero, para poder soportar conexiones concurrentes, una vez recibe

una petición de conexión la deriva a un puerto efímero y sigue atento a nuevas conexiones por el puerto 20.

Juntando conceptos

¿Podemos, en el caso de TCP, tener dos conexiones simultáneas al puerto 65400? La respuesta en este caso sería sí, siempre que vengan de direcciones IP distintas y que nuestra máquina conteste desde un puerto efímero.

Hemos quedado que una conexión se identifica por la pareja:

$(IP_{\text{origen}}, Puerto_{\text{origen}}), (IP_{\text{destino}}, Puerto_{\text{destino}})$

mientras que un socket está asociado a

$(IP, Puerto)$

Por tanto siempre que nuestra máquina mantenga un socket efímero podemos tener tantas conexiones al puerto 65400 como podamos soportar.

Aunque conocer los puertos bien conocidos o los well-known ports no es sencillo sin recurrir a fuentes externas, recordar 1024 protocolos y sus números es toda una proeza, una buena manera de tener una idea de cuales son en nuestro Linux es recurrir al fichero `/etc/services`.

Este fichero no es más que un fichero que traduce los números de los puertos bien conocidos a nombres. Podemos editar este fichero para que aquellos puertos que nosotros queramos se

muestren con el nombre del protocolo al que pertenecen en lugar de con el número, por ejemplo al ejecutar netstat.

El fichero /etc/services es un fichero de texto plano donde cada línea equivale a un puerto y su descripción.

```
smtp 25/tcp mail
```

El fichero /etc/services no afecta en nada a los puertos habilitados o no. Simplemente provee una manera de traducir nombres de servicio a puerto y a la inversa. Así pues un programa puede emplear un nombre que aparezca en /etc/services para el puerto pero el borrar una entrada de /etc/services no deshabilitará el servicio que se borra.

10.9 Ethernet inalámbrico

En principio las conexiones inalámbricas no son muy diferentes de las conexiones cableadas. El término más extendido para este tipo de conexiones es el de conexiones Wi-Fi. Este nombre proviene de una marca comercial, por tanto nosotros usaremos el término wireless o inalámbrico.

Igual que sucedía en el caso de las conexiones cableadas, de las que hemos hablado hasta ahora, las conexiones inalámbricas tienen cada una su interfaz. Los conceptos de enrutamiento, puertos y protocolos son exactamente los mismos. Lo único que cambia son los protocolos a bajo nivel y no entraremos en detalles sobre ellos. Cada elemento de hardware que nos permite conectarse en las conexiones inalámbricas tiene su propia MAC y las interfaces inalámbricas disponen también de su propia dirección IP. Aún así encontramos algunas diferencias, sobre todo debido a que la configuración de las

redes inalámbricas requiere muchas más intervención del usuario.

Si pensamos en las redes cableadas no tenemos muchas opciones, conectamos el cable y allí donde el cable está conectado estará conectado nuestro sistema. En este sentido escoger la red pasa por conectar el cable a una u otra red y darle al sistema la IP apropiada. No sucede lo mismo con los sistemas inalámbricos. En el mejor de los casos nuestro adaptador detectará varias redes y solo algunas de ellas serán conocidas. Así pues, será el usuario el que deberá encargarse de seleccionar la red.

Algunas de las características añadidas a las conexiones inalámbricas de las que no disponen las conexiones físicas son:

- **SSID:** La identificación de la red o el nombre que se le da a la red. En el caso de las redes cableadas no es necesario que las redes tengan un nombre, basta con conocer la subred. En el caso de las redes inalámbricas es distinto. En cada vecindario es posible encontrar más de 10 redes inalámbricas. Como diferenciarlas por números sería complejo, se emplea un identificador.
- **Gestión:** Muchas redes inalámbricas están conectadas a puntos de acceso. Los puntos de acceso no son más que el puente que permite a una red inalámbrica conectarse con una red cableada. Aquellos dispositivos que se encuentran conectados a un punto de acceso, comparten dirección de subred con

los dispositivos conectados a través de cable.

- **Detalles de transmisión:** Aunque estos detalles suelen ir asociados a cada conexión, seguro que a más de uno le suena el concepto de canal, intensidad de la señal, o el concepto de ruido. Todos estos detalles son más o menos configurables cuando uno es el dueño de la conexión. También puede que los debamos considerar cuando tratamos de conectarnos.
- **Autenticación y cifrado:** Dado que el aire es patrimonio de todos y es el medio por el que se transmiten las redes inalámbricas se plantea la problemática de proteger la red de invitados indeseados. Aparecen entonces los conceptos de autenticación y cifrado de los datos para protegerlos de aquellos que no deben entrar en la red.

10.9.1 La herramienta iw

La herramienta iw es la que permite listar y manipular dispositivos inalámbricos. Es una herramienta relativamente nueva y es posible que en su lugar encontremos wireless-tools, una herramienta más antigua que provee de una gran variedad de herramientas, cada una para una de las tareas que ahora realiza iw. Es posible que dependiendo de la distribución iw no se encuentre instalado en el sistema. En este caso deberemos instalarlo. Puede parecer paradójico el hecho de necesitar acceso a Internet para poder descargar un

programa que permita configurar la conexión a una red. Quizás lo sea, pero lo mismo es muy posible que nos suceda con los drivers de la tarjeta de red inalámbrica que tengamos. En estos casos se recomienda descargar el software necesario desde otra máquina o proveer a la máquina de una conexión cableada en primer lugar. Aunque no sucederá en todas las distribuciones, no habrá manera de saberlo hasta que no nos encontremos de frente con el problema.

`iw` es un comando que recibe como parámetro el nombre de una interfaz. Precedido o no por los modificadores “`dev`” para interfaces lógicas (`wlan0`) o “`phy`” para interfaces físicas (el dispositivo hardware). A continuación recibe el comando que se debe ejecutar sobre la interfaz. Cada comando tiene sus propios parámetros y opciones por lo que es difícil hablar de todos ellos. Vamos a ver algunos comandos que pueden resultar muy interesantes.

- **link:** Provee información sobre el link. Es decir sobre si el dispositivo está listo para enviar y recibir datos o si por el contrario no lo está. En caso de que nos encontremos conectados a una red inalámbrica obtendremos diversa información sobre la red a la que nos hemos conectado.
- **scan:** Escanea las posibles redes y puntos de acceso. En una interfaz gráfica, el desplegar las redes inalámbricas disponibles equivaldría al comando `scan`.
- **connect:** Este comando puede recibir bastantes parámetros en función de lo que se

deseo hacer. Su funcionalidad permite conectar una interfaz a la red seleccionada.

Vamos a ver varios ejemplos que nos ayudarán a entender mejor como funcionan estos comandos junto con iw.

#Lista las redes disponibles así como sus características.

#Debemos buscar el campo SSID marcado en gris

iw dev wlan0 scan

BSS 1a:2b:3c:4d:5e:6f (on wlan0)

TSF: 5311608514951 usec (61d, 11:26:48)

freq: 2462

beacon interval: 100

capability: ESS Privacy ShortSlotTime (0x0411)

signal: -53.00 dBm

last seen: 104 ms ago

Information elements from Probe Response frame:

SSID: miRed

Supported rates: 1.0* 2.0* 5.5* 11.0* 18.0 24.0 36.0 54.0

DS Parameter set: channel 11

ERP: Barker_Preamble_Mode

RSN: * Version: 1

* Group cipher: CCMP

* Pairwise ciphers: CCMP

* Authentication suites: PSK

* Capabilities: 16-PTKSA-RC (0x000c)

Extended supported rates: 6.0 9.0 12.0 48.0

[...]

```
#Conectamos a la red miRed
```

```
iw wlan0 connect miRed
```

```
#Miramos el estado del link
```

```
iw dev wlan0 link
```

```
Connected to 1a:2b:3c:4d:5e:6f (on wlan0)
```

```
SSID: miRed
```

```
freq: 2412
```

```
RX: 26951 bytes (265 packets)
```

```
TX: 1400 bytes (14 packets)
```

```
signal: -51 dBm
```

```
tx bitrate: 6.5 MBit/s MCS 0
```

```
bss flags: short-slot-time
```

```
dtimperiod: 0
```

```
beacon int: 100
```

Si nuestra red utiliza además mecanismos de cifrado necesitaremos además de `iw wpa-suplicant`. Adía de hoy la situación más normal es que se requiera de este software. `wpa-suplicant` provee de un fichero de configuración, normalmente almacenado en `/etc/wpa_supplicant.conf`. El fichero `wpa_supplicant.conf` además de permitir añadir seguridad a la red permite la conexión a una red de forma automática sin tener que proveer los datos cada vez.

El fichero `/etc/wpa_supplicant` es un fichero de texto plano del estilo:

```
network={
```

```
ssid="casa"
scan_ssid=1
key_mgmt=WPA-PSK
psk="frase super secreta"
}

network={
    ssid="trabajo"
    scan_ssid=1
    key_mgmt=WPA-EAP
    pairwise=CCMP TKIP
    group=CCMP TKIP
    eap=TLS
    identity="usuario@ejemplo.com"
    ca_cert="/etc/cert/ca.pem"
    client_cert="/etc/cert/user.pem"
    private_key="/etc/cert/user.prv"
    private_key_passwd="contraseña"
}
```

Cada entrada "network", especifica los detalles para una red. Las redes son leídas en orden de aparición. Eso significa, que si la conexión es posible con la primera de las redes que aparece en el fichero, el sistema no probará de conectarse a ninguna otra. La modificación del fichero `wpa_supplicant.conf` requiere del reinicio de los servicios de red para tener efecto. Para conocer todos los detalles que pueden ser especificados en una red, lo mejor es recurrir a la

entrada del manual de wpa_supplicant, dado que es muy detallada y estará completamente actualizada según la versión y distribución de Linux de la que dispongamos.

man wpa_supplicant.conf

Advertencia

Se debe ser cuidadoso con los ficheros wpa_supplicant ya que las contraseñas y frases de protección de certificados aparecen escritas en texto claro. Es decir, sin cifrar ni ocultar. Toda persona que tenga acceso al fichero tendrá acceso también a las contraseñas de la red.

En las secciones de red han aparecido diversos campos y valores que no han sido explicados. Antes de entrar en profundidad en estos valores y en el significado de los campos es recomendable profundizar en distintos conceptos de red. Empezando por profundizar en el modelo de capas, los protocolos que intervienen, el modelo de enrutamiento y los métodos de acceso a la capa física. El libro Comunicaciones y Redes de Computadores de William Stallings es una muy buena opción para iniciarse en el campo de las redes de computadores.

Epílogo

Poner punto final a un texto que trate sobre Linux parece realmente complicado. Hay una gran variedad de comandos, opciones, programas y distribuciones. Aún más, hay una comunidad detrás de cada distribución Linux que hace crecer y mejorar a los distintos sistemas. Si a todo este le sumamos las capacidades de red, tan necesarias hoy en día, resulta todavía más difícil saber donde se debe poner el punto y final.

A lo largo de este texto se han revisado algunos de los pilares de los sistemas Linux desde una perspectiva que, espero, ayude a entender que sucede cuando trabajamos con estos sistemas y ayude a ir un poco más allá que el saber escribir un comando en el terminal.

Muchas cosas han quedado en el tintero y otras se han explicado con palabras que quizás no sean las más adecuadas. Simplemente se ha tratado de poner el esfuerzo en que el texto pudiese llegar a la mayor cantidad de gente posible.

La información sobre los sistemas operativos Linux es muy abundante en Internet y uno no debe tener miedo de buscarla i enfrentarse a ella. Con unos mínimos conocimientos de inglés y una idea de lo que se desea hacer será sencillo encontrar información al respecto. Seguro que la información disponible y que ofrece la comunidad será suficiente para solventar las dudas que haya podido dejar este texto.

Bibliografía

Antes de citar una lista de referencias, decir que mucha de la información ha sido extraída de las páginas de manuales de los distintos comandos o de las páginas de referencia de la Linux Foundation o Freedesktop. ServerFault también llega a convertirse en una página muy útil para todos aquellos que deben administrar un sistema.

Carretero J, García F, Anasagasti P, Pérez F, Sistemas Operativos: una visión aplicada, Mc Graw Hill, 2007

Evi N, Unix and Linux System Administration Handbook, Pearson Education, 2011.

Garfinkel S, Spafford G, Schwartz A, Practical Unix & Internet Security, O'Reilly 2003

Kroha-Hartman G, Linux Kernel in a Nutshell, O'Reilly 2006

Nguyen B, Linux Filesystem Hierarchy, Version 0.65, 2015,
<http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/>

Poettering L, systemd System and Service Manager, 2015,
<http://www.freedesktop.org/wiki/Software/systemd/>

Prieto A, Lloris A, Torres JC, Introducción a la informática,

McGrawHill, 2006

Samar V, Schemers R, Unified login with pluggable authentication modules (PAM), Open Software Fundation, RFC 86, 1995. <http://www.opengroup.org/rfc/rfc86.0.html>

Shotts W, The Linux command Line: a Complete Introduction, no starch press, 2012

Stallings W, Comunicaciones y redes de computadores, Pearson, 2004.

Suárez R, Descubriendo Udev, Todo Linux: la revista mensual para entusiastas de GNU/LINUX, (79), 18-22. 2007