Administración local

Josep Jorba Esteve

PID_00238585

Universitat Oberta de Catalunya

Índice

Introducción..... 5

1. Herramientas básicas para el administrador..... 7
1.1. Herramientas gráficas y líneas de comandos 8
1.2. Documentos de estándares 10
1.3. Documentación del sistema en línea 13
1.4. Herramientas de gestión de paquetes 15
1.4.1. Paquetes TGZ 16
1.4.2. Fedora/Red Hat: paquetes RPM 19
1.4.3. Debian: paquetes DEB 24
1.4.4. Nuevos formatos de empaquetado: Snap y Flatpak 28
1.5. Herramientas genéricas de administración 37
1.6. Otras herramientas 38

2. Distribuciones: particularidades..... 39

3. Niveles de arranque y servicios..... 42
3.1. Arranque SysVinit 42
3.2. Upstart 45
3.3. Systemd 48

4. Observar el estado del sistema..... 54
4.1. Arranque del sistema 54
4.2. Kernel: directorio */proc*..... 55
4.3. *Kernel*: */sys*..... 57
4.4. Udev: gestión de dispositivos /dev 57
4.5. Procesos 62
4.6. Logs del sistema 63
4.7. Memoria 66
4.8. Discos y *filesystems*..... 67
4.9. Upower, Udisks 72

5. Sistema de ficheros..... 80
5.1. Puntos de montaje 82
5.2. Permisos 85
5.3. Sistemas de ficheros: Xfs, Zfs y Btrfs 86

6. Usuarios y grupos..... 92

7. Servidores de impresión..... 97
7.1. BSD LPD 101

7.2. CUPS 102

8. Discos y gestión *filesystems* 104
8.1. RAID software 106
8.2. Volúmenes lógicos (LVM) 117

9. Software: actualización 128

10. Trabajos no interactivos 130

11. Taller: prácticas combinadas de los apartados 132

Actividades 141

Bibliografía 142

Introducción

Una de las primeras tareas con la que tendrá que enfrentarse el administrador será la gestión de los recursos locales presentes en el sistema a administrar. En el presente módulo veremos algunas de estas tareas de administración básicas, y algunos de los aspectos de personalización y rendimiento de los recursos.

Antes de comenzar con los aspectos más prácticos de la administración, revisaremos algunas de las herramientas básicas de que dispondrá el administrador (algunas como los *shell scripts* ya las hemos revisado previamente).

Posteriormente, analizaremos el proceso de arranque de un sistema GNU/Linux, que nos hará comprender la estructura inicial del sistema y su relación con los servicios que este proporciona.

A continuación, aprenderemos cómo obtener una visión general del estado actual del sistema por medio de los diferentes procedimientos y comandos de que disponemos para evaluar las partes del sistema. De este modo, podremos tomar decisiones de administración si detectamos algún fallo o deficiencia de rendimiento, o la falta de algún recurso.

Nota

La administración local engloba muchas tareas variadas, que quizás sean las más utilizadas por el administrador en su trabajo diario.

Uno de los principales puntos de la administración es la gestión de usuarios, ya que cualquier configuración de la máquina estará destinada a que pueda ser utilizada por estos. Veremos cómo definir nuevos usuarios en el sistema y controlar su nivel de acceso a los recursos.

En cuanto a los periféricos del sistema, como discos e impresoras, disponemos de diferentes posibilidades de gestión, ya sea vía diferentes servidores (caso impresión) o diferentes sistemas de archivos que podemos tratar, así como algunas técnicas de optimización del rendimiento de los discos.

También examinaremos el problema de la actualización del sistema, así como la nueva incorporación de software de aplicación y cómo hacerlo disponible a los usuarios. Asimismo, analizaremos la problemática de ejecutar trabajos temporizados en el sistema.

En el taller final examinaremos la evaluación de estado de una máquina, siguiendo los puntos vistos en este módulo, y llevaremos a cabo algunas de las tareas de administración básicas descritas. En el desarrollo de la unidad comentaremos algunos comandos y, posteriormente, en el taller, veremos algunos de ellos con más detalle en lo que respecta a su funcionamiento y opciones.

1. Herramientas básicas para el administrador

El administrador de sistemas GNU/Linux tiene que enfrentarse, diariamente, a una gran cantidad de tareas. En general, en la filosofía UNIX no suele haber una única herramienta para cada tarea o una sola manera de hacer las cosas. Lo común es que los sistemas UNIX proporcionen una gran cantidad de herramientas más o menos simples para afrontar las diferentes tareas.

Será la combinación de las herramientas básicas, cada una con una tarea muy definida, la que nos dará la posibilidad de solucionar un problema o tarea de administración.

En este apartado veremos diferentes grupos de herramientas, identificaremos algunas de sus funciones básicas y veremos varios ejemplos de sus usos. Comenzaremos por examinar algunos estándares del mundo GNU/Linux, que nos permitirán hallar algunas de las características básicas que esperamos de cualquier distribución de GNU/Linux. Estos estándares, como el LSB (o Linux Standard Base) [Linc] y el FHS (Filesystem Hierarchy Standard) [Key], nos hablan de herramientas que esperamos encontrar disponibles, de una estructura común para el sistema de ficheros, así como de distintas normas que tienen que cumplirse para que una distribución sea considerada un sistema GNU/Linux y mantenga reglas comunes para la compatibilidad entre estos estándares.

En la automatización de tareas de administración suelen utilizarse comandos agrupados en *shell scripts* (también llamados guiones de comandos), mediante lenguaje interpretados por el *shell* (intérprete de comandos) del sistema. En la programación de estos *shell scripts* se nos permite unir los comandos del sistema con estructuras de control de flujo, y así disponer de un entorno de prototipo rápido de herramientas para la automatización de tareas.

Otro esquema habitual es la utilización de herramientas de compilación y depuración de lenguajes de alto nivel (como por ejemplo C). En general, serán utilizadas por el administrador para generar nuevos desarrollos de aplicaciones o herramientas, o para incorporar al sistema aplicaciones que vengan como código fuente y tengan que adaptarse y compilarse.

También analizaremos el uso de algunas herramientas gráficas con respecto a las habituales de la línea de comandos. Estas herramientas suelen facilitar las tareas al administrador, pero su uso es limitado, ya que dependen fuertemente de la distribución de GNU/Linux, o incluso de cada versión. Aun así, hay algunas herramientas útiles que son compartidas entre distribuciones.

Por último, analizaremos un grupo de herramientas imprescindibles para mantener el sistema actualizado, las herramientas de gestión de paquetes. El software servido en la distribución GNU/Linux, o incorporado posteriormente, se suele ofrecer en unidades denominadas paquetes, que incluyen los archivos de un determinado software, más pasos necesarios para la preparación de la instalación, la configuración posterior o, si es el caso, la actualización o desinstalación de un determinado software. Y cada distribución suele aportar software de gestión para mantener las listas de paquetes instalados, o por instalar, así como el control de las versiones existentes, o posibilidades diversas de actualización por medio de diferentes fuentes origen (repositorios de paquetes).

1.1. Herramientas gráficas y líneas de comandos

Existe un gran número de herramientas, de las que examinamos una pequeña porción en este y en los siguientes módulos, que como herramientas de administración son proporcionadas por terceros de forma independiente a la distribución o por el mismo distribuidor del sistema GNU/Linux.

Estas herramientas pueden cubrir más o menos aspectos de la administración de una tarea concreta, y presentarse con múltiples interfaces diferentes: ya sean herramientas de línea de comandos con múltiples opciones y/o ficheros de configuración asociados, o herramientas textuales con algún tipo de menús elaborados, o bien herramientas gráficas con interfaces más adecuadas para el manejo de información, o asistentes que automaticen las tareas, o bien interfaces web de administración.

Funcionalidad

Las herramientas gráficas de administración no suelen ofrecer una funcionalidad completa, y es interesante conocer cuáles son los efectos de sus acciones.

Todo esto nos ofrece un gran número de posibilidades de cara a la administración, pero siempre tenemos que valorar su facilidad de uso junto con las prestaciones y los conocimientos que posea el administrador que se dedica a estas tareas.

Las tareas habituales del administrador GNU/Linux pueden pasar por trabajar con diferentes distribuciones (por ejemplo, las que comentaremos, Fedora [Fed] o Debian [Debb], o cualquier otra), o incluso por trabajar con variantes comerciales de otros UNIX. Esto conlleva que tengamos que establecer una cierta manera de trabajar que nos permita realizar las tareas de la misma forma en los diferentes sistemas.

Por esta razón, en los diferentes apartados intentaremos destacar todos aquellos aspectos más comunes, y las técnicas de administración serán realizadas en su mayor parte a bajo nivel, mediante una línea de comandos y/o con edición de ficheros de configuración asociados.

Cualquiera de las distribuciones de GNU/Linux suele aportar herramientas del tipo línea de comandos, textual o, en particular, gráficas, que complementan las anteriores y simplifican, en mayor o menor medida, la administración de las tareas [Sm]. Pero hay que tener en cuenta varias puntualizaciones:

a) Estas herramientas son una interfaz más o menos elaborada de las herramientas básicas de línea de comandos y los correspondientes ficheros de configuración.

b) Normalmente no ofrecen todas las prestaciones o configuraciones que pueden realizarse a bajo nivel.

c) Los errores pueden no gestionarse bien, o simplemente proporcionar mensajes tipo "la tarea no se ha podido realizar".

d) El uso de estas herramientas oculta, a veces completamente, el funcionamiento interno del servicio o tarea. Comprender bien el funcionamiento interno es un conocimiento básico para el administrador, y más si tiene que desarrollar tareas de corrección de errores o optimización de servicios.

e) Estas herramientas son útiles en la mejora de la producción. Una vez que el administrador tiene los conocimientos adecuados, puede gestionar con ellas de forma más eficaz las tareas rutinarias y automatizarlas.

f) O también el caso contrario, la tarea puede ser tan compleja, o necesitar tantos parámetros, o generar tantos datos, que se vuelve imposible controlarla de forma manual. En estos casos, las herramientas de alto nivel pueden ser muy útiles y volver practicables algunas tareas que de otra manera son difíciles de controlar. Por ejemplo, dentro de esta categoría entrarían las herramientas de visualización, monitorización y resumen de actividades o servicios complejos.

g) En la automatización de tareas, estas herramientas (de más alto nivel) pueden no ser las más adecuadas: pueden no haber sido pensadas para los pasos que hay que realizar, o bien hacerlo de una forma no eficaz. Un caso concreto puede ser la creación de usuarios, una herramienta visual puede ser muy atrayente, por la forma de introducir los datos, pero ¿qué sucede cuando en lugar de introducir uno o pocos usuarios queremos introducir una lista de decenas o centenares de estos? La herramienta, si no está preparada, se vuelve totalmente ineficiente.

h) Por último, los administradores suelen querer personalizar sus tareas utilizando las herramientas que consideran más cómodas y fáciles de adaptar. En este aspecto, suele ser habitual la utilización de las herramientas básicas de bajo nivel y la utilización de *shell scripts* para combinarlas de modo que formen una tarea.

Tenemos que saber valorar estas herramientas extra según el valor añadido que tengan para nuestras tareas.

Podemos dar a estas herramientas un uso casual (o cotidiano), si tenemos los conocimientos suficientes para tratar los errores que puedan producirse, o bien con el objetivo de facilitar algún proceso para el que haya sido pensada la herramienta, pero siempre controlando las tareas que implementamos y el conocimiento técnico subyacente.

1.2. Documentos de estándares

Los estándares, ya sean genéricos del mundo UNIX o particulares de GNU/Linux, nos permiten seguir unos criterios básicos, por los que nos guiamos en el momento de aprender o realizar una tarea, y que nos proporcionan información básica para comenzar nuestro trabajo.

En GNU/Linux podemos encontrarnos con estándares como el FHS (Filesystem Hierarchy Standard) [Lin04b], que nos explica qué podemos encontrarnos (o dónde buscarlo) en la estructura del sistema de ficheros de nuestro sistema. O el LSB (Linux Standard Base), que nos comenta diferentes componentes software que solemos encontrar en los sistemas [Lin11c].

En el estándar **FHS** (Filesystem Hierchachy Standard) se describe la estructura en árbol del sistema de ficheros principal (/), donde se especifica la estructura de los directorios y los principales ficheros que contendrán. Este estándar es usado en mayor o menor medida también para los UNIX comerciales, en los cuales al principio hubo muchas diferencias que hicieron que cada fabricante cambiara la estructura a su gusto. El estándar pensado en origen para GNU/Linux se hizo para normalizar esta situación y evitar cambios drásticos. Aun así, el estándar es seguido con diferentes grados; la mayoría de distribuciones siguen en un alto porcentaje el FHS, realizando cambios menores o aportando ficheros o directorios que no existían en el estándar.

El estándar FHS

El Filesystem Hierchachy Standard es una herramienta básica para el conocimiento de una distribución, que nos permite conocer la estructura y funcionalidad del sistema de archivos principal del sistema.

Ved FHS en http://www.pathname.com/fhs

También se pueden consultar las actividades del grupo FHS en:

http://www.linuxfoundation.org/collaborate/workgroups/lsb/fhs.

Un esquema básico de directorios podría ser:

- ***/bin***: utilidades de base del sistema, normalmente programas empleados por los usuarios, ya sean desde los comandos básicos del sistema (como */bin/ls*, listar directorio), pasando por los shells (*/bin/bash*), etc.

- ***/boot***: archivos estáticos relacionados con el *boot loader*, necesarios durante el arranque del sistema previos al arranque del *kernel*. El cual puede encontrarse en / (la raiz) o en este mismo directorio, por ejemplo, */boot/vmlinuz*.

FHS y LSB

Pueden encontrarse las últimas especificaciones en estado beta en:

http://www.linuxbase.org/betaspecs/fhs/

http://www.linuxbase.org/betaspecs/lsb/

- ***/dev***: aquí encontramos ficheros especiales que representan los dispositivos posibles en el sistema. El acceso a los periféricos en sistemas UNIX se hace como si fueran ficheros. Podemos encontrar ficheros como */dev/console, /dev/modem, /dev/mouse, /dev/cdrom, /dev/floppy*... que suelen ser enlaces a dispositivos más específicos del tipo de controlador o interfaz que utilizan los dispositivos, como */dev/mouse*, enlazado a */dev/psaux*, representado un ratón de tipo PS2, o */dev/cdrom* a */dev/hdc*, un CD-ROM que es un dispositivo del segundo conector IDE y máster. Aquí encontramos los dispositivos IDE como */dev/hdx*, los scsi/sata */dev/sdx*... con x variando según el número de dispositivo. Hay que mencionar que las últimas distribuciones, respecto a los dispositivos de disco, soportan en el *kernel* emulación de dispositivos IDE como si fueran scsi; por eso es corriente ver hoy en día todos los discos como dispositivos de tipo */dev/sdx*. Aquí cabe comentar que, en su inicio, este directorio era estático, con los ficheros predefinidos, y/o configurados en determinados momentos. Actualmente, se utilizan técnicas tecnologías dinámicas (como *hotplug*, y principalmente *udev*), que permiten detectar dispositivos y crear los archivos */dev* dinámicamente, bien al inicio del sistema o durante la ejecución, con la inserción de dispositivos removibles.

- ***/etc***: ficheros de configuración. La mayoría de tareas de administración necesitarán examinar o modificar los ficheros contenidos en este directorio. Por ejemplo, */etc/passwd* contiene parte de la información de las cuentas de los usuarios del sistema. Se recomienda que la configuración de servicios se guarde en subdirectorios de */etc*, y no en el mismo */etc* directamente.

- ***/home***: contiene las cuentas de los usuarios, es decir, los directorios personales de cada usuario. Aunque FHS no lo fuerza, por tanto, es posible que las distribuciones o el administrador creen directorios diferentes (por ejemplo, */users* u otros). Los programas o útiles de administración no deberían presuponer que las cuentas de usuario se encuentran en este directorio.

- ***/lib***: las bibliotecas del sistema, compartidas por los programas de usuario, ya sean estáticas (extensión *.a*) o dinámicas (extensión *.so*). Por ejemplo, la biblioteca C estándar, en ficheros *libc.so* o *libc.a*. También en particular suelen encontrarse los módulos dinámicos del kernel Linux, en */lib/modules*.

- ***/mnt***: punto para montar (comando *mount*) sistemas de ficheros de forma temporal. Históricamente se había usado para montar dispositivos removibles, pero actualmente se prefiere para montar sistemas de ficheros de forma temporal.

- ***/media***: para punto de montaje habitual de dispositivos extraibles. Tipicamente, es donde encontraremos montados los dispositivos de CD o USB

removibles. En distribuciones con soporte de systemd, este directorio progresivamente se desplazará a */run/media/$nombre_usuario*.

- ***/opt***: suele colocarse el software añadido al sistema posterior a la instalación. Otra instalación válida es en */usr/local*.

- ***/run***: usado para incorporar datos en tiempo de ejecución, por diferentes programas, en especial para almacenar información desde el arranque del sistema. Tipicamente se habia utilizado */var/run* para este objetivo, pero se está migrando progresivamente a este directorio. Una de las funciones es almacenar PID (identificadores de proceso) de programas en ejecución. Suele disponerse, en caso de algunos servicios, de un subdirectorio para cada uno. También en sistemas con systemd presente, se ha trasladado a */run/media* el montaje de dispositivos removibles.

- ***/sbin***: utilidades de base del sistema. Suelen ser comandos reservados al administrador (*root*). Por ejemplo, */sbin/fsck* para verificar el estado de los sistemas de ficheros.
  En algunas distribuciones este esquema no es tan preciso, y suelen estar dispersos en varios directorios, como */sbin*, */usr/sbin*, */usr/local/sbin* principalmente. Generalmente se enlaza (o monta como sistema de ficheros) en */usr/sbin*, y */usr/local/sbin* suele guardarse para herramientas locales instaladas por el administrador.

- ***/svr***: directorio reservado para especificar los datos de los servicios que se están ejecutando en el sistema; la idea es incorporar un directorio por servicio. Si los datos son internos al servicio, y no consumibles/consultables por terceros, se recomienda ponerlos en */var/lib*. De momento, las distribuciones no han hecho un gran uso de este espacio y hsn mantenido muchos de los datos bien en directorios en */etc* o en */var/lib*, cuando podrían migrarse a este espacio.

- ***/tmp***: ficheros temporales de las aplicaciones o del propio sistema. Aunque son para la ejecución temporal, entre dos ejecuciones la aplicación/servicio no puede asumir que va a encontrar los anteriores ficheros. Incluso se recomienda a las distribuciones que, entre arranques del sistema, este directorio sea borrado.

- ***/usr***: diferentes elementos instalados en el sistema, normalmente solo de lectura. Algún software de sistema más completo se instala aquí, además de complementos multimedia (iconos, imágenes, sonidos, por ejemplo en */usr/share*) y la documentación del sistema (*/usr/share/doc*). También en */usr/local* se suele utilizar para instalar documentación, software local accesible a los usuarios o elementos complementarios. Como caso especial importante, */usr/bin*, que es donde suelen guardarse la mayoría de comandos ejecutables por los usuarios comunes del sistema. También podemos destacar */usr/include* y */usr/lib*, donde se almacenan los ficheros estándar

de include para el desarrollo en lenguaje C, y las librerías disponibles para programación.

- ***/var***: ficheros de registro de sesión o de estado (ficheros de tipo log) y/o errores del propio sistema y de diferentes servicios, tanto locales como de red. Por ejemplo, ficheros de sesión en */var/log*, contenido de los mails en */var/spool/mail* o trabajos de impresión en */var/spool/lpd*.

Estos son algunos de los directorios definidos en el FHS para el sistema raíz; luego se especifican algunas subdivisiones, como el contenido de los */usr* y */var* y los ficheros de datos y/o ejecutables típicos que se esperan encontrar como mínimo en los directorios (ved las referencias a los documentos FHS).

Con respecto a las distribuciones, Fedora/Red Hat sigue el estándar FHS muy de cerca. Solo presenta algunos cambios en los archivos presentes en */usr* y */var*. En */etc* suele existir un directorio por componente configurable, y hay algún directorio especial, como */etc/sysconfig*, donde se encuentra gran parte de la configuración de varios servicios básicos del sistema. En */opt*, */usr/local* no suele existir software instalado, a no ser que el usuario lo instale. Debian, por su parte, sigue el estándar, aunque añade algunos directorios de configuración especiales en */etc*. También ambos grupos de distribuciones suelen disponer en */etc/default* de algunas configuraciones por defecto de servicios o componentes del sistema.

Otro estándar en proceso es el LSB (Linux Standard Base) [Linc]. La idea de este es definir unos niveles de compatibilidad entre las aplicaciones, bibliotecas y utilidades, de manera que sea posible la portabilidad de las aplicaciones entre distribuciones sin demasiados problemas. Además del estándar, proporcionan conjuntos de prueba (tests) para verificar el nivel de compatibilidad. LSB en sí mismo es un recopilatorio de varios estándares aplicados a GNU/Linux.

Nota

Estándares de especificaciones: http://refspecs.linuxfoundation.org

LSB: http://refspecs.linuxfoundation.org/lsb.shtml y https://wiki.linuxfoundation.org/en/LSB

1.3. Documentación del sistema en línea

Uno de los aspectos más importantes para nuestras tareas de administración será disponer de la documentación correcta para nuestro sistema y el software instalado. Hay muchas fuentes de información, entre las que destacaremos las siguientes:

a) man es la ayuda por excelencia. Nos permite consultar el manual de GNU/Linux, que está agrupado en varias secciones, correspondientes a comandos administración, formatos de ficheros, comandos de usuario, llamadas de lenguaje C, etc. Normalmente, para obtener la ayuda asociada, tendremos suficiente con:

```
man comando
```

Cada página describiría el comando junto con sus opciones y aportaría algunos ejemplos de utilización. A veces, puede haber más de una entrada en el manual. Por ejemplo, es posible que haya una llamada C con igual nombre que un comando. En este caso, hay que especificar qué sección quiere mirarse:

```
man n comando
```

siendo n el número de sección (por ejemplo, 2 para las llamadas a sistema, o 3 para las rutinas de la librería C).

Existen también unas cuantas herramientas de exploración de los manuales, por ejemplo *xman* y *tkman*, que mediante interfaz gráfica facilitan el examen de las diferentes secciones, así como índices de los comandos (también los entornos KDE/Gnome permiten en sus ayudas acceder a las páginas man). Otro comando interesante es *apropos*, palabra que nos permitirá localizar páginas man que hablen de un tema determinado (asociado con la palabra buscada).

b) info es otro sistema de ayuda habitual. Es un programa desarrollado por GNU para la documentación de muchas de sus herramientas. Se trata de una herramienta textual en la que los capítulos y las páginas se pueden recorrer por medio de un sistema de navegación simple (basado en teclado).

c) Documentación de las aplicaciones: además de ciertas páginas man, es habitual incluir en las aplicaciones documentación extra, ya sea en forma de manuales o tutoriales, o simples guías de usuario. Estos componentes de documentación se instalan en el directorio */usr/share/doc* (o */usr/doc*, dependiendo de la distribución), donde se crea un directorio por paquete de aplicación (por lo general, la aplicación puede disponer de paquete de documentación por separado).

d) Sistemas propios de las distribuciones. Red Hat suele venir con unos CD de manuales de consulta que son instalables en el sistema y tienen formatos HTML o PDF. Fedora dispone de un proyecto de documentación en la web del proyecto. Debian trae los manuales como un paquete de software más y suelen instalarse en */usr/share/doc*. Por otra parte, dispone de herramientas que clasifican la documentación presente en el sistema y la organizan por menús para su visualización, como *dwww* o *dhelp*, que presentan interfaces web para examinar la documentación del sistema.

e) Por último, **los escritorios X**, como Gnome y KDE, también traen sistemas de documentación propios con su documentación y manuales, así como información para desarrolladores, ya sea en forma de ayudas gráficas en sus aplicaciones, o en aplicaciones propias que recopilan las ayudas (por ejemplo, *devhelp* en Gnome).

1.4. Herramientas de gestión de paquetes

En cualquier distribución, los paquetes son el elemento básico para tratar las tareas de instalación de nuevo software, actualización del existente o eliminación del no utilizado.

Básicamente, un paquete es un conjunto de ficheros que forman una aplicación o una unión de varias aplicaciones relacionadas, formando un único fichero (denominado paquete), con un formato propio y comprimido, que es el que se distribuye, ya sea vía CD, DVD o mediante acceso a servicios de ftp o web a repositorios públicos.

El uso de paquetes facilita añadir o quitar software, al considerarlo una unidad y no tener que trabajar con los ficheros individuales.

En determinadas ocasiones, algunas aplicaciones o componentes son distribuidos en más de un paquete, separando los componentes esenciales de aquellos opcionales o relacionados con desarrollo, *plugins* de usuario o componentes modulares. Podemos encontrar: *paquete* como el principal, mientras otros, como *paquete-common*, nos ofrezcan los ficheros comunes asociados; *paquete-devel* y/o *-libs*, los ficheros asociados a desarrollo y otras denominaciones para ficheros extra para la aplicación/componente del sistema.

En el contenido de la distribución (sus CD/DVD o imágenes ISO) los paquetes suelen estar agrupados por categorías como:

a) Base: paquetes indispensables para el funcionamiento del sistema (útiles, programas de inicio, bibliotecas de sistema).

b) Sistema: útiles de administración, comandos de utilidad.

c) Desarrollo (*developement*): útiles de programación, como editores, compiladores, depuradores...

d) Gráficos: controladores e interfaces gráficas, escritorios, gestores de ventanas.

e) Otras categorías.

Para la instalación de un paquete, será necesario efectuar una serie de pasos:

1) Previo (preinstalación): comprobar que existe el software necesario (y con las versiones correctas) para su funcionamiento (dependencias), ya sean bibliotecas de sistema u otras aplicaciones que sean usadas por el software.

2) Descompresión del contenido del paquete, copiando los ficheros a sus localizaciones definitivas, ya sean absolutas (tendrán una posición fija) o, si se permite, reubicadas a otros directorios.

3) Postinstalación: retocar los ficheros necesarios, configurar posibles parámetros del software, adecuarlo al sistema...

Dependiendo de los tipos de paquetes, estos pasos pueden ser automáticos en su mayoría (así es en el caso de RPM [Bai03] y DEB [Deb02]). También puede que se necesite hacerlos todos de forma manual (caso TGZ), dependiendo de las herramientas que proporcione la distribución.

Veremos, a continuación, quizás los tres paquetes más clásicos de la mayoría de distribuciones. Cada distribución suele tener uno por estándar y soporta alguno de los demás. También observaremos una serie de nuevas propuestas para sistemas de empaquetado "universales" que puedan ser usados de forma genérica por más de una distribución.

1.4.1. Paquetes TGZ

Los paquetes TGZ son quizás los de utilización más antigua. Las primeras distribuciones de GNU/Linux los utilizaban para instalar el software, y todavía los usan varias distribuciones (por ejemplo, Slackware) y algunos UNIX comerciales.

Son una combinación de ficheros unidos por el comando *tar* en un único fichero .tar, que luego ha sido comprimido por la utilidad gzip, suele aparecer con la extensión .tgz o bien .tar.gz. Asimismo, hoy en día es común encontrar los tar.bz2 que utilizan, en lugar de gzip, otra utilidad llamada bzip2 que, en algunos casos, consigue mayor compresión del archivo.

Este tipo de paquete no contiene ningún tipo de información de dependencias, y puede presentar tanto contenido de aplicaciones en formato binario como en código fuente. Podemos considerarlo como una especie de colección de ficheros comprimida.

Nota

Los paquetes TGZ son una herramienta básica a la hora de instalar software no organizado y son muy útiles para realizar procesos de *backup* y restauración de archivos.

En contra de lo que pudiera parecer, este es un formato muy utilizado y, sobre todo, por creadores o distribuidores de software externo a la distribución. Muchos creadores de software que trabajan para plataformas varias, como varios UNIX comerciales y diferentes distribuciones de GNU/Linux, lo prefieren como sistema más sencillo y portable.

Ejemplo

Uno de estos casos es el proyecto GNU, que distribuye su software en este formato (en forma de código fuente), ya que puede utilizarse en cualquier UNIX, ya sea un sistema propietario, una variante BSD o una distribución GNU/Linux.

Si se trata de formato binario, tendremos que tener en cuenta que sea adecuado para nuestro sistema; por ejemplo, suele ser común alguna denominación como la que sigue (en este caso, la versión 1.4 de un paquete):

```
paquete-i686-pc-linux-gnu-1.4-installer.tar.gz
```

donde tenemos el nombre del paquete, arquitectura a la que ha destinado *i686* (Pentium II o superiores o compatibles). Podría ser *i386*, *i586*, *i686*, *k6* (*amd k6*), *k7* (*amd athlon*), *amd64* u *x86_64* (para AMD64 y algunos intel de 64bits), o *ia64* (intel Itaniums). Otras para arquitecturas de otras máquinas, como sparc, powerpc, mips, hppa, alpha... Después nos indica que es para Linux, en una máquina PC, la versión del software 1.4.

Si fuese en formato fuente, suele aparecer como:

```
paquete-source-1.4.tar.gz
```

donde se nos indica la palabra *source*. En este caso, no menciona versión de arquitectura de máquina, lo que nos indica que está preparado (en principio) para compilarse en diferentes arquitecturas.

De otro modo, habría diferentes códigos para cada sistema operativo o fuente: *GNU/Linux*, *Solaris*, *bsd*...

El proceso básico con estos paquetes consiste en:

1) Descomprimir el paquete (no suelen utilizar *path* absoluto, con lo que se pueden descomprimir en cualquier directorio):

```
tar -xzvf fichero.tar.gz (o fichero.tgz)
```

Con el comando *tar* ponemos opciones de *x*: extraer ficheros, *z*: descomprimir, *v*: ver pasos proceso, *f*: dar nombre del fichero a tratar. Dependiendo de la compresión utilizada, *gzip* o *bzip2* (u otras), puede ser necesario cambiar la opción de descomprimir, consultar la página *man* del comando *tar*. Por ejemplo, usar *-j* para *bzip2*, *-J* para *xz*, etc.

También se puede hacer por separado (sin la *z* del *tar*):

```
gunzip fichero.tar.gz
```

(nos deja un fichero tar)

```
tar -xvf fichero.tar
```

2) Una vez tenemos descomprimido el *tgz*, tendremos los ficheros que contenía; normalmente, el software debe incluir algún fichero de tipo *Readme* o *Install*, donde nos especificarán las opciones de instalación paso a paso, y también posibles dependencias del software.

En primer lugar, habrá que verificar las dependencias (suelen indicarlas en el fichero con los pasos de instalación), por si disponemos del software adecuado. Si no, deberemos buscarlo e instalarlo.

Si se trata de un paquete binario, la instalación suele ser bastante fácil, ya que o bien directamente ya será ejecutable donde lo hayamos dejado, o traerá algún instalador propio. Otra posibilidad será que tengamos que hacerlo manualmente, con lo que bastará con copiar (*cp -r*, copia recursiva) o mover (comando *mv*) el directorio a la posición deseada.

Otro caso es el formato de código fuente. En este, antes de instalar el software, tendremos que hacer un paso de compilación. Para eso habrá que leer con cierto detalle las instrucciones que lleve el programa. Pero la mayoría de desarrolladores usan un sistema de GNU llamado *autoconf* (de autoconfiguración), en el que habitualmente se usan los siguientes pasos (si no aparecen errores):

a) ***./configure***: se trata de un *script* que configura el código para poder ser compilado en nuestra máquina, y verifica que existan las herramientas adecuadas. La opción *--prefix = directorio* permite especificar dónde se instalará el software. Normalmente, se soportan muchas opciones adicionales de configuración según el software (con *–help* se mostraran las que acepta).

b) ***make***: la compilación propiamente dicha.

c) ***make install***: la instalación del software a un lugar adecuado, especificado previamente como opción al *configure* o asumida por defecto.

Este es un proceso general, pero depende del software que lo siga o no; hay casos bastante peores, donde todo el proceso se tiene que realizar a mano, retocando ficheros de configuración o el mismo *makefile*, y/o compilando uno a uno los ficheros, pero esto, por suerte, es cada vez menos habitual.

En caso de querer borrar el software instalado, habrá que utilizar el desinstalador si nos lo proporcionan, o si no, borrar directamente el directorio o los ficheros que se instalaron, teniendo cuidado de posibles dependencias.

Los paquetes *tgz*, y otros como *tar.bz2*, *tar.xz*, son bastante habituales como mecanismo de *backup* en tareas de administración, por ejemplo, para guardar copias de datos importantes, hacer *backups* de cuentas de usuario, o guardar copias antiguas de datos que no sabemos si volveremos a necesitar. Suele uti-

lizarse el siguiente proceso: supongamos que queremos guardar copia del directorio "dir" *tar -cvf dir.tar dir* (*c:* compactar dir en el fichero dir.tar) *gzip dir.tar* (comprimir) o bien en una sola instrucción como:

```
tar -czvf dir.tgz dir
```

El resultado será un fichero *dir.tgz*. Hay que ser cuidadoso si nos interesa conservar los atributos de los ficheros, y permisos de usuario, así como posiblemente ficheros de enlace (links) que pudieran existir (deberemos examinar las opciones de *tar* para que se ajuste a las opciones de *backup* deseadas).

1.4.2. Fedora/Red Hat: paquetes RPM

El sistema de paquetes RPM [Bai] creado por Red Hat supone un paso adelante, ya que incluye la gestión de dependencias y tareas de configuración del software. Además, el sistema guarda una pequeña base de datos con los paquetes ya instalados, que puede consultarse y se actualiza con las nuevas instalaciones.

Los paquetes RPM, por convención, suelen usar un nombre como:

```
paquete-version-rev.arq.rpm
```

donde *paquete* es el nombre del software, *version* es la numeración de versión del software, *rev* suele ser la revisión del paquete RPM, que indica las veces que se ha construido, y *arq,* la arquitectura a la que va destinado el paquete, ya sea Intel/AMD (i386, i586, i686, x86_64, ia64) u otras como alpha, sparc, ppc... La "arquitectura" *noarch* suele usarse cuando es independiente, por ejemplo, un conjunto de *scripts*, y *src* en el caso de que se trate de paquetes de código fuente. La ejecución típica incluye la ejecución de rpm, con las opciones de la operación a realizar, junto con uno o más nombres de paquetes por procesar juntos.

Ejemplo

El paquete *apache-1.3.19-23.i686.rpm* indicaría que se trata del software Apache (el servidor web), en su versión 1.3.19, revisión del paquete RPM 23, para arquitecturas Pentium II o superiores.

Las operaciones típicas con los paquetes RPM incluyen:

- **Gestión de dependencias**: los paquetes RPM incorporan la idea de gestión de dependencias y de base de datos de los paquetes existentes.

- **Información del paquete**: se consulta sobre el paquete una información determinada, se usa la opción *-q* acompañada del nombre del paquete (con *-p* si se hace sobre un archivo rpm). Si el paquete no ha sido instalado todavía, la opción sería *-q* acompañada de la opción de información que se quiera pedir, y si se quiere preguntar a todos los paquetes instalados a la vez, la opción sería *-qa*. Preguntas a un paquete instalado pueden ser:

Consulta	Opciones RPM	Resultados
Archivos	*rpm -ql*	Lista de los archivos que contiene el paquete (pasado como parámetro)
Información	*rpm -qi*	Descripción del paquete
Requisitos	*rpm -qR*	Requisitos previos, bibliotecas o software

- **Instalación:** simplemente *rpm -i paquete.rpm*, o bien puede hacerse con URL donde encontrar el paquete, para descargarlo desde servidores FTP o web. Solo hay que utilizar la sintaxis ftp:// o http:// para dar la localización del paquete. La instalación podrá realizarse siempre que se estén cumpliendo las dependencias del paquete, ya sea software previo o bibliotecas que deberían estar instaladas. En caso de no cumplirlo, se nos listará qué software falta, y el nombre del paquete que lo proporciona. Puede forzarse la instalación (a riesgo de que no funcione) con las opciones *--force* o *--nodeps,* o simplemente no hacer caso de la información de las dependencias. La tarea de instalación (realizada por rpm) de un paquete conlleva diferentes subtareas:
  - Verificar las posibles dependencias.
  - Examinar por conflictos con otros paquetes previamente instalados.
  - Realizar tareas previas a la instalación.
  - Decidir qué hacer con los ficheros de configuración asociados al paquete si previamente existían.
  - Desempaquetar los ficheros y colocarlos en el sitio correcto.
  - Realizar tareas de postinstalación.
  - Finalmente, almacenar registro de las tareas realizadas en la base de datos de RPM.

- **Actualización:** equivalente a la instalación, pero comprobando primero que el software ya existe *rpm -U paquete.rpm*. Se encargará de borrar la instalación previa.

- **Verificación:** durante el funcionamiento normal del sistema, muchos de los archivos instalados cambian. En este sentido, RPM permite verificar los archivos para detectar las modificaciones, bien por proceso normal, bien por algún error que podría indicar datos corrompidos. Mediante *rpm -V paquete* verificamos un paquete concreto, y mediante *rpm -Va* los verificamos todos.

- **Eliminación:** borrar el paquete del sistema RPM (*-e* o *--erase*). Si hay dependencias, puede ser necesario eliminar otros primero.

Ejemplo

Para un caso remoto:

```
rpm -i ftp://sitio/directorio/paquete.rpm
```

nos permitiría descargar el paquete desde el sitio ftp o web proporcionado, con su localización de directorios, y proceder en este caso a la instalación del paquete.

Nota

Ver el sitio www.rpmfind.net

Hay que vigilar la procedencia de los paquetes, y solo utilizar fuentes de paquetes conocidas y fiables, ya sea del propio fabricante de la distribución, o de sitios en los que confiemos. Se nos ofrece junto con los paquetes alguna "firma" digital de estos para que podamos comprobar su autenticidad; suelen utilizarse las sumas md5 para comprobar que el paquete no se ha alterado, y otros sistemas como GPG (versión Gnu de PGP) para comprobar la autenticidad del emisor del paquete. Hay también en Internet diferentes almacenes de paquetes RPM genéricos, donde están disponibles para diferentes distribuciones que usen o permitan el formato RPM.

Para un uso seguro de paquetes, los repositorios (oficiales, y algunos de terceros) firman electrónicamente los paquetes, por ejemplo con el mencionado GPG. Esto nos permite asegurar (si disponemos de las firmas) que los paquetes proceden de la fuente fiable. Cada proveedor (el repositorio) incluye unos ficheros de firma PGP con la clave para su sitio. De los repositorios oficiales ya se encuentran instaladas, de terceros deberemos obtener el fichero de clave, e incluirla en RPM, típicamente:

```
$ rpm --import GPG-KEY-FILE
```

Siendo *GPP-KEY-FILE* el fichero clave GPG o la URL de dicho fichero, este fichero también tendrá suma md5 para comprobar su integridad. Podemos conocer las claves existentes en el sistema con:

```
$ rpm -qa | grep ^gpg-pubkey
```

Podemos observar más detalles a partir de la llave obtenida:

```
$ rpm -qi gpg-key-xxxxx-yyyyy
```

Para un paquete rpm concreto podremos comprobar si dispone de firma y cuál se ha utilizado con:

```
$ rpm --checksig -v <paquete>.rpm
```

Y para verificar que un paquete es correcto en base a las firmas disponibles, puede comprobarse con:

```
$ rpm -K <paquete.rpm>
```

Debemos ser cuidadosos en importar solo aquellas claves de los sitios en que confiemos. Cuando RPM encuentre paquetes con firma que no poseamos en nuestro sistema, o el paquete no esté firmado, nos avisará, y la acción ya dependerá de nuestra actuación.

En cuanto al soporte RPM en las distribuciones, en Fedora (Red Hat, y también en sus derivadas), RPM es el formato por defecto de paquetes y el que usa ampliamente la distribución para las actualizaciones, y la instalación de software. En Debian se utiliza el formato denominado DEB (como veremos posteriormente), pero también hay soporte para RPM (existe el comando rpm), pero solo para consulta o información de paquetes. En el caso de que sea imprescindible instalar un paquete rpm en Debian, se recomienda utilizar la utilidad *alien*, que permite convertir formatos de paquetes, en este caso de RPM a DEB, y proceder a la instalación con el paquete convertido.

Además del sistema base de empaquetado de la distribución, hoy en día cada una suele aportar un sistema de gestión de software intermedio de más alto nivel, que añade una capa superior al sistema base, facilitando las tareas de gestión del software y añadiendo una serie de utilidades para controlar mejor el proceso.

En el caso de Fedora (Red Hat y derivados) se utiliza el sistema YUM, que permite, como herramienta de más alto nivel, la instalación y gestión de paquetes en sistemas rpm, así como la gestión automática de dependencias entre los paquetes. Facilita el acceso a múltiples repositorios diferentes, en algunas distribuciones su configuración se basaba en el fichero */etc/yum.conf*, pero actualmente está repartida entre diversos directorios.

Nota

Ved YUM en http://yum.baseurl.org/

Nota

Para distribuciones de Fedora superiores a la versión 22, el nuevo sistema DNF substituye a YUM. Más sobre DNF en http://dnf.baseurl.org/

La configuración de yum se basa en:

```
/etc/yum.config | (fichero de opciones)
/etc/yum | (directorio para algunas utilidades asociadas)
/etc/yum.repos.d | (directorio de especificación de repositorios, un fichero para cada uno,
      se incluye información del acceso y localización de las firmas <i>gpg</i>)
```

Para las operaciones típicas de yum, un resumen sería:

Orden	Descripción
yum install <nombre>	Instalar el paquete con el nombre
yum update <nombre>	Actualizar un paquete existente

Orden	Descripción
yum remove <nombre>	Eliminar paquete
yum list <nombre>	Buscar paquete por nombre (solo nombre)
yum search <nombre>	Buscar más ampliamente
yum provices <file>	Buscar paquetes que proporcionen el fichero
yum update	Actualizar todo el sistema
yum upgrade	Ídem al anterior incluyendo paquetes adicionales

Fedora también proporciona algunas utilidades gráficas para la actualización, como Yumex, y dependiendo de la versión, han existido diferentes herramientas, pero de existencia y permanencia corta. Alguna otra (textual) como Fedup se ha convertido en el estándar para la actualización de la distribución entre versiones.

El sistema YUM, que se venía utilizando en la distribución Fedora y otras distribuciones derivadas de Red Hat, se está substituyendo progresivamente por el sistema DNF (Dandified YUM). Este nuevo sistema es la opción por defecto a partir de Fedora 22 para la gestión de alto nivel de paquetes en Fedora. Las demás distribuciones basadas en Fedora/Red Hat se irán adaptando progresivamente con migraciones desde YUM a DNF.

Nota

Ved DNF en http://dnf.baseurl.org/.

En el nuevo DNF se ha intentado mejorar ciertas deficiencias que se observaban en YUM:

- Se han mejorado las prestaciones.
- Se ha mejorado el uso intensivo de memoria que se hacía.
- Se ha reducido cierta lentitud en el proceso de resolución de dependencias, además de la posible corrupción de dependencias en algunos casos.
- Se ha corregido la dependencia de ciertas API no bien documentadas, y de lenguajes (para sus *scripts*) como Python (versión 2) cuando la mayoria de distribuciones están migrando a Python 3.

Para mejorar estas deficiencias se ha desarrollado un *core* del sistema, con una serie de librerías independientes que están asociadas a resolver las problemáticas individuales: gestión de paquetes y sus dependencias (mediante RPM, y las librerías *libsolv* y *hawkey*), gestión de metadatos y repositorios (*librepo*), y gestión de grupos de paquetes (*libcomps*). Esto permite disponer a los desarrolladores del sistema DNF de unas API claras, basadas en una serie de librerías externas que pueden mejorarse independientemente.

La interfaz de órdenes se ha mantenido prácticamente compatible con YUM y solamente deberá cambiarse el uso de la orden `yum` por `dnf`. Así, una instalación de paquete individual pasa a realizarse con:

```
$ dnf install package-name
```

Siendo DNF habitual en las distribuciones que lo han adoptado como alternativa a YUM, el propio YUM nos recuerda ante el uso de la orden `yum`, que este se encuentra obsoleto y nos advierte que hay que usar los nuevos comandos mediante un aviso semejante a este:

```
Yum command has been deprecated, use dnf instead.
See 'man dnf' and 'man yum2dnf' for more information.
To transfer transaction metadata from yum to DNF, run 'dnf migrate'
```

Nota

Respecto las diferencias de CLI, de DNF y de YUM puede encontrarse una lista actualizada en: http://dnf.readthedocs.io/en/latest/cli_vs_yum.html.

Como diferencias de línea de comandos de DNF y YUM, podemos destacar:

- Las opciones *update* y *upgrade* son exactamente iguales, no hay diferencia en DNF.

- *dnf remove nombrepaquete* borra todos los paquetes (previa confirmación) con el nombre mencionado, en caso de desear una versión concreta deberemos especificarla en la orden.

- Hay otras diferencias menores de subopciones especificas que pueden consultarse en Read the Docs o en la página *man* de la orden: `man dnf`.

Respecto a la configuración del sistema global, DNF utiliza su fichero de configuración */etc/dnf/dnf.conf* y todos los ficheros de descripción de repos (**.repo*) que se encuentran en */etc/yum.repos.d*. Por otra parte, los ficheros del sistema mantenidos en caché pueden encontrarse en */var/cache/dnf*.

Nota

Para una referencia completa de la configuración del sistema podemos consultar: http://dnf.readthedocs.io/en/latest/conf_ref.html.

1.4.3. Debian: paquetes DEB

Debian tiene herramientas interactivas como *tasksel*, que permiten escoger unos subconjuntos de paquetes agrupados por tipo de tareas: paquetes para X, para desarrollo, para documentación, etc., o como *dselect* que facilita navegar por toda la lista de paquetes disponible (hay miles) y escoger aquellos que queramos instalar o desinstalar. De hecho, estas son solo un *front-end* del gestor de software nivel intermedio APT (equivalente al yum en Fedora).

En el nivel de línea de comandos dispone de *dpkg*, que es el comando de más bajo nivel (sería el equivalente a rpm), para gestionar directamente los paquetes DEB de software [Deb], típicamente *dpkg -i paquete.deb* para realizar la instalación. Pueden realizarse todo tipo de tareas, de información, instalación, borrado o cambios internos a los paquetes de software.

El nivel intermedio (como el caso de yum en Fedora) lo presentan las herramientas APT (la mayoría son comandos *apt-xxx*). APT permite gestionar los paquetes por medio de una lista de paquetes actuales y disponibles a partir de varias fuentes de software, ya sea desde los propios CD de la instalación, sitios ftp o web (HTTP). Esta gestión se hace de forma transparente, de manera que el sistema es independiente de las fuentes de software.

La configuración del sistema APT se efectúa desde los archivos disponibles en */etc/apt*, donde */etc/apt/sources.list* es la lista de fuentes disponibles. Podría ser, por ejemplo:

```
deb http://http.us.debian.org/debian stable main contrib non-free
debsrc http://http.us.debian.org/debian stable main contrib non-free
deb http://security.debian.org stable/updates main contrib non-free
```

en que hay recopiladas varias de las fuentes "oficiales" para una Debian (*stable* en este caso), desde donde se pueden obtener los paquetes de software, así como las actualizaciones que estén disponibles. Básicamente, se especifica el tipo de fuente (web/ftp en este caso), el sitio, la versión de la distribución (*stable*), y categorías del software que se buscará (libre, o contribuciones de terceros o de licencia no libre o comercial).

Los paquetes de software están disponibles para las diferentes versiones de la distribución Debian. Existen paquetes para las versiones *stable*, *testing* y *unstable*. El uso de unos u otros determina el tipo de distribución (previo cambio de las fuentes de repositorios en *sources.list*). Pueden tenerse fuentes de paquetes mezcladas, pero no es muy recomendable, ya que se podrían dar conflictos entre las versiones de las diferentes distribuciones.

Un sistema muy potente

Los paquetes DEB de Debian son quizás el sistema de instalación más potente existente en GNU/Linux. Una de sus prestaciones más destacables es la independencia del sistema de las fuentes de los paquetes (mediante APT).

Una vez tenemos las fuentes de software configuradas, la principal herramienta para manejarlas en nuestro sistema es *apt-get*, que nos permite instalar, actualizar o borrar desde el paquete individual, hasta actualizar la distribución entera. Existe también un *front-end* a *apt-get*, llamado *aptitude*, cuya interfaz de opciones es prácticamente igual (de hecho, podría calificarse de emulador de *apt-get*, ya que la interfaz es equivalente). Como ventaja, aporta una mejor gestión de dependencias de los paquetes, y algoritmos para solucionar los conflictos de paquetes que pueden aparecer. De hecho, en las últimas versiones de Debian, *aptitude* es la interfaz por defecto en línea de comandos para la gestión

de paquetes. Pueden seguir usándose ambas sin problemas, pero ante problemas complejos, de resolución de conflictos entre paquetes, se recomienda *aptitude*, por aportar mejores resoluciones.

Algunas funciones básicas de *apt-get* son:

1) Instalación de un paquete particular:

```
apt-get install paquete
```

2) Borrado de un paquete:

```
apt-get remove paquete
```

3) Actualización de la lista de paquetes disponibles:

```
apt-get update
```

4) Actualización de la distribución, podríamos efectuar los pasos combinados:

```
apt-get update
apt-get upgrade
apt-get dist-upgrade
```

Mediante este último proceso, podemos mantener nuestra distribución actualizada permanentemente, actualizando los paquetes instalados y verificando las dependencias con los nuevos. Una herramienta útil para construir esta lista es *apt-spy*, que intenta buscar los sitios oficiales más rápidos, o *netselect*, que nos permite probar una lista de sitios. Por otro lado, podemos buscar las fuentes oficiales (configurarlas con *apt-setup*), o bien copiar algún fichero de fuentes disponible. Software adicional (de terceros) puede necesitar añadir otras fuentes más (a */etc/apt/sources.list*), se pueden obtener listas de sitios no oficiales de fuentes alternativas (pero cabe ser precavido con su uso, por problematicas de seguridad).

La actualización del sistema en particular genera una descarga de un gran número de paquetes (en especial en *unstable)*, lo que hace recomendable vaciar la cache, el repositorio local, con los paquetes descargados (se mantienen en */var/cache/apt/archive*) que ya no vayan a ser utilizados, bien con *apt-get clean,* para eliminarlos todos, o bien con *apt-get autoclean*, para eliminar aquellos paquetes no necesarios porque ya hay nuevas versiones y ya no serán necesarios (en principio). Hay que tener en cuenta que no volvamos a necesitar estos paquetes por razones de reinstalación, porque en este caso tendríamos que volver a descargarlos.

El sistema APT también permite lo que se denomina SecureAPT, que es la gestión segura de paquetes mediante verificación de sumas (md5) y la firma de fuentes de paquetes (de tipo GPG). Si durante la descarga no están disponibles las firmas, *apt-get* informa de ello y genera un listado con los paquetes no firmados, pidiendo si se van a dejar instalar o no, dejando la decisión al administrador. Se obtiene la lista de fuentes confiables actuales con:

```
# apt-key list
```

Las claves *gpg* de los sitios oficiales de Debian son distribuidas mediante un paquete. Las instalamos:

```
apt-get install debian-archive-keyring
```

evidentemente considerando que tenemos *sources.list* con los sitios oficiales. Se espera que por defecto (dependiendo de la versión Debian) estas claves ya se instalen, en la instalación inicial del sistema. Para otros sitios no oficiales (que no proporcionen la clave en paquete), pero que consideremos confiables, podemos importar su clave, obteniéndola desde el repositorio (tendremos que consultar dónde tienen la clave disponible, no hay un estándar definido, aunque suele estar en la página web inicial del repositorio). Se utiliza *apt-key add* con el fichero, para añadir la clave, o también:

```
# gpg --import fichero.key
# gpg --export --armor XXXXXXXX | apt-key add -
```

siendo *X* un hexadecimal relacionado con la clave (ved instrucciones del repositorio para comprobar la forma recomendada de importar la clave y los datos necesarios).

Otra funcionalidad importante del sistema APT son las funciones de consulta de información de los paquetes, con la herramienta *apt-cache*, que nos permite interactuar con las listas de paquetes de software Debian.

Ejemplo:

La herramienta *apt-cache* dispone de opciones que nos permiten buscar información sobre los paquetes, como por ejemplo:

1) Buscar paquetes sobre la base de un nombre incompleto:

```
apt-cache search nombre
```

2) Mostrar la descripción del paquete:

```
apt-cache show paquete
```

3) De qué paquetes depende:

```
apt-cache depends paquete
```

Otra herramienta o funcionalidad de *apt* interesante es *apt-show-versions*, que nos especifica qué paquetes pueden ser actualizados (y por qué versiones, ved opción *-u*).

Otras tareas más específicas necesitarán realizarse con la herramienta de más bajo nivel, como *dpkg*. Se puede, por ejemplo, obtener la lista de archivos de un paquete determinado ya instalado:

```
dpkg -L paquete
```

O la lista de paquetes entera con:

```
dpkg -l
```

O buscar de qué paquete proviene un elemento (fichero, por ejemplo):

```
dpkg -S fichero
```

Este en particular funciona para paquetes instalados; *apt-file* permite también buscar para paquetes todavía no instalados.

Por último, cabe mencionar también algunas herramientas gráficas para APT, como *synaptic*. O las textuales ya mencionadas, como *aptitude* o *dselect*.

En conclusión, cabe destacar que el sistema de gestión APT (en combinación con el base *dpkg*) es muy flexible y potente a la hora de gestionar las actualizaciones, y es el sistema de gestión de paquetes usado en Debian y sus distribuciones derivadas, como Ubuntu, Kubuntu, Knoppix, Linex, etc.

1.4.4. Nuevos formatos de empaquetado: Snap y Flatpak

En los últimos tiempos han surgido nuevas alternativas a los sistemas de paquetes tradicionales para disminuir algunos de los problemas asociados al uso de los sistemas previos. Por una parte como principales contendientes en este punto, cabe señalar a Snappy/Snap (denominado así dependiendo de si hacemos referencia al sistema de empaquetado o al paquete individual) y a Flatpak.

Entre los problemas habituales de los sistemas de empaquetado clásicos podemos destacar los siguientes:

- Los paquetes son excesivamente dependientes de la distribución. A pesar de ser un estándar concreto (como DEB o RPM) pueden depender de paquetes específicos de la distribución concreta, o puede haber dependencias de librerías o versiones de estas que pueden no encontrarse en la distribución destino.

- Lo anterior crea una "fragmentación" de los paquetes GNU/Linux, que hace casi imposible empaquetar aplicaciones de forma válida para diferentes distribuciones GNU/Linux.

- En una misma distribución, el paquete puede necesitar versiones de bibliotecas anteriores o posteriores a las existentes, creando dependencias difíciles (o imposibles en algunos casos) de resolver.

- No son posibles los paquetes "universales" válidos para cualquier distribución.

Los nuevos sistemas como Snappy/Snap (proveniente de Canonical, desarrollador de Ubuntu) y Flatpak (antes conocido como xdg-app, y promovido por desarrolladores de la comunidad Gnome y Red Hat) prometen en cambio diferentes prestaciones (resueltas en este momento con diferentes resultados):

- Facilitar al desarrollador un empaquetamiento en un único formato que permite la instalación prácticamente en cualquier distribución GNU/Linux. Todo dependerá de que cada distribución decida dar soporte a uno o más de estos nuevos formatos.

- Se promete que los paquetes en estos nuevos formatos estarán aislados unos de otros y también de las partes criticas del sistema operativo, de manera que se maximice la seguridad.

- Alguno de ellos (Flatpak) permite empaquetar la aplicación directamente con las librerías que utiliza, de manera que no dependa de las librerías finales disponibles en el sistema, sino que la aplicación esté empaquetada con todo aquello necesario para funcionar. El concepto es bastante similar al concepto de aplicación instalable en MacOS X de Apple. Los críticos de este concepto aducen que se crean infinidad de copias de las mismas librerías, con múltiples versiones, difíciles de controlar (*bugs*, seguridad en especial), además de consumir en exceso memoria de la máquina y también disco para el almacenaje. Flatpak tiene alguna solución por ejemplo para la existencia de múltiples copias, utilizando deduplicación de librerías: si ya existen copia de ellas en el sistema las usa sin añadir copias extra.

- Flatpak son aplicaciones de tipo *sandboxed*. La aplicación solo puede verse a sí misma, y a un número limitado de librerías y interfaces (API) del sistema operativo.

- Snappy se usa tanto en paquetes para servidor o como para aplicaciones de escritorio, mientras que Flatpak (y su relación con Gnome, aunque podrá ser utilizado en otros entornos de ventanas) está pensado para correr aplicaciones en sesiones de usuario, aprovechando servicios como *dbus* y *systemd*.

- En el caso de Snap, existe el concepto de repositorio centralizado (Snap Store o Ubuntu myApps u otros nombres dependiendo de la plataforma destino) controlado por Canonical, mientras que en Flatpak no existe tal concepto, solamente repositorios múltiples que añadir para tener fuentes de diferentes paquetes.

Veamos algunas pruebas de los nuevos sistemas de paquetes en diferentes distribuciones en las que actualmente se encuentran más extendidos.

Snappy

En estos momentos Snappy/Snap se encuentra implementado (como nativo) en Ubuntu a partir de la versión 16.04 LTS, y también está disponible en mayor o menor grado en otras distribuciones. Para su uso en la distribución debe estar instalado el paquete *snapd* (o nombre similar dependiendo de la distribución), que incluye el comando `snap` para la gestión de paquetes. Si vamos a crear paquetes, también será necesario el paquete *snapcraft*.

Nota

Ved construcción de paquetes Snap por parte del desarrollador en: http://snapcraft.io/create/.

Veamos en Ubuntu (>= 16.04) una pequeña sesión de los comandos (CLI) de interfaz de usuario con este sistema de empaquetado.

Entre otras en la siguiente sesión de comando *snap* observamos:

`snap help`	Para obtener información de las opciones disponibles.
`snap find [`*`paquete`*`]`	Para buscar paquetes disponibles, o por nombre o por parte de este. Podemos combinar mediante pipe con grep para encontrar otras palabras asociadas al paquete en sus comentarios.
`snap install` *`paquete`*	Para instalar un paquete determinado.
`snap refresh` *`paquete`*	Para actualizar un paquete previamente instalado, si está disponible una nueva versión se actualizará.
`snap list`	Para listar paquetes instalados en el sistema. Podemos combinar con grep para encontrar un paquete/palabra determinada de los paquetes instalados.
`snap remove` *`paquete`*	Para desinstalar un paquete.
`snap changes`	Para listar cambios recientes en el sistema *snap*.

```
user@ubuntu:~$ sudo snap help
Usage:
snap [OPTIONS] <command>

The snap tool interacts with the snapd daemon to control the snappy software platform.
```

```
Available commands:
  abort          Abort a pending change
  ack            Adds an assertion to the system
  change         List a change's tasks
  changes        List system changes
  connect        Connects a plug to a slot
  create-user    Creates a local system user
  disconnect     Disconnects a plug from a slot
  find           Finds packages to install
  help           Help
  install        Install a snap to the system
  interfaces     Lists interfaces in the system
  known          Shows known assertions of the provided type
  list           List installed snaps
  login          Authenticates on snapd and the store
  logout         Log out of the store
  refresh        Refresh a snap in the system
  remove         Remove a snap from the system
  run            Run the given snap command
  try            Try an unpacked snap in the system

user@ubuntu:~$ sudo snap find
Name           Version    Developer      Notes    Summary
ab             1.0        snappy-test    -        Test snap with shortest name
ag-mcphail     1.0.1      njmcphail      -        The Silver Searcher - mcphail's build and
                                                    upstream git version
alsa-utils     1.1.0-1    woodrow        -        Utilities for configuring and using ALSA
apktool        2.1.1      ligboy         -        A tool for reverse engineering 3rd party,
                                                    closed, binary Android apps.
atom-cwayne    1.9.0      cwayne18       -        Atom Text Editor
audovia        3.3.1      songbuilder    -        Database application for making music using
                                                    JFugue MusicStrings
base           16.04+20160525.05-00  canonical  -  Base content for snapd
blender-tpaw   2.77a      tpaw           -        Blender is the free and open source 3D creation
                                                    suite.
cassandra      3.7        ev             -        Cassandra distributed database
cla-check      0.1        kyrofa         -        Check if Canonical's Contributor License
                                                    Agreement has been signed
click-parser   3.0.5      bhdouglass     -        Extract data from Ubuntu's click & snap
                                                    packages
compass-straightedge  1   caozhen        -        Construct geometric figures with
                                                    compass-and-straightedge construction
dolgia-test-tools  1.0    fgimenez       -        dolgia test tools
drmips         2.0.1-4    brunonova      -        Educational MIPS simulator
eeevil         1          chipaca        -        very evil
ejabberd       16.04      wyrddreams     -        ejabberd XMPP server
electrum-tpaw  2.6.4-tpaw3  tpaw         -        Lightweight Bitcoin Client
```

```
filebot          4.7~snap1  pointplanck     9.99EUR  The ultimate TV and Movie Renamer / Subtitle
                                                      Downloader
foobar21         2.1        testuser        -        This is a test snap
freecad          0.17       vejmarie        -        This is the first beta for freecad 0.17
                                                      supporting OCCT 7 / Netgen and new Smesh
                                                      version
freechartgeany 2.0.3-1snap  lucast70        -        Technical analysis software for stocks
gdoc-html-cleaner  0.1.2    caldav          -        Download Google Docs as cleaned HTML files
ghex-udt         1          canonical       -        Hex Editor
gmailfilter      0.0.1a     thomi           -        Programmatically filter gmail messages
gnuchess         2.1        tomechangosubanana  -   Plays a game of chess, includes GUI and CLI.
                                                      Run "gnuchess.readme" for more information.
gtwang-demo      1.0        gtwang          -        G.T.Wang demo application.
hangups          0.3.6      tomdryer        -        Third-party instant messaging client for Google
                                                      Hangouts
hello            2.10       canonical       -        GNU Hello, the "hello world" snap
..................

user@ubuntu:~$ sudo snap find | grep world
hello            2.10       canonical       -        GNU Hello, the "hello world" snap

user@ubuntu:~$ sudo snap install hello
Name   Version  Rev  Developer  Notes
hello  2.10     20   canonical  -

user@ubuntu:~$ sudo snap refresh hello
error: cannot perform the following tasks:
- Download snap "hello" from channel "stable" (revision 20 of snap "hello" already installed)

user@ubuntu:~$ sudo snap list
Name         Version               Rev  Developer  Notes
hello        2.10                  20   canonical  -
ubuntu-core  16.04+20160531.11-56  122  canonical  -

user@ubuntu:~$ sudo snap remove hello

Done
user@ubuntu:~$ sudo snap changes
ID   Status  Spawn                 Ready                 Summary
20   Done    2016-07-09T15:44:25Z  2016-07-09T15:44:28Z  Install "hello" snap
21   Error   2016-07-09T15:44:36Z  2016-07-09T15:44:36Z  Refresh "hello" snap
22   Done    2016-07-09T15:46:19Z  2016-07-09T15:46:20Z  Remove "hello" snap
```

Los paquetes en el sistema de ficheros se encuentran en el directorio */snap*, y dependiendo del tipo, si son de ejecución grafica se añaden a los menús del sistema, o si son ejecutables vía instrucciones, serán llamados desde el *shell* con nombres parecidos al nombre del paquete. Siempre podemos examinar los directorios:

- */snap/bin,* para encontrar los binarios disponibles para ejecutar.

- */snap/nombre_paquete/current/bin*, para encontrar los binarios/ejecutables asociados a un paquete, en su versión más actual. Aunque el lanzador de la aplicación –puede que en forma de *shell script* para realizar configuraciones previas– se encuentra habitualmente en el previo */snap/bin*, es recomendable la ejecución de la aplicación vía el comando disponible en el primer directorio, o acceder a los menús de sistema donde se haya instalado la aplicación.

Flatpak

En el caso de Flatpak probaremos su uso a partir de una distribución Fedora, como mínimo la versión 23 (se distribuyo oficialmente en Fedora 24). También está disponible en otras distribuciones que incluyen en su repositorio el paquete Flatpak.

Nota

Podéis ver la disponibilidad actual en de Flatpak en http://flatpak.org/getting.html. Para creadores de paquetes, véase la guía para desarrolladores en http://flatpak.org/index.html#developers.

En Fedora (>=23) podemos realizar la instalación con:

```
# dnf install flatpak
```

Una vez instalado el paquete, dispondremos del comando `flatpak` mediante el cual realizaremos las tareas siguientes (normalmente estas tareas contienen ciertos pasos):

- Instalar los repositorios de paquetes necesarios. Incluye importar las llaves GPG para identificar al repositorio y sus paquetes, e importar los repositorios concretos a los que queremos acceder.

- Instalar desde el repositorio incorporado el *runtime* necesario. Este proporciona todas las dependencias necesarias para las aplicaciones en el repositorio.

- Podemos así ya ver con diferentes opciones, qué paquetes hay disponibles, instalarlos y ejecutar las aplicaciones instaladas.

En este cuadro resumen, vemos algunos de los comandos que haremos servir en la sesión de comandos posterior:

`flatpak remote-add --gpg-import=llavegpg repo urlrepo`	Para añadir un repositorio `repo` identificado por una `llavegpg` (descargada previamente), y localizado en la URL `urlrepo`.

`flatpak remote-list`	Para listar repositorios remotos disponibles.
`flatpak install repo runtime version`	Para instalar desde el repositorio `repo` el *runtime* `runtime` con el número de versión concreta `version`.
`flatpak remote-ls repo --app`	Para listar las aplicaciones disponibles en el `repo`.
`flatpak install repo aplicacion stable`	Para instalar la `aplicacion` en su versión estable desde el repositorio `repo`.
`flatpak list`	Para listar aplicaciones disponibles.
`flatpak run nombre`	Para ejecutar una aplicación `nombre` disponible.
`flatpak update`	Para actualizar desde repositorio todas las aplicaciones instaladas.

Veamos una sesión de instrucciones para la instalación de algunos paquetes de Gnome desde los repositorios Flatpak de Gnome (gnome y gnome-apps), e incorporando el *runtime* de Gnome (alguna de las instrucciones necesitan permisos de *root*):

```
$ wget https://sdk.gnome.org/keys/gnome-sdk.gpg
--2016-07-10 18:47:52--  https://sdk.gnome.org/keys/gnome-sdk.gpg
Resolviendo sdk.gnome.org (sdk.gnome.org)... 209.132.180.169
Conectando con sdk.gnome.org (sdk.gnome.org)[209.132.180.169]:443... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 629
Grabando a: "gnome-sdk.gpg"

gnome-sdk.gpg                   100%[=====================================================>]
629  --.-KB/s    in 0s

2016-07-10 18:47:53 (73,4 MB/s) - "gnome-sdk.gpg" guardado [629/629]

$ flatpak remote-add --gpg-import=gnome-sdk.gpg gnome https://sdk.gnome.org/repo/
$ flatpak remote-add --gpg-import=gnome-sdk.gpg gnome-apps https://sdk.gnome.org/repo-apps/

$ flatpak remote-list
gnome
gnome-apps

$ flatpak remote-ls gnome
org.freedesktop.BasePlatform
org.freedesktop.BaseSdk
org.freedesktop.Platform
org.freedesktop.Platform.Locale
org.freedesktop.Platform.Var
org.freedesktop.Sdk
org.freedesktop.Sdk.Debug
org.freedesktop.Sdk.Locale
org.gnome.Platform
```

```
org.gnome.Platform.Locale
org.gnome.Sdk
org.gnome.Sdk.Debug
org.gnome.Sdk.Locale

$ flatpak install gnome org.gnome.Platform
error: Multiple branches available for org.gnome.Platform, you must specify one of: 3.20, 3.18, 3.16

$ flatpak install gnome org.gnome.Platform 3.20
9 delta parts, 71 loose fetched; 167139 KiB transferred in 23 seconds
Installing related: org.gnome.Platform.Locale
30 metadata, 135 content objects fetched; 803 KiB transferred in 7 seconds

$ flatpak remote-ls gnome-apps --app
org.gnome.Builder
org.gnome.Calculator
org.gnome.Calendar
org.gnome.Characters
org.gnome.Dictionary
org.gnome.Epiphany
org.gnome.Evince
org.gnome.Maps
org.gnome.Polari
org.gnome.Rhythmbox3
org.gnome.Todo
org.gnome.Weather
org.gnome.bijiben
org.gnome.clocks
org.gnome.eog
org.gnome.gedit
org.gnome.iagno

$ flatpak install gnome-apps org.gnome.gedit
1 delta parts, 1 loose fetched; 1725 KiB transferred in 5 seconds
Installing related: org.gnome.gedit.Locale

$ flatpak install gnome-apps org.gnome.Calculator stable
1 delta parts, 1 loose fetched; 665 KiB transferred in 3 seconds
Installing related: org.gnome.Calculator.Locale
6 metadata, 1 content objects fetched; 20 KiB transferred in 2 seconds

$ flatpak list
org.gnome.Calculator
org.gnome.gedit

$ flatpak run org.gnome.Calculator
```

En cuanto a la estructura de sistema de ficheros, Flatpak (para las versiones examinadas en Fedora) genera entre otros:

- En el usuario local, un directorio *.var/app/nombreapp* de configuraciones para la ejecución de la aplicación.

- En el sistema en */var/lib/flatpak* se encuentran las aplicaciones instaladas, los repositorios disponibles y los *runtimes* utilizados por el sistema.

- En el sistema en */var/lib/flatpak/app/nombreapp*, se mantiene la aplicación en sí misma, con la estructura de ficheros necesaria para su ejecución por parte de flatpak. Por ejemplo, para la última version de una aplicación *appname* se encontrarían sus ficheros en: */var/lib/flatpak/app/appname/current/active/files,* donde entre otros podremos observar los binarios o *scripts* de la aplicación, así como la inclusión de las librerias que necesita la aplicación para su ejecución.

Una vez examinados los nuevos sistemas Snappy y Flatpak, concluiremos comentando algunos aspectos sobre los nuevos sistemas de paquetes pendientes de resolver y las problemáticas que pueden generar:

- Tanto en un caso como en otro (Snap/Flatpak) existen algunos problemas pendientes con los modelos de seguridad, especialmente para las aplicaciones gráficas en X Window, que por razones de inseguridad de la plataforma (por ejemplo, una aplicación puede usar X Window para enviar falsos eventos de teclado y causar que una aplicación interactue de una forma no deseada), impiden poder disponer de cierto grado de seguridad o *sandbox* aislando las aplicaciones que hagan uso de la interfaz gráfica. Se espera que esta parte vaya solucionandose a medida que el X Window Server se vaya substituyendo por alternativas como Wayland (que podria aprovechar Flatpak para mejorar la seguridad) o Mir (en el caso de Ubuntu y Snap).

- Por lo demás, en la medida de que se vayan solucionando los diversos problemas es de esperar que las distribuciones adopten los nuevos sistemas (Snap y Flatpak) de forma dual, o bien se decidan por uno de ellos (creando una nueva fragmentación parecida al caso DEB/RPM).

- También existen en este momento, otras propuestas como AppImage y OrbitalApps. La primera es más simple que las nuevas propuestas mencionadas, y la segunda está pensada para desarrollar aplicaciones portables, como las que podríamos llevar en USB entre varios sistemas.

1.5. Herramientas genéricas de administración

En el campo de la administración, también podríamos considerar algunas herramientas, como las pensadas de forma genérica para la administración. Aunque cabe indicar que para estas herramientas es difícil mantenerse al dia, debido a los actuales planes de versiones de las distribuciones, con evolución muy rápida. Algunas de estas herramientas (aunque en un momento determinado pueden no ser funcionales al completo en una distribución dada) son:

1) Webmin: es una herramienta de administración pensada desde interfaz web. Funciona con una serie de *plugins* que se pueden añadir para cada servicio a administrar y cuenta con formularios donde se especifican los parámetros de configuración de los servicios. Además, ofrece la posibilidad (si se activa) de permitir administración remota desde cualquier máquina con navegador.

Nota

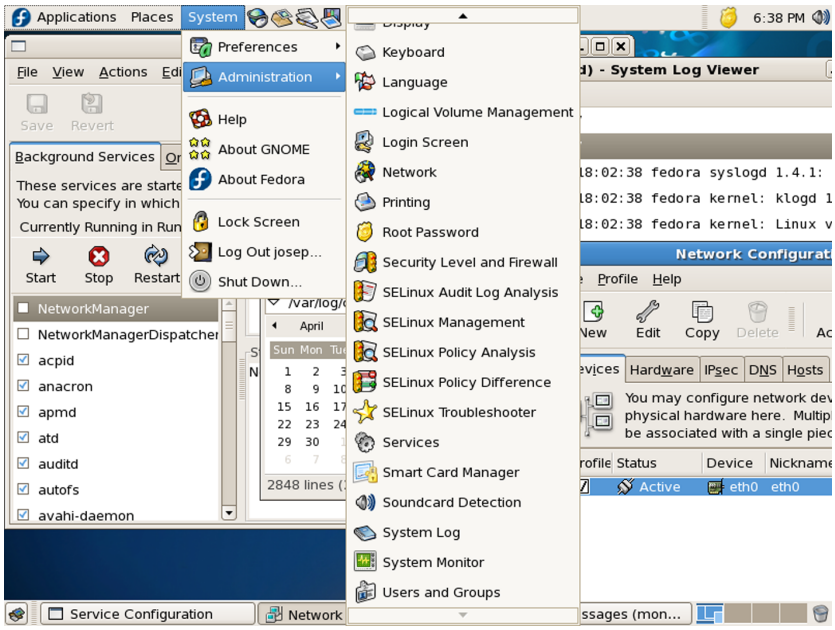
Podéis encontrar esta herramienta en Webmin: www.webmin.com

2) Otras en desarrollo, como **cPanel**, **ISPConfig**...

Por otra parte, en los entornos de escritorio de Gnome y KDE, suelen disponer del concepto de "Panel de control", que permite gestionar tanto el aspecto visual de las interfaces gráficas como tratar algunos parámetros de los dispositivos del sistema.

En cuanto a las herramientas gráficas individuales de administración, la propia distribución de GNU/Linux ofrece algunas directamente (herramientas que acompañan tanto a Gnome como KDE), herramientas dedicadas a gestionar un dispositivo (impresoras, sonido, tarjeta de red, etc.) y otras para la ejecución de tareas concretas (conexión a Internet, configurar arranque de servicios del sistema, configurar X Window, visualizar logs...). Muchas de ellas son simples *frontends* (o carátulas) a las herramientas básicas de sistema, o bien están adaptadas a particularidades de la distribución.

Cabe destacar, en especial en este apartado, la distribución Fedora (Red Hat y derivados), que intenta disponer de distintas utilidades (más o menos minimalistas) para diferentes funciones de administración. Las podemos encontrar en el escritorio (en el menú de administración), o en comandos como *system-config-xxxxx* para diferentes funcionalidades como gestión de pantalla, impresora, red, seguridad, usuarios, paquetes, etc. Aunque en las últimas versiones de Fedora se están substituyendo progresivamente por el concepto de panel de control (sea la configuración de Gnome o el panel KDE) con funcionalidades parecidas a estas utilidades. Podemos ver en la figura algunas de ellas:



Algunas utilidades gráficas de administración en Fedora.

1.6. Otras herramientas

En el espacio limitado de esta unidad no pueden llegar a comentarse todas aquellas herramientas que nos pueden aportar beneficios para la administración. Citaremos algunas de las herramientas que podríamos considerar básicas:

Nota

Consultad las páginas man de los comandos, o una referencia de herramientas como [Stu01].

- **Los múltiples comandos de utilidad UNIX básicos**: `grep`, `awk`, `sed`, `find`, `diff`, `gzip`, `bzip2`, `cut`, `sort`, `df`, `du`, `cat`, `more`, `file`, `which`...

- **Los editores**, imprescindibles para cualquier tarea de edición, cuentan con editores como *Vi*, muy utilizado en tareas de administración por la rapidez de efectuar pequeños cambios en los ficheros. *Vim* es el editor compatible *Vi*, que suele incorporar GNU/Linux. Permite sintaxis coloreada en varios lenguajes. *Emacs*, editor muy completo, adaptado a diferentes lenguajes de programación (sintaxis y modos de edición), dispone de un entorno muy completo y de una versión *X* denominada *xemacs*. *Joe* es un editor compatible con Wordstar. Y muchos otros...

- **Lenguajes de tipo *script***, útiles para administración, como *Perl*, muy adecuado para tratamiento de expresiones regulares y análisis de ficheros (filtrado, ordenación, etc.); *PHP*, lenguaje muy utilizado en entornos web; *Python*, otro lenguaje que permite hacer prototipos rápidos de aplicaciones...

- **Herramientas de compilación y depuración de lenguajes de alto nivel**: *GNU gcc* (compilador de C y C++ entre otros), *gdb* (depurador), *xxgdb* (interfaz *X* para *gdb*), *ddd* (depurador para varios lenguajes).

2. Distribuciones: particularidades

Intentamos destacar ahora algunas diferencias técnicas menores (que cada vez se reducen más) en las distribuciones (Fedora/Red Hat y Debian) utilizadas [Mor03] [Soy12], que iremos viendo con más detalle a lo largo de las unidades, a medida que vayan apareciendo.

Cambios o particularidades de Fedora/Red Hat:

- **Uso del gestor de arranque *grub*** (una utilidad GNU). A diferencia de antiguas versiones de la mayoría de distribuciones, que solían usar *lilo*, Fedora utiliza *grub*. GRUB (Grand Unified Bootloader) tiene una configuración en modo texto (normalmente en */boot/grub/grub.conf*) bastante sencilla, y que puede modificarse en el arranque. Es quizás más flexible que *lilo*. Últimamente las distribuciones tienden al uso de *grub*; Debian también lo incluye ya por defecto. La mayoría de distribuciones actuales han migrado ya a GRUB 2, que incluye un sistema más modular de configuración, así como soporte ampliado de parémetros del *kernel* Linux, y soporte mejorado para diversos sistemas operativos. Grub 2 dispone del fichero principal de configuración normalmente en */etc/grub/grub.cfg* o */etc/grub2/grub.cfg*.

- **Gestión de alternativas**. En el caso de que haya más de un software equivalente presente para una tarea concreta, mediante un directorio (*/etc/alternatives*) se indica cuál es la alternativa que se usa. Este sistema se tomó prestado de Debian, que hace un uso amplio de él en su distribución.

- **Programa de escucha de puertos TCP/IP basado en *xinetd***. En */etc/xinetd.d* podemos encontrar, de forma modular, los ficheros de configuración para algunos de los servicios TCP/IP, junto con el fichero de configuración */etc/xinetd.conf*. En los sistemas UNIX clásicos, el programa utilizado es el *inetd*, que poseía un único fichero de configuración en */etc/inetd.conf*, caso, por ejemplo, de la distribución Debian, que utiliza *inetd* en las versiones *stable*, dejando *xinetd* como opción. En las últimas distribuciones Fedora, xinetd se ha convertido en opcional, no está presente por defecto pero puede instalarse *a posteriori* desde repositorio. También debido al impacto de systemd en GNU/Linux, muchas de las distribuciones, como veremos, han migrado la gestión de los servicios del operativo hacia este nuevo gestor de servicios.

- **Algunos directorios de configuración especiales**: */etc/profile.d*, archivos que se ejecutan cuando un usuario abre un shell; */etc/xinetd.d*, configuración de algunos servicios de red; */etc/sysconfig*, datos de configuración de varios aspectos y servicios del sistema; */etc/cron.*, varios directorios donde se especifican trabajos para hacer periódicamente (mediante *crontab*); */etc/*

pam.d, donde *pam* son los denominados módulos de autentificación: en cada uno de cuyos archivos se configuran permisos para el programa o servicio particular; */etc/logrotate.d*, configuración de rotación (cuando hay que limpiar, comprimir, etc.) de algunos de los ficheros de log para diferentes servicios.

En el caso de Debian:

- **Sistema de empaquetado propio basado en los paquetes DEB**, con herramientas de varios niveles para trabajar con los paquetes como: *dpkg*, *apt-get*, *dselect*, *tasksel*.

- **Debian sigue el FHS**, sobre la estructura de directorios, añadiendo algunos particulares en */etc*, como por ejemplo: */etc/default*, archivos de configuración, y valores por defecto para algunos programas; */etc/network*, datos y guiones de configuración de las interfaces de red; */etc/dpkg* y */etc/apt*, información de la configuración de las herramientas de gestión de paquetes; */etc/alternatives*, enlaces a los programas por defecto, en aquellos en los que hay (o puede haber) varias alternativas disponibles.

- **Sistema de configuración** de muchos paquetes de software por medio de la herramienta *dpkg-reconfigure*. Por ejemplo:

```
dpkg-reconfigure locales
```

  permite escoger las configuraciones regionales de idiomas y juegos de caracteres que estarán disponibles.

```
dpkg-reconfigure tzdata
```

  nos permite reconfigurar la zona horaria por defecto.

- **Utiliza configuración de servicios TCP/IP por *inetd***, configuración en fichero */etc/inetd.conf*. Dispone de una herramienta *update-inetd* para inhabilitar o crear entradas de servicios. Puede usarse xinetd como alternativa para controlar los servicios, en aquellos que tengan soporte.

- Algunos directorios de configuración especiales: */etc/cron.*, varios directorios donde se especifican trabajos para hacer periódicamente (mediante *crontab*); */etc/pam.d*, donde *pam* son módulos de autentificación.

- Debian dispone de repositorios: *free*, *non-free* y *contrib*, para separar la procedencia del software incluido. Mientras Fedora dispone de un solo genérico (en algunos casos se añaden repositorios por defecto de updates y extras).

- Debian tradicionalmente ha usado arranque de sistema con sysvinit, pero en las últimas versiones se está migrando progresivamente hacia el sistema systemd.

3. Niveles de arranque y servicios

Un primer punto importante en el análisis del comportamiento local del sistema es su funcionamiento en los llamados niveles de ejecución (o *runlevels*), que determinan (en el nivel) el modo actual de trabajo del sistema y los servicios que se proporcionan [Wm02].

Nota

Actualmente coexisten en las distribuciones GNU/Linux diferentes modelos de arranque del sistema y servicios, los más destacables son: sysvinit, upstart y systemd.

Un servicio es una funcionalidad proporcionada por la máquina, normalmente basada en *daemons* (o procesos en segundo plano de ejecución, que controlan peticiones de red, actividad del hardware, u otros programas que controlen alguna tarea).

A partir de este punto, describiremos el método clásico de arranque por defecto del sistema y de sus servicios (en inglés, *init system*). Este método clásico, denominado *sysvinit* (o *init* de tipo *SysV*), es el que clásicamente han seguido la mayoría de UNIX comerciales, y en el comienzo, la mayoría de distribuciones GNU/Linux. Progresivamente se está migrando a nuevos métodos como *Upstart* y *systemd*. Es necesario conocer los diversos métodos de arranque, ya que algunos mantienen cierta compatibilidad, y en las distribuciones Debian/Fedora, podemos encontrar diferentes métodos de arranque dependiendo de la versión de la distribución.

En *Sysvinit*, la activación o parada de servicios se realiza mediante la utilización de *scripts*. La mayoría de los servicios estándar, que suelen tener su configuración en el directorio */etc*, suelen controlarse mediante los *scripts* presentes en */etc/init.d/*. En este directorio suelen aparecer *scripts* con nombres similares al servicio donde van destinados, y se suelen aceptar parámetros de activación o parada.

3.1. Arranque SysVinit

El arranque Sysvinit se realiza:

```
/etc/init.d/servicio start
```

Arranque del servicio.

```
/etc/init.d/servicio stop
```

Parada del servicio.

```
/etc/init.d/servicio restart
```

Parada y posterior arranque del servicio.

Cuando un sistema GNU/Linux arranca, primero se carga el *kernel* del sistema, después se inicia el primer proceso, denominado *init*, que es el responsable de ejecutar y activar el resto del sistema, mediante la gestión de los niveles de ejecución (o *runlevels*).

Un nivel de ejecución es sencillamente una configuración de programas y servicios que se ejecutarán orientados a un determinado funcionamiento.

Nota

Podemos utilizar los comandos *runlevel* o *who -r*, para conocer el *runlevel* actual en el que nos encontramos.

Los niveles típicos suelen ser (puede haber diferencias en el orden, en especial en los niveles 2-5, en la tabla la configuración en Fedora, y la recomendada por el standard LSB):

Runlevel	Función	Descripción
0	Parada	Finaliza servicios y programas activos, así como desmonta *filesystems* activos y para la CPU.
1	Monousuario	Finaliza la mayoría de servicios, permitiendo solo la entrada del administrador (*root*). Se usa para tareas de mantenimiento y corrección de errores críticos.
2	Multiusuario sin red	No se inician servicios de red, permitiendo solo entradas locales en el sistema.
3	Multiusuario	Inicia todos los servicios excepto los gráficos asociados a X Window.
4	Multiusuario	No suele usarse, típicamente es igual que el 3.
5	Multiusuario X	Igual que 3, pero con soporte X para la entrada de usuarios (*login* gráfico).
6	Reinicio	Para todos los programas y servicios. Reinicia el sistema.

Por el contrario, cabe señalar que Debian usa un modelo donde prácticamente los niveles 2-5 son equivalentes, realizando exactamente la misma función (aunque podría ocurrir que en alguna versión esto fuera a cambiar para coincidir con el estándar LSB).

Estos niveles suelen estar configurados en los sistemas GNU/Linux (y UNIX) por dos sistemas diferentes, el BSD, o el SystemV (a veces abreviado como sysinitV, o arranque SysV). En el caso clásico de las antiguas versiones de Fedora y Debian, se utilizaba el sistema SystemV clásico, que es el que mostraremos en esta sección (pero hay que comentar que estas distribuciones están migrando, o ya lo han hecho, hacia uso de systemd del que hablaremos más tarde), pero algunos UNIX utilizan el modelo BSD.

En el caso del modelo runlevel de SysV, cuando el proceso *init* (normalmente es el proceso con identificador PID=1) arranca, utiliza un fichero de configuración llamado */etc/inittab* para decidir el modo de ejecución en el que va a

entrar. En este fichero se define el runlevel por defecto (*initdefault*) en arranque (por instalación en Fedora el 5, en Debian el 2) y una serie de servicios de terminal por activar para atender la entrada del usuario.

Después, el sistema, según el runlevel escogido, consulta los ficheros contenidos en */etc/rcn.d*, donde *n* es el número asociado al *runlevel* (nivel escogido), en el que se encuentra una lista de servicios por activar o parar en caso de que arranquemos en el *runlevel* o lo abandonemos. Dentro del directorio encontraremos una serie de *scripts* o enlaces a los *scripts* que controlan el servicio.

Cada *script* posee un nombre relacionado con el servicio, una *S* o *K* inicial que indica si es el *script* para iniciar *(S)* o matar *(K)* el servicio, y un número que refleja el orden en que se ejecutarán los servicios.

Una serie de comandos de sistema sirven de ayuda para manejar los niveles de ejecución. Cabe mencionar:

- **Los *scripts***, que ya hemos visto, en */etc/init.d/* nos permiten arrancar, parar o reiniciar servicios individuales.

- ***telinit/init*** nos permite, como *root*, cambiar de nivel de ejecución; solo tenemos que indicar el número. Por ejemplo, necesitamos hacer una tarea crítica en *root*; sin usuarios trabajando, podemos hacer un *telinit 1* (también puede usarse *S*) para pasar a *runlevel* monousuario, y después de la tarea un *telinit 3* para volver a multiusuario. También puede utilizarse el comando *init* para la misma tarea, aunque *telinit* aporta algún parámetro extra. Por ejemplo, el reinicio típico de un sistema UNIX se hacía con *sync; sync; sync; init 6*. El comando *sync* fuerza el vaciado de los *buffers* del sistema de archivos, y luego reiniciamos en *runlevel* 6.

- ***shutdown*** permite parar (*'-h'* de *halt*) o reiniciar el sistema (*'-r'* de *reboot*). Puede darse un intervalo de tiempo para hacerse, o bien realizarse inmediatamente. Para estas tareas también existen los comandos *halt/poweroff* y *reboot*. Dependiendo de si el sistema tiene soporte de gestión de energía, pueden tener efectos diferentes (para las CPU solo en *halt*, o en *poweroff*, enviar comando ACPI de desconectar la alimentación eléctrica). Los reinicios básicamente controlan un init 6.

- ***wall*** permite enviar mensajes de advertencia a los usuarios del sistema. Concretamente, el administrador puede anunciar que se va a parar la máquina en un determinado momento. Comandos como *shutdown* suelen utilizarlo de forma automática.

- ***pidof*** permite averiguar el PID (*process* ID) asociado a un proceso. Con *ps* obtenemos los listados de procesos, y si queremos eliminar un servicio o proceso mediante *kill*, necesitaremos su PID.

Respecto a todo el modelo de arranque, las distribuciones presentan algún pequeño cambio:

- **Fedora/Red Hat** (solo las versiones con soporte SysVinit): el *runlevel* 4 no tiene un uso declarado. Los directorios */etc/rcn.d* existen como enlaces hacia subdirectorios de */etc/rc.d*, donde están centralizados los *scripts* de arranque. Los directorios son, así: */etc/rc.d/rcn.d*; pero como existen los enlaces, es transparente al usuario. El *runlevel* por defecto es el 5 con arranque con *X*.
  Los comandos y ficheros relacionados con el arranque del sistema están en los paquetes de software sysvinit y initscripts.
  Respecto a los cambios de ficheros y guiones en Fedora, cabe destacar: en */etc/sysconfig* podemos encontrar archivos que especifican valores por defecto de la configuración de dispositivos o servicios. El guión */etc/rc.d/rc.sysinit* es invocado una vez cuando el sistema arranca; el guión */etc/rc.d/rc.local* se invoca al final del proceso de carga y sirve para indicar inicializaciones específicas de la máquina que nos interesen sin pasar por todo el sistema siguiente.
  El arranque real de los servicios se hace por medio de los guiones almacenados en */etc/rc.d/init.d*. Hay también un enlace desde */etc/init.d*. Además, Fedora proporciona unos *scripts* de utilidad para manejar servicios: */sbin/service* para parar o iniciar un servicio por el nombre y */sbin/chkconfig* para añadir enlaces a los ficheros *S* y *K* necesarios para un servicio, o la obtención de información sobre los servicios.

- **Debian** dispone de comandos de gestión de los *runlevels* como *update-rc.d*, que permite instalar o borrar servicios arrancándolos o parándolos en uno o más runlevels. Otro comando es *invoke-rc.d*, que permite las clásicas acciones de arrancar, parar o reiniciar el servicio.
  El *runlevel* por defecto en Debian es el 2, el X Window System no se gestiona desde */etc/inittab*, sino que existe el gestor (por ejemplo, *gdm* o *kdm*), como si fuera un servicio más del *runlevel* 2.

3.2. Upstart

Upstart es un sistema de arranque basado en eventos, diseñado para sustituir al proceso comentado generado por el *init* del sistema de arranque *sysvinit*. Uno de los principales problemas del *daemon init* de SystemV es que no fue pensado para hardware moderno, que permite dispositivos removibles, o dispositivos que puedan ser conectados/sustituidos en caliente (*hotplug*).

Nota

Podéis obtener más información respecto de Upstart en "The Upstart Cookbook": http://upstart.ubuntu.com/cookbook.

Este sistema fue diseñado inicialmente por la empresa Canonical, para incorporarse en Ubuntu 6.10 (2006) y ha sido incorporado posteriormente a las diferentes versiones de Ubuntu, y a Fedora (a partir de la versión 9) y OpenSUSE (11.3). Debian mantiene el anterior sistema (sysvinit en la versión estable) con ciertas modificaciones hasta las últimas versiones, pero había de

migrar progresivamente al nuevo sistema (debido a los intentos, por ambas partes, por mantener el máximo nivel de semejanza con Ubuntu y Debian). En el 2014 ambas distribuciones decidieron migrar a systemd de forma progresiva, es de esperar que estas distribuciones vayan adaptando por completo el modelo systemd (con algunos intentos puntuales de mantener al mismo tiempo la compatibilidad con SysV y Upstart). Cabe señalar que Upstart en algunas versiones antiguas de distribuciones GNU/Linux todavía está en uso, y en nuestras tareas de administración nos podemos encontrar con versiones antiguas, pero soportadas (como algunas de las LTS) de Debian y Ubuntu que sigan soportando Upstart.

Aunque cabe destacar que Upstart (al menos de momento) mantiene una compatibilidad total con los *scripts* iniciales del proceso de *init* (vistos en este apartado, "Niveles de arranque y servicios"), lo que se modifica es el proceso interno que gestiona los servicios de arranque, que pasa a estar controlado por el *daemon* de Upstart.

Identificación de tipo de arranque

Podemos detectar la presencia de Upstart, por la existencia (en el listado del comando *ps*), de varios *daemons* denominados con "upstart-" como prefijo.

Las distribuciones con soporte systemd suelen presentar el *daemon systemd* con PID=1, y suelen existir (en ps) varias instancias (hijas) del mismo *daemon*. En algunas distribuciones se mantiene un proceso *init* con PID=1, pero existen diferentes procesos hijo con el *daemon systemd* o variaciones de este.

Los sistemas con sysvinit no presentan ninguno de los anteriores, solamente el proceso *init* con PID=1, y demás servicios son arrancados desde *init*.

El demonio *init* de Upstart está basado en eventos que permiten ejecutar programas (o acciones) específicas a partir de cambios en el sistema, que suelen ser típicamente (respecto al *init* SystemV) *scripts* de parada o arranque de servicios. En System V esto solamente se producía por cambio de nivel de ejecución; en Upstart puede provocarlo cualquier evento que suceda dinámicamente en el sistema. Por ejemplo, Upstart suele comunicarse (por eventos) con demonios (como *udev*) que controlan los cambios del hardware de la máquina, por ejemplo si aparece/desaparece un determinado hardware, pueden activarse o desactivarse los servicios de sistema asociados a él.

La gestión de los trabajos de upstart está centralizada en */etc/init* y, a medida que se vaya migrando (en las distribuciones) hacia upstart, este directorio irá remplazando los contenidos de */etc/init.d* y los directorios asociados a los niveles de ejecución */etc/rc<n>.d*.

En la terminologia de Upstart, un evento es un cambio de estado del sistema que puede ser informado al proceso *init*. Un *job* es un conjunto de instrucciones que *init* lee, típicamente o bien un binario o un *shell script*, junto con el nombre del evento. Así, cuando el evento aparece en el sistema, Upstart lanza el *job* asociado (los *jobs* del sistema podemos encontrarlos definidos en el directorio mencionado */etc/init*). Los *jobs* pueden ser de dos tipos, *task* o *service*.

Una *task* es un *job* que realiza su tarea y retorna a un estado de espera cuando acaba; un *service* no termina por sí mismo, sino por una decisión basada en un evento o bien una intervención manual de parada. En */etc/init* podemos observar los *jobs* definidos en el sistema upstart, y examinar su configuración, a modo de ejemplo. En Upstart también es posible definir *jobs* de usuario, colocándolos en un directorio ".init" dentro de la cuenta del usuario.

Así, el proceso general *init* de Upstart es un funcionamiento de máquina de estados; mantiene control del estado de los trabajos (*jobs*) y los eventos que dinámicamente aparecen, y gestiona los cambios de estado de los trabajos en reacción a los eventos aparecidos.

Hay una serie de trabajos (*jobs*) predefinidos con nombres *rc* que sirven para emular los cambios de *runlevel* de SystemV, para así emular el concepto de nivel de ejecución.

Para examinar el sistema Upstart, podemos usar como herramienta básica el comando *initctl,* que nos permite (entre otras tareas):

- initctl list (listar *jobs* y su estado).
- initctl emit EVENT (emitir manualmente eventos concretos).
- initctl start/stop/status JOB (la acción sobre un *job*). Como abreviaturas existen como comandos: `start`, `stop`, `restart`, `reload`, `status` con respecto al JOB directamente.

Para finalizar, podemos observar que Upstart podrá tener cierta evolución futura, ya sustituyendo completamente al SysVinit inicial o evolucionando a nuevas funcionalidades, pues el sistema basado en máquina de estados es muy completo para integrar diferentes funcionalidades que ahora están repartidas por varios elementos del sistema. Se prevee que podría llegar a reemplazar programación de tareas periódicas de sistema (la usada actualmente, como veremos, por elementos como *cron*, *anacron*, *atd*) y posiblemente gestiones varias de demonios de red (como las realizadas por *inetd* o *xinetd*).

Upstart es un aspecto al que deberemos prestar atención en algunas distribuciones GNU/Linux antiguas que continúen con soporte. Por otra parte, Sysvinit sigue utilizándose en la mayoría de UNIX propietarios, y versiones empresariales de GNU/Linux.

3.3. Systemd

El nuevo sistema de arranque systemd, se está imponiendo recientemente debido a la decisión de Debian y de las distribuciones asociadas (Ubuntu, Mint) de escoger esta opción, así como varias ramas de Fedora (RHEL, y CentOS) también han tomado este camino.

Nota

Sitio oficial systemd: https://freedesktop.org/wiki/Software/systemd/

Systemd, es un *daemon* de gestión del sistema diseñado para Linux, que sustituye al proceso *init* y, por tanto, se coloca como proceso con PID=1 (aunque en algunas distribuciones, se observan *daemons systemd* como hijos del proceso PID=1) en arranque del sistema Linux. También el mismo nombre, systemd, se utiliza para denominar al paquete completo de componentes del sistema, que incluyen el *daemon systemd*.

Systemd se ha desarrollado para Linux como sustituto del sistema *init* heredado de SysV (sysvinit). Igual que *init*, el *daemon systemd*, controla otros *daemons* que se encargan de iniciar, siendo procesos en segundo plano; systemd es el primer *daemon* en arrancar y el último en terminar durante el proceso de apagado.

Los objetivos principales de diseño fueron intentar expresar dependencias entre *daemons* en tiempo de arranque, para así determinar todos los procesos que podían hacerse concurrentemente o en paralelo durante el proceso de arranque, y en general, reducir el coste computacional hacia el *shell*.

Este proceso de integración de systemd también ha recibido bastantes críticas de la comunidad, por ser un intento de aglutinar un gran número de funcionalidades en un único sistema (en sentido contrario a la filosofía UNIX, de unas pequeñas herramientas interconectadas, cada una realizando tareas concretas y bien definidas).

Nota

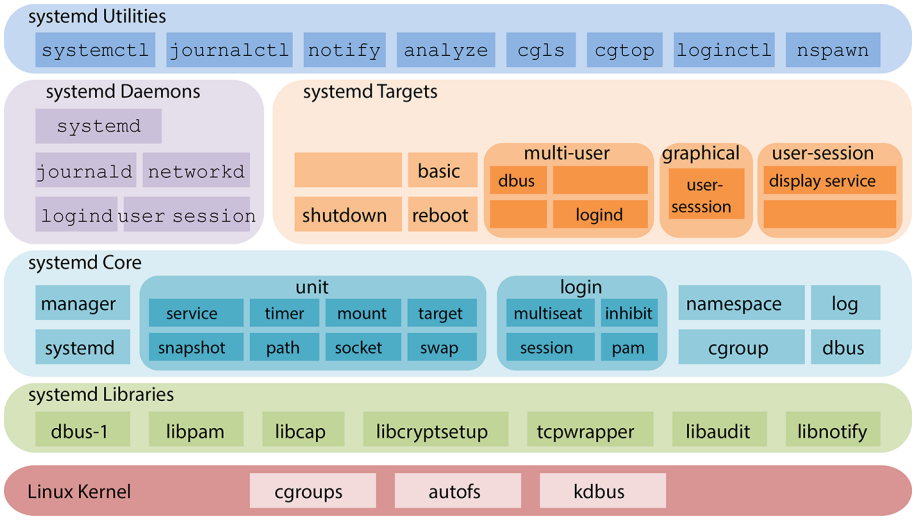
El sitio "Boycott systemd" recogió gran parte de estas críticas a systemd:

http://without-systemd.org/wiki/index.php/Local_copy_of_boycottsystemd.org_archive

Se pueden ver más argumentos y alternativas a systemd en:

http://without-systemd.org/wiki/index.php/Main_Page

En estos momentos, systemd, ha aglutinado u ofrece alternativas a toda una serie de *daemons* clásicos, como *udev*, el sistema de arranque *sysvinit*, *inetd*, *acpid*, *syslog*, o *cron*, entre otros. Además, últimamente se han agregado también componentes de configuración de redes, entre otras, añadiendo aún más componentes y funcionalidades dispersas, a un solo bloque de funcionalidad, todo ello controlado por systemd.



Algunos componentes systemd. Fuente: figura proveniente de la Wikipedia en entrada *systemd* bajo licencia CC.

En systemd podemos distinguir, entre otros, componentes como:

- Ficheros **Unit**: donde se recogen las instrucciones de inicialización para cada *daemon* en un fichero de configuración (denominado "unit file"). Hay diferentes tipos como: *service, socket, device, mount, automount, swap, target, path, timer, snapshot, slice* y *scope*.

- Componentes **core**:
  - *systemd*, el daemon principal del sistema systemd que actua de gestor de servicios para GNU/Linux.

  - *systemctl*, como comando principal para controlar el estado del sistema y del gestor de servicios, así como para gestionar las diferentes *units* del sistema. Veremos a posteriori la gestión de servicios mediante este comando.

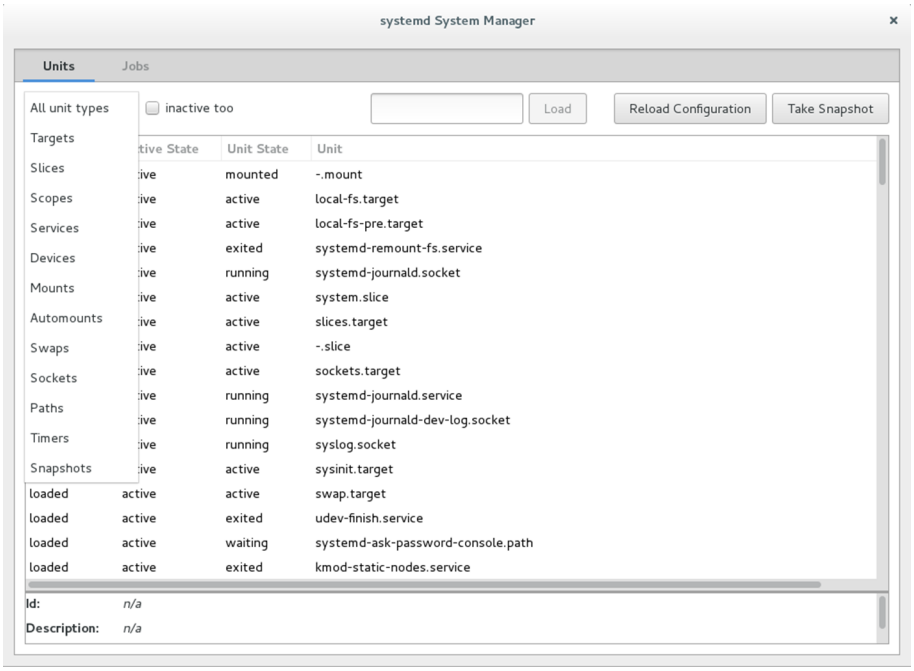
  - *systemd-analize*, para el análisis de las prestaciones del sistema de arranque, y para obtener estadísticas acerca de él. Por ejemplo, con *systemd-analize blame* obtendremos el tiempo en el arranque de cada *unit*. También es muy interesante la opción *plot* donde obtendremos un fichero gráfico (svg) con el diagrama temporal del despliegue de las units a lo largo del arranque.

  - Utilidades como *systemd-cgls* y *systemd-cgtop* permiten seguir el uso de grupos de control por parte de systemd, que los usa (el subsistema *cgroups* del kernel Linux) para controlar el seguimiento de los procesos, en lugar de usar identificadores de procesos (PID). Systemd también usa los *cgroups*, para mecanismos de contenedores Linux, mediante comandos como *systemd-nspawn* y *machinectl*.

- Una serie de **componentes accesorios** (pueden variar dependiendo de la versión actual del sistema systemd), los cuales la mayoría son *daemons* (los finalizados con *d* en la siguiente lista) de la forma *systemd-nombre*:
  - **consoled**: *systemd-consoled* proporciona un *daemon* para la gestión de consolas textuales, con objetivo de substituir a las consolas/terminales virtuales de GNU/Linux.
  - **journald**: *system-journald* es el *daemon* responsable del control de *log* de eventos, gestionada en systemd por ficheros *log* binarios de solo lectura. Es posible gestionar los *logs* dentro de systemd, con *system-journald*, *syslog-ng*, o *rsyslog* dependiendo de la decisión del administrador.
  - **logind**: es un *daemon* encargado del acceso de los usuarios del sistema.
  - **networkd**: es un *daemon* encargado de gestionar configuraciones de los interfaces de red.
  - **timedated**: se encarga de controlar las opciones relacionadas con la hora del sistema, y las zonas horarias.
  - **udevd**: es el *daemon* gestor de dispositivos que se encarga de gestionar dinámicamente el directorio */dev* del sistema, y las acciones de añadir o quitar dispositivos desde usuario, así como control de carga de *firmwares*.
  - **libudev**: en este caso una librería estándar para utilizar *udev*, que permite que aplicaciones de terceros puedan acceder a consultar los recursos de tipo *udev*.
  - **systemd-boot**: un gestor de arranque del sistema (procede de uno existente previamente denominado *gummiboot*), con soporte de *firmare* UEFI. En principio pensado como substituto ligero de GNU Grub.

Aparte también se dispone de algunos *frontends* gráficos para gestionar los servicios, y consultar las units disponibles como *systemd-ui* (para entornos Gnome, también conocido como *systemadm*) y *systems-kcm* (para entornos KDE). Aunque estos entornos no suelen mantenerse al mismo ritmo que las versiones de systemd, que se concentran en las herramientas textuales como *systemctl* y *systemd-analize*.

Systemd introduce el concepto de units, que son representadas por ficheros de configuración, y básicamente representan información sobre un servicio de sistema, sockets de escucha, estados de sistema, u otros objetos relevantes para el arranque.



system-ui (systemadm) mostrando en una distribución Debian parte de las units del sistema, así como el menú de tipos de estas.

Podemos encontrar los ficheros en (por orden inverso de precedencia, si se dan ficheros comunes a diferentes niveles):

- */usr/lib/systemd/system/*. Unidades de sistema (normalmente procedentes de paquetes).

- */run/systemd/system/*. Unidades creadas en tiempo de ejecución.

- */etc/systemd/system/*. Unidades creadas y gestionadas por el administrador.

En cuanto a los tipos de unidades, podemos encontrar (entre otros):

Unidad de	Extensión de fichero	Descripción
Servicio	*.service*	Un servicio de sistema
Target	*.target*	Grupo de unidades systemd
Dispositivo	*.device*	Fichero de dispositivo
Automount point	*.automount*	Un punto de automontaje de un fs
Mount	*.mount*	Un punto de montaje de fs
Snapshot	*.snapshot*	Un estado guardado del gestor systemd
Socket	*.socket*	Un *socket* de comunicación entre procesos
Timer	*.timer*	Un temporizador systemd

Para la gestión de servicios del sistema, se utilizan las unidades de servicio (*.service*), que son usadas para propósitos semejantes a los antiguos ficheros de servicios presentes en */etc/init.d/*. Systemd dispone del comando *systemctl*, que se usa en combinación con las opciones: *start*, *stop*, *restart*, *enable*, *disable*, *status*, *is-enabled* del correspondiente servicio:

Por ejemplo, para parar un servicio, podemos omitir la extensión ".service", ya que es la que asume *systemctl* por defecto. Respecto a otros comandos semejantes de sistemas de arranque anteriores (init o upstart), se mantienen en algunas distribuciones los comandos como *service* o *chkconfig*, estos en una distribución basada en systemd deberían evitarse y usar en lo posible *systemctl*.

Un comando como:

```
# systemctl list-units --type service --all
```

Nos proporciona el *status* de todos los servicios (o unidades de tipo servicio en terminología systemd).

```
# systemctl list-units-files --type service
```

Nos proporciona nombre del servicio con nombre completo e información de si está activo o no. Para un servicio concreto disponemos de:

```
# systemctl status name.service
```

Para preguntar si el servicio está activo:

```
# systemctl is-active name.service
```

Para preguntar si el servicio está habilitado:

```
# systemctl is-enable name.service
```

Con respecto a los niveles de ejecución (*runlevels*), systemd proporciona diferentes unidades equivalentes, en formato *target*; así existen unidades *target* denominadas: *poweroff*, *multi-user*, *graphical*, *reboot*,que se pueden listar mediante:

```
# systemctl list-units --type target
```

```
# systemctl isolate name.target
```

En el segundo caso nos permite cambiar el *target* actual. Y conocer el de por defecto (o colocarlo con *set-default*):

```
# systemctl get-default
```

Para las diferentes opciones de parada y reinicio de máquina, se dispone de las opciones para *systemctl*: *halt*, *poweroff*, *reboot*, *suspend*, *hibernate*.

Una particularidad de systemd es que también puede actuar en máquinas remotas por medio de conexión ssh, por ejemplo, podemos lanzar un comando con:

```
# systemctl --host usuario@host_remoto comando
```

Para investigar errores de diferentes unidades, podemos realizar lo siguiente:

```
# systemctl --state=failed
  Nos proporciona unidades que han provocado fallos
# systemctl status  unidad_con_fallos.extension
  Preguntamos por el estado con el que ya podemos examinar si nos proporciona
  alguna información adicional.
```

También, como veremos después a partir del PID del proceso involucrado (si no lo tenemos podemos intentar un restart), podemos examinar el journal correspondiente con:

```
# journalctl -b _PID=pid_encontrado
```

Systemd tiene muchas posibilidades, dependiendo de las unidades utilizadas y los múltiples componentes de que dispone, algunos de ellos los comentaremos en secciones posteriores.

4. Observar el estado del sistema

Una de las principales tareas del administrador (*root*) en su día a día será verificar el correcto funcionamiento del sistema y vigilar la existencia de posibles errores o de saturación de los recursos de la máquina (memoria, discos, etc.). Pasaremos a detallar, en los siguientes apartados, los métodos básicos para examinar el estado del sistema en un determinado momento y llevar a cabo las acciones necesarias para evitar problemas posteriores.

En el taller final de este módulo, realizaremos un examen de un sistema ejemplo, para que podáis ver algunas de estas técnicas.

4.1. Arranque del sistema

En el arranque de un sistema GNU/Linux se produce todo un volcado de información interesante. Cuando el sistema arranca, suelen aparecer los datos de detección de las características de la máquina, detección de dispositivos, arranque de servicios de sistema, etc., y se mencionan los problemas aparecidos. Además se crean cierto número de sistemas de ficheros virtuales, que generan y permiten manipular diferente cantidad de información del espacio de usuario (*user-space*) y del espacio de *kernel* (*kernel-space*).

En la mayoría de las distribuciones, mucha de esta información de arranque (a modo de eventos) puede verse en la consola del sistema directamente durante el proceso de arranque. Sin embargo, o la velocidad de los mensajes o algunas modernas distribuciones que los ocultan tras carátulas gráficas pueden impedir seguir los mensajes correctamente, con lo que necesitaremos una serie de herramientas para este proceso.

Básicamente, podemos utilizar:

- **Comando dmesg**: da los mensajes del último arranque del *kernel*.

- **Fichero */var/log/messages***: log general del sistema, que contiene los mensajes generados por el *kernel* y otros *daemons* (puede haber multitud de archivos diferentes de log, normalmente en */var/log*, y dependiendo de la configuración del servicio *syslog*).
  En algunas distribuciones modernas, el sistema de logs (*syslog* por defecto en la mayoría), se ha substituido por *rsyslog*, que generalmente tiene configurado el log de sistema principal en */var/log/syslog*. En otras distribuciones que utilizan systemd como sistema de arranque, puede que no veamos ninguno de los anteriores ficheros (*messages* o *syslog*); systemd se encarga también del log de los eventos con un componente/*daemon* denominado

Journald, en este caso se dispone del comando `journalctl`, para obtener en forma textual el listado de eventos del log del sistema.

- **Comando *uptime***: indica cuánto tiempo hace que el sistema está activo.

En tiempo de arranque además, como ya hemos comentado, se generan una serie de sistemas de ficheros virtuales, que nos permiten acceder a información del proceso de *boot*, y al estado dinámico del sistema, tanto del espacio de usuario, como del de *kernel*, y de sus diversos componentes. Podemos detectar estos sistemas montados examinándolos bien con el comando `df`, o con `mount` directamente. Entre los sistemas que encontraremos cabe destacar:

- **Sistema */proc***: pseudosistema de ficheros (*procfs*) que utiliza el *kernel* para almacenar la información de procesos y de sistema.

- **Sistema */sys***: pseudosistema de ficheros (*sysfs*) que apareció con la rama 2.6.x del *kernel*, con objetivo de proporcionar una forma más coherente de acceder a la información de los dispositivos y sus controladores (*drivers*).

- **Sistema /config**: *Configfs*, es similar a *sysfs*, pero en este caso es complementario, ya que permite crear, gestionar y destruir objetos/datos de espacio de *kernel*, desde espacio de usuario. Típicamente lo encontramos montado en */config* o en otras ocasiones en */sys/kernel/config*.

- **Sistema de ficheros tmpfs**: típicamente es usado para montar espacio de almacenamiento temporal (de forma virtual, en memoria volátil, sin almacenamiento en disco). Varias distribuciones lo usan para montar el directorio */run* y sus subdirectorios.

- **Sistema de ficheros devtmpfs**: proporciona la información de los dispositivos, montándose en */dev* en arranque de la máquina. Información, que es después utilizada por el gestor de dispositivos *udev* (a través de su *daemon udevd*). *Udev* forma ahora parte del sistema de arranque systemd.

4.2. Kernel: directorio */proc*

El *kernel*, durante su arranque, pone en funcionamiento sistema de ficheros virtual procfs montándolo en */proc*, donde vuelca la información que recopila de la máquina, así como muchos de sus datos internos, durante la ejecución. El directorio */proc* está implementado sobre memoria y no se guarda en disco. Los datos contenidos son tanto de naturaleza estática como dinámica (varían durante la ejecución).

Hay que tener en cuenta que al ser */proc* fuertemente dependiente del *kernel*, propicia que su estructura dependa del *kernel* de que dispone el sistema y la estructura y los ficheros incluidos pueden cambiar.

Una de las características interesantes es que en el directorio */proc* podremos encontrar las imágenes de los procesos en ejecución, junto con la información que el *kernel* maneja acerca de ellos. Cada proceso del sistema se puede encontrar en el directorio */proc/<pidproceso>,* donde hay un directorio con ficheros que representan su estado. Esta información es básica para programas de depuración, o bien para los propios comandos del sistema, como *ps* o *top,* que pueden utilizarla para ver el estado de los procesos en ejecución. En general, muchas de las utilidades del sistema consultan la información dinámica del sistema desde */proc (*en especial, algunas utilidades proporcionadas en el paquete *procps).*

Nota

El directorio */proc* es un recurso extraordinario para obtener información de bajo nivel sobre el funcionamiento del sistema. Muchos comandos de sistema se apoyan en él para sus tareas.

Por otra parte, en */proc* podemos encontrar otros ficheros de estado global del sistema. Comentamos de forma breve, algunos ficheros que podremos examinar para obtener información importante:

Fichero	Descripción
/proc/bus	Directorio con información de los buses PCI y USB
/proc/cmdline	Línea de arranque del *kernel*
/proc/cpuinfo	Información de la CPU
/proc/devices	Listado de dispositivos del sistema de caracteres o bloques
/proc/driver	Información de algunos módulos *kernel* de hardware
/proc/filesystems	Sistemas de ficheros habilitados en el *kernel*
/proc/scsi	SCSI, en el fichero scsi caracteristicas de discos
/proc/interrups	Mapa de interrupciones hardware (IRQ) utilizadas
/proc/ioports	Puertos de E/S utilizados
/proc/meminfo	Datos del uso de la memoria
/proc/modules	Módulos del *kernel*
/proc/mounts	Sistemas de archivos montados actualmente
/proc/net	Directorio con toda la información de red
/proc/scsi	Directorio de dispositivos SCSI, o IDE emulados por SCSI
/proc/sys	Acceso a parámetros del *kernel* configurables dinámicamente
/proc/version	Versión y fecha del *kernel*

A partir de la rama 2.6 del *kernel*, se inició una transición progresiva de *procfs* (*/proc)* a *sysfs* (*/sys*) con el objetivo de mover toda aquella información que no esté relacionada con procesos, en especial dispositivos y sus controladores (módulos del kernel) hacia el sistema */sys*.

4.3. *Kernel*: */sys*

El sistema Sys se encarga de hacer disponible la información de dispositivos y controladores, información de la cual dispone el *kernel*, al espacio de usuario, de manera que otras API o aplicaciones puedan acceder de una forma flexible a la información de los dispositivos (o sus controladores). Suele ser utilizada por capas como el servicio *udev* para la monitorización y configuración dinámica de los dispositivos.

Dentro del concepto de *sys* existe una estructura de datos en árbol de los dispositivos y controladores (digamos el modelo conceptual fijo), y después se accede a él por medio del sistema de ficheros *sysfs* (cuya estructura puede cambiar entre versiones).

En cuanto se detecta o aparece en el sistema un objeto añadido, en el árbol del modelo de controladores (controladores, dispositivos incluyendo sus diferentes clases), se crea un directorio en *sysfs*. La relación padre/hijo se refleja con subdirectorios bajo */sys/devices/* (reflejando la capa física y sus identificadores). En el subdirectorio */sys/bus* se colocan enlaces simbólicos, reflejando el modo en el que los dispositivos pertenecen a los diferentes buses físicos del sistema. Y en */sys/class* muestra los dispositivos agrupados de acuerdo a su clase, como por ejemplo red, mientras que */sys/block/* contiene los dispositivos de bloques.

Alguna de la información proporcionada por */sys* puede encontrarse también en */proc*, pero se consideró que este estaba mezclando diferentes cosas (dispositivos, procesos, datos hardware, parámetros *kernel*) de forma no coherente, y ello fue uno de los motivos para crear */sys*. Se espera que, progresivamente, se migre información de */proc* a */sys* para centralizar la información de los dispositivos.

4.4. Udev: gestión de dispositivos /dev

A diferencia de los sistemas tradicionales, donde los nodos de dispositivos, presentes en el directorio /dev eran considerados como un conjunto estático de ficheros, el sistema *udev* proporciona dinámicamente los nodos para los dispositivos presentes en el sistema.

Udev es el gestor de dispositivos para el *kernel* Linux, que sustituye a algunas iteraciones anteriores, como los sistemas *hotplug* y *devfsd*, tratando la gestión de nodos de dispositivo en el directorio */dev*, y al mismo tiempo, gestiona los eventos generados en espacio de usuario, debido a la nueva presencia (o ausencia) de dispositivos hardware incluidos en el sistema, incluyéndose la cara

de *firmware* necesaria para ciertos dispositivos (aunque esta última función se desplazará progresivamente al espacio de *kernel*, reservándola a los *drivers* de los dispositivos).

Entre otras características *udev*, soporta:

- Nombrado persistente de dispositivos, por ejemplo, evitando el orden de llegada (o conexión) de los dispositivos al sistema. Entre ellos, se proporcionan nombres persistentes para los dispositivos de almacenamiento. Cada disco reconocido dispone de un id de sistema de ficheros, el nombre del disco, y la localidad física del hardware donde está conectado.

- Notificación a sistemas externos, de los cambios de los dispositivos.

- Creación de un directorio */dev* dinamico.

- Se ejecuta en espacio de usuario, quitando del *kernel* la responsabilidad de nombrar los dispositivos, por tanto, pueden usarse nombrados de dispositivos específicos a partir de las propiedades del dispositivo.

El sistema *udev* está compuesto básicamente de tres partes:

- La librería **libudev**, que permite el acceso a la información del dispositivo. Dependiendo de la implementación, esta librería puede encontrarse aislada, o el caso de distribuciones con arranque systemd, ha pasado a incluirse como una funcionalidad incluida en este sistema.

- Un *daemon* de espacio de usuario, *udevd*, que gestiona el directorio virtual */dev*. El *daemon* escucha el *socket* de comunicación entre *kernel* y espacio de usuario, para determinar cuándo un dispositivo es añadido o quitado del sistema, y *udev* gestiona el resto de operaciones: creación del nodo en */dev*, carga del módulo/*driver* necesario del *kernel*, etc. En sistemas con systemd de arranque, el funcionamiento del *daemon* es controlado por *systemd-udevd*.

- Una colección de utilidades administrativas, *udevadm*, para diagnósticos.

Clásicamente el directorio */dev* en una máquina Linux, es donde deben encontrase todos los ficheros de dispositivo del sistema. A través de un fichero de dispositivo, es como un programa de espacio de usuario, debe acceder a un dispositivo hardware o una función de este. Por ejemplo, el fichero de dispositivo */dev/sda* es tradicionalmente usado para representar el primer disco del sistema. El nombre sda corresponde a un par de números denominados el mayor y menor del dispositivo, y que son utilizados por el *kernel* para determinar con qué dispositivo hardware esta interaccionando. Cada uno de los números mayores y menores son asignados a un nombre que corresponde con el tipo

de dispositivo. Esta correspondencia es mantenida por la autoridad LANANA, y la lista de estas asignaciones puede encontrase en su página web: http://www.lanana.org/docs/device-list/devices-2.6+.txt.

A partir de ciertos desarrollos del *kernel* se comprobó que esta situación no era práctica, debido a la limitación del esquema de números, y a asignar estos números en situaciones dinámicas donde los dispositivos son removibles, o a la gran cantidad de números necesarios. Además, debido a la situación estática de */dev* inicialmente, este era sobrecargado por las distribuciones, por todos aquellos posibles dispositivos que nos podríamos encontrar a lo largo del tiempo, creando una larga lista de entradas en el directorio, que no iban a ser usadas.

Udev, después de unas iteraciones previas con otros intentos de solución, como *hotplug* y *devfs*, se propuso como solución como soporte de los puntos que hemos comentado anteriormente.

Nota

Se recomienda consultar las páginas man, correspondientes a la distribución, para: *udev*, *udevd* (o *systemd-udevd*) y *udevadm*.

Para el nombrado del dispositivo, se realizan una serie de pasos para intentar determinar un nombre único, a través de indicar:

- **Etiqueta** (*label*) o número de serie que lo pueda identificar según la clase de dispositivo.
- **Número del dispositivo** en el bus donde está conectado (por ejemplo, en un bus PCI, su identificador).
- **Topología en el bus**: posición en el bus al que pertenece.
- **Nombre de reemplazo** por si hay coincidencia entre varios dispositivos presentes.
- **Nombre en el *kernel***.

Si los pasos anteriores no pueden proporcionar un nombre, se utiliza finalmente el disponible en el *kernel*.

El sistema *udev*, cuya información suele residir en */lib/udev y /etc/udev* (también es posible que exista un */run/udev* en tiempo de ejecución), entre otras, incluye una serie de reglas para el nombrado y las acciones a tomar sobre los dispositivos, que podemos observar en las distribuciones en:

/lib/udev/rules.d

/etc/udev/rules.d

En el primer caso, son las reglas por defecto incluidas en el sistema, mientras en el segundo, son las que el administrador pueda llegar a incluir como nuevas para identificar algún dispositivo especial, o cambiar su denominación (si existe regla en */etc/udev/rules.d*, esta tiene preferencia sobre la de defecto en el sistema en */lib/udev/rules.d*). El primer número en el nombre de estos ficheros de reglas se refiere al orden de aplicación de las reglas. Y las reglas pueden contener cambios de nombre previstos, o bien actuaciones a llevar a cabo según eventos de añadido de dispositivos, cambios, o extracción de estos.

En general, el funcionamiento del *daemon udevd* gestiona los eventos a través de:

- En proceso de arranque, se monta el directorio */dev* como sistema virtual de archivos.
- *udevd* copia los nodos de dispositivo estáticos que tiene en */lib/udev/devices* en el directorio */udev* (dependiente de la distribución).
- *udevd* queda a la escucha de los eventos del *kernel* (*uevents*), para los dispositivos conectados en el sistema.
- *udevd* pasa los datos *uevents* que le llegan y los intenta corresponder con las reglas especificadas en */etc/udev/rules.d* (si no, con */lib/udev/rules.d*).
- Se crean los nodos de dispositivos y los enlaces según especifican las reglas.
- *udevd* lee las reglas */etc/udev/rules.d/*.rules* (o */lib/udev/*.) y las almacena en la memoria (en algunas distribuciones hay que incorporar ficheros binarios con las reglas precargadas, por ejemplo */lib/udev/hwd.d*).
- *udevd* puede recibir notificaciones. Si las reglas cambian, se encargará de leer los cambios y actualizar la copia de memoria.

Si hay que cargar algun *driver*, *udev* también usa los *modalias* para cargar el *driver* correcto. Estos se encuentran a partir de la información de los módulos que acaban creando (vía *depmod* al instalar nuevos módulos) el fichero correspondiente en */lib/modules/`uname -r`/module.alias*.

Por otra parte, la utilidad *udevadm*, entre otras cosas, nos permite obtener información del dispositivo, con los campos que son usados en las reglas *udev*, de manera que podemos crear nuevas reglas que nos permitan cambiar el comportamiento dependiendo de los eventos del dispositivo. Se obtiene la información mediante el parámetro *info* y su *devpath* (el camino dentro de */sys* para llegar a él). Por ejemplo, para la primera tarjeta de red:

Nota

Algunas referencias sobre la escritura de reglas udev:
https://wiki.debian.org/udev
http://www.reactivated.net/writing_udev_rules.html

```
# udevadm info -a -p /sys/class/net/eth0/
looking at device '/devices/pci0000:00/0000:00:03.0/net/eth0':
```

```
    KERNEL=="eth0"
    SUBSYSTEM=="net"
    DRIVER==""
    ATTR{addr_assign_type}=="0"
    ATTR{addr_len}=="6"
    ATTR{dev_id}=="0x0"
    ATTR{ifalias}==""
    ATTR{iflink}=="2"
    ATTR{ifindex}=="2"
    ATTR{type}=="1"
    ATTR{link_mode}=="0"
    ATTR{address}=="08:00:27:37:1a:ce"
    ATTR{broadcast}=="ff:ff:ff:ff:ff:ff"
    ATTR{carrier}=="1"
    ATTR{speed}=="1000"
    ATTR{duplex}=="full"
    ATTR{dormant}=="0"
    ATTR{operstate}=="up"
    ATTR{mtu}=="1500"
    ATTR{flags}=="0x1003"
    ATTR{tx_queue_len}=="1000"
    ATTR{netdev_group}=="0"

  looking at parent device '/devices/pci0000:00/0000:00:03.0':
    KERNELS=="0000:00:03.0"
    SUBSYSTEMS=="pci"
    DRIVERS=="e1000"
    ATTRS{vendor}=="0x8086"
    ATTRS{device}=="0x100e"
    ATTRS{subsystem_vendor}=="0x8086"
    ATTRS{subsystem_device}=="0x001e"
    ATTRS{class}=="0x020000"
    ATTRS{irq}=="19"
    ATTRS{local_cpus}=="00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000000,
                        00000000,00000000,00000000,00000000,00000000,00000000,00000000,00000001"
    ATTRS{local_cpulist}=="0"
    ATTRS{numa_node}=="-1"
    ATTRS{dma_mask_bits}=="32"
    ATTRS{consistent_dma_mask_bits}=="32"
    ATTRS{enable}=="1"
    ATTRS{broken_parity_status}=="0"
    ATTRS{msi_bus}==""

  looking at parent device '/devices/pci0000:00':
    KERNELS=="pci0000:00"
    SUBSYSTEMS==""
```

```
    DRIVERS==""
```

Donde podemos observar las características del dispositivo, desde el punto final, hasta ascender por el bus pci (hasta el id 0 del PCI, posición del bus donde está conectada la tarjeta).

4.5. Procesos

Los procesos que se encuentren en ejecución en un determinado momento serán, en general, de diferente naturaleza. Podemos encontrar:

- **Procesos de sistema**, ya sean procesos asociados al funcionamiento local de la máquina, *kernel*, o bien procesos (denominados *daemons*) asociados al control de diferentes servicios. Por otro lado pueden ser locales, o de red, debido a que estemos ofreciendo el servicio (actuamos de servidor) o estemos recibiendo los resultados del servicio (actuamos de clientes). La mayoría de estos procesos de sistema aparecerán asociados al usuario *root* (aunque se suelen migrar a pseudousuarios especializados por servicio). Puede haber algunos servicios asociados a otros usuarios de sistema (lp, bin, www, mail, etc.). Estos son pseudousuarios "virtuales", no interactivos, que utiliza el sistema para ejecutar ciertos procesos.

- **Procesos del usuario administrador**: en caso de actuar como *root*, nuestros procesos interactivos o aplicaciones lanzadas también aparecerán como procesos asociados al usuario *root*.

- **Procesos de usuarios del sistema**: asociados a la ejecución de sus aplicaciones, ya sea tareas interactivas en modo texto o en modo gráfico.

Como comandos rápidos y más útiles, podemos utilizar:

- ***ps***: el comando estándar, lista los procesos con sus datos de usuario, tiempo, identificador de proceso y línea de comandos usada. Una de las opciones más utilizada es *ps -ef* (o *-ax*), pero hay muchas opciones disponibles (véase man).

- ***top***: una versión que nos da una lista actualizada a intervalos, monitorizando dinámicamente los cambios. Y nos permite ordenar el listado de procesos por diferentes ítems, como gasto de memoria, de uso CPU, con propósito de obtener un ranking de los procesos que acaparan los recursos. Muy útil para dar indicios en situaciones extremas de saturación de uso de recursos, de la posible fuente de problemas.

- ***kill***: nos permite eliminar procesos del sistema mediante el envío de señales al proceso como, por ejemplo, la de terminación *kill -9 pid_del_proceso* (*9* corresponde a *SIGKILL*), donde indicamos el identificador del proceso. Resulta útil para procesos con comportamiento inestable o programas in-

teractivos que han dejado de responder. Podemos ver una lista de las señales válidas en el sistema con *man 7 signal*.

4.6. Logs del sistema

Tanto el *kernel* como muchos de los *daemons* de servicios, así como diferentes aplicaciones o subsistemas de GNU/Linux, pueden generar mensajes que vayan a parar a ficheros log, ya sea para tener una traza de su funcionamiento, o bien para detectar errores o advertencias de mal funcionamiento o situaciones críticas. Este tipo de logs son imprescindibles, en muchos casos, para las tareas de administración y se suelen emplear bastante tiempo de administración en el procesamiento y análisis de sus contenidos.

Nota

Hay diversos gestores de *logs* dependiendo de la distribución usada, el clásico UNIX y de GNU/Linux en su inicio es *Syslogd*, pero progresivamente se han comenzado a usar los sistemas *rsyslog* y *Journald* (parte de *systemd*) como alternativas, o forma conjunta con *syslogd*.

La mayor parte de los logs se generan en el directorio */var/log*, aunque algunas aplicaciones pueden modificar este comportamiento. La mayoría de logs del propio sistema sí que se encuentran en este directorio.

Un *daemon* particular del sistema (importante) es el *daemon Syslogd* (comentamos después las alternativas *rsyslog* y *journald*), que se encarga de recibir los mensajes que se envían por parte del *kernel* y otros *daemons* de servicios y los envía a un fichero log que se encuentra en */var/log/messages*. Este es el fichero por defecto, pero *Syslogd* es también configurable (en el fichero */etc/syslog.conf*), de manera que se pueden generar otros ficheros dependiendo de la fuente, según el *daemon* que envía el mensaje, y así dirigirlo a un log u a otro (clasificando así por fuente), y/o también clasificar los mensajes por importancia (nivel de prioridad): *alarm, warning, error, critical*, etc.

Nota

El *daemon Syslogd* es el servicio más importante de obtención de información dinámica de la máquina. El proceso de análisis de los logs nos ayuda a entender el funcionamiento, los posibles errores y el rendimiento del sistema.

Dependiendo de la distribución (cabe señalar que *Syslog* es cada vez menos usado, en favor de sus alternativas *rsyslog* y *journald*), puede estar configurado de diferentes modos por defecto, en */var/log* suele generar (por ejemplo) ficheros como: *kern.log, mail.err, mail.info*... que son los logs de diferentes servicios. Podemos examinar la configuración para determinar de dónde provienen los mensajes y en qué ficheros los guarda. Una opción que suele ser útil es la posibilidad de enviar los mensajes a una consola virtual de texto (en */etc/syslog.conf* se especifica para el tipo, o tipos, de mensaje una consola de destino, como */dev/tty8* o */dev/xconsole*), de manera que podremos ir viendo los mensajes a medida que se produzcan. Esto suele ser útil para monitorizar la ejecución del

sistema sin tener que estar mirando los ficheros de log a cada momento. Una modificación simple de este método podría ser introducir, desde un terminal, la instrucción siguiente (para el log general):

```
tail -f /var/log/messages
```

Esta sentencia nos permite dejar el terminal o ventana de terminal, de manera que irán apareciendo los cambios que se produzcan en el fichero.

Ademas, de los *logs* de *syslog* como tal, también otros comandos de sistema nos pueden ayudar en otros ámbitos:

- ***uptime***: tiempo que hace que el sistema está activo. Útil para comprobar que no hay existido algún rearranque del sistema inesperado.
- ***last***: analiza log de entradas/salidas del sistema (*/var/log/wtmp*) de los usuarios, y los arranques del sistema. O *lastlog*, control de la última vez que los usuarios han sido vistos en el sistema (información en */var/log/lastlog*).
- **Varias utilidades para procesamiento combinado de logs**, que emiten resúmenes (o alarmas) de lo sucedido en el sistema, como por ejemplo: *logwatch*, *logcheck*...

Como nuevos sistemas de generación de logs de sistema, hay que destacar *rsyslogd* y *journald* (asociado al *systemd*), que progresivamente están sustituyendo a *syslogd* como gestión de los *logs* del sistema.

En el caso de *rsyslogd*, el fichero de la configuración, lo encontraremos en */etc/rsyslog.conf*, siendo la configuración por defecto suficiente para la mayoría de los casos. Como puede observarse, la mayoría de la configuración es similar a la explicada en *syslogd*, debido a que se mantiene compatibilidad con este, agregando una serie de nuevas prestaciones. Entre ellas la posibilidad de utilizar filtros basados en contexto, mayor riqueza de filtros, y la utilización de TCP como mecanismo de comunicación, lo que nos permite monitorización remota de los *logs* de un sistema (fuera de su segmento de red, previamente solo era posible UDP) o disponer de un sistema con servidor centralizado de *logs* de múltiples máquinas cliente, además de tener soporte nativo para bases de datos MySQL y Postgres, permitiendo almacenar los *logs* directamente en estas bases de datos (entre otras).

Ejemplo

Un ejemplo de servidor centralizado de logs con Rsyslog:

http://tecadmin.net/setup-centralized-logging-server-using-rsyslogd/

En cuanto a la posición de los *logs*, estos se suelen generar como antes preferentemente en */var/log* y algunas distribuciones que usan rsyslog siguen manteniendo */var/log/messages*, mientras otras han cambiado el fichero a */var/log/rsyslog*.

En cuanto a *Journald*, está incluido en el *systemd*, y no sin criticas (sobre todo por su conversión de los anteriores *(r)syslog* basados en ficheros y *logs* texto, a formatos binarios), se está haciendo un eco importante, a medida que las distribuciones van adoptando el sistema de arranque de *systemd*.

Systemd, en este caso, ya no requiere la ejecución de *daemons* de tipo *syslog*. Para la lectura de *logs* solo es necesario utilizar:

```
# journalctl
```

Los *logs* del sistema, en este caso, se están guardando (depende de la distribución) normalmente en */var/log/journal*, y en formato binario, no legible directamente, si no son procesados con *journalctlz*. Se dispone de un fichero de configuración en */etc/systemd/journald.conf* (consultar página man de este último para los parámetros configurables).

Con *journalctl* podemos hacer diferentes consultas, algunas plantillas destacables son:

 # journalctl -b Mensajes del último *boot*.

 # journalctl -b -p err Mensajes de error en el último *boot*.

 # journalctl -since=yesterday Mensajes aparecidos desde ayer (combinar -since -until para un periodo).

 # journalctl -f Quedar pendientes de los próximos mensajes a medida que se generen.

 # journalctl _PID=1 Listar mensajes del proceso con este PID (En este caso, es el mismo systemd), también _UID= o _GID=.

 # journalctl /usr/sbin/cron Mensajes provenientes de un ejecutable concreto.

 # journalctl /dev/sda Mensajes referentes al disco.

 # journalctl --disk-usage Espacio de disco utilizado por los *logs*.

 # journalctl _TRANSPORT=kernel Mensajes procedentes del *kernel* (equivale a -k).

 # journalctl --list-boots Listado de los últimos *boots* de sistema y sus marcas de tiempo.

 # journalctl _SYSTEMD_UNIT=crond.service Mensajes de un servicio *systemd*, o unidad controlada por *systemd*, el siguiente nos permite conocer las unidades disponibles.

 # journalctl -F _SYSTEMD_UNIT Unidades disponibles para *log* (posibles valores de *_SYSTEMD_UNIT* por los que podemos preguntar).

 # journalctl -F <campo> Como el caso anterior pero para el campo concreto (ver *man 7 systemd.journal-fields* para conocer los posibles campos).

Se recomienda consultar las páginas man: *journalctl*, *7 systemd.journal-fields*, para una descripción exhaustiva de los campos y opciones disponibles.

4.7. Memoria

Respecto a la memoria del sistema, deberemos tener en cuenta que disponemos de la memoria física de la propia máquina y de la memoria virtual, que puede ser direccionada por los procesos. Normalmente (a no ser que estemos tratando con servidores empresariales), no dispondremos de cantidades demasiado grandes, de modo que la memoria física será menor que el tamaño de memoria virtual necesario (4GB en sistemas de 32 bits, o en 64 bits un espacio teórico de 16 ExaBytes, 2^{64} bytes, que en los sistemas físicos actuales suele estar limitado a 256 TeraBytes). Esto obligará a utilizar una zona de intercambio (*swap*) sobre disco, para implementar los procesos asociados a memoria virtual, cuando estos sobrepasen el uso de memoria física existente.

Esta zona de intercambio (*swap*) puede implementarse como un fichero en el sistema de archivos, pero es más habitual encontrarla como una partición de intercambio (llamada de *swap*), creada durante la instalación del sistema. En el momento de particionar el disco, se declara como de tipo Linux Swap.

Para examinar la información sobre memoria, tenemos varios métodos y comandos útiles:

- **Fichero */etc/fstab***: aparece la partición *swap* (si existiese). Con un comando *fdisk* podemos averiguar su tamaño (o consultar a */proc/swaps*).
- **Comando *ps***: permite conocer qué procesos tenemos, y con las opciones de porcentaje y memoria usada.
- **Comando *top***: es una versión ps dinámica actualizable por periodos de tiempo. Puede clasificar los procesos según la memoria que se usa o el tiempo de CPU.
- **Comando *free***: informa sobre el estado global de la memoria. Da también el tamaño de la memoria virtual.
- **Comando *vmstat***: informa sobre el estado de la memoria virtual, y el uso que se le da.

- **Algunos paquetes** como *dstat* permiten recoger datos de los diferentes parámetros (memoria, *swap* y otros) a intervalos de tiempo (de forma parecida a *top*).

4.8. Discos y *filesystems*

Examinaremos qué discos tenemos disponibles, cómo están organizados y de qué particiones y sistemas de archivos (*filesystems*) disponemos.Cuando dispongamos de una partición y de un determinado *filesystem* accesible, tendremos que realizar un proceso de montaje para integrarla en el sistema, ya sea explícitamente o bien programada en arranque. En el proceso de montaje, se conecta el sistema de archivos asociado a la partición a un punto del árbol de directorios.

Para conocer los discos (o dispositivos de almacenamiento) que tenemos en el sistema, podemos basarnos en la información de arranque del sistema (comando *dmesg* o */var/log/messages*), donde se detectan los presentes, como los */dev/hdx* para los dispositivos IDE o los SCSI con dispositivos */dev/sdx*. Últimamente los discos SATA y antiguos IDE son presentados como scsi, debido a una capa de emulación del *kernel* que trata a los discos como scsi. Otros dispositivos, como discos duros conectados por USB, discos *flash* (los de tipo *pen drive*), unidades removibles, CD-ROM externos, suelen ser dispositivos con algún tipo de emulación scsi, por lo que también se verán como dispositivo de este tipo.

Cualquier dispositivo de almacenamiento presentará una serie de particiones de su espacio. Típicamente, un disco con el formato clásico MBR soporta un máximo de cuatro particiones físicas, o más si estas son lógicas (permiten colocar varias particiones de este tipo sobre una física). Cada partición puede contener tipos de *filesystems* diferentes, ya sean de un mismo operativo o de operativos diferentes.

El formato clásico de particiones, guardado en la estructura MBR (*Master Boot Record*), fue introducido en 1983 y es un tipo especial de sector de arranque (512 o mas bytes en el primer sector del disco), que guarda como están organizadas las particiones del disco, además de contener código que permite desde el disco arrancar en sistema operativo residente en alguna de las particiones, a partir de petición de la BIOS. Pero MBR, tiene algunas limitaciones importantes, como la de solo 4 particiones físicas (expandible por lógicas), y una limitación de capacidad del disco a 2TB. Debido a estas limitaciones, el esquema de particionado de los discos actuales se está transfiriendo al nuevo esquema GPT (*GUID Partition Table*) superando estas limitaciones. Los reemplazos para la BIOS de PC-Compatibles como EFI i UEFI tienen soporte nativo para GPT, además de permitir el acceso a discos de tamaños superiores a 2TB. Aunque cabe señalar que durante la transicion se han realizado diversas

modificaciones al esquema de MBR para permitir cierta compatibilidad entre los dos esquemas. Y algunas BIOS de algunos fabricantes también disponen de soporte GPT.

GPT es parte de la especificación UEFI (*Unified Extensible Firmware Interface*), para substituir a las antiguas BIOS PC, este esquema de particionado de discos, es usado para el arranque en sistemas operativos con soporte EFI, UEFI, como por ejemplo OS X y windows, pero en el caso de Linux y algunos BSD, pueden arrancarse de GPT, con el soporte de EFI/UEFI o sin él, en BIOS antiguas. En el caso de Linux, versiones superiores a por ejemplo Fedora 8, Ubuntu 8, soportan ya GPT, y proporcionar herramientas como gdisk, GNU Parted, fdisk, y el arranque de Grub 2 con soporte GPT incluido. En el caso de *fdisk*, el soporte no es completo y se recomienda usar *GNU parted* para un soporte completo de discos con GPT.

Volviendo, a nuestro analisis de los discos, para conocer su estructura, o cambiar la disposición del particionamiento del disco, podemos utilizar el comando *fdisk* (en discos MBR, en GPT el soporte es parcial y se recomienda *GNU gparted*), o cualquiera de las variantes de *fdisk*, más o menos interactivas (como *cfdisk*, *sfdisk*). Por ejemplo al examinar un disco (antiguo) de de tipo *IDE /dev/hda*, nos da la siguiente información:

```
# fdisk -l /dev/hda
Disk /dev/hda: 20.5 GB, 20520493056 bytes 255 heads, 63 sectors/track, 2494 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Device  Boot Start End   Blocks Id System
/dev/hda1    *     1     1305   10482381   7 HPFS/NTFS
/dev/hda2    *     1306  2429   9028530    83 Linux
/dev/hda3          2430  2494   522112+    82 Linux swap
```

Disco MBR de 20 GB con tres particiones (se identifican con el número añadido al nombre del dispositivo), donde observamos dos particiones con arranque (columna *Boot* con *****) de tipo NTFS y Linux, lo que supone la existencia de un Windows NT/2000/XP/Vista/7 junto con una distribución GNU/Linux, y la última partición que es usada de *swap* para Linux. Además, tenemos información de la estructura del disco y de los tamaños de cada partición. También se puede obtener información de todas las particiones presentes en el sistema consultado el fichero */proc/partitions*.

En este otro ejemplo observamos un disco SATA, */dev/sda* en otro sistema:

```
$ fdisk -l /dev/sda


Disco /dev/sda: 500.1 GB, 500107862016 bytes
255 cabezas, 63 sectores/pista, 60801 cilindros, 976773168 sectores en total
Unidades = sectores de 1 * 512 = 512 bytes
Tamaño de sector (lógico / físico): 512 bytes / 512 bytes
```

```
Tamaño E/S (mínimo/óptimo): 512 bytes / 512 bytes
Identificador del disco: 0x000aba55

Dispositivo Inicio    Comienzo      Fin      Bloques  Id  Sistema
/dev/sda1   *        2048   960016383   480007168   83  Linux
/dev/sda2       960018430   976771071     8376321    5  Extendida
/dev/sda5       960018432   976771071     8376320   82  Linux swap
```

En este disco de 500GB podemos observar tres particiones (sda1, sda2, sda5) de las cuales una es lógica extendida (sda2) que en realidad contiene la sda5. Por tanto, tenemos dos particiones finales: la sda1 de tipo *boot* que arranca Linux y la sda5 que es utilizada de *swap*. La extendida (en este caso, sda2) nos permitirá, en el caso MBR, poder crear particiones adicionales (si disponemos de espacio en el disco), como por ejemplo, sda6, sda7, sda8, dentro de la extendida sda2, superando así el limite de 4 particiones físicas, con que nos encontramos en discos MBR.

En un sistema con discos con particiones GPT:

```
# fdisk -l /dev/sdb

WARNING: GPT (GUID Partition Table) detected on '/dev/sdb'! The util fdisk doesn't support GPT.
                  Use GNU Parted.

Disco /dev/sdb: 5999.5 GB, 5999532441600 bytes
255 heads, 63 sectors/track, 729401 cylinders
Units = cilindros of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

Disposit. Inicio    Comienzo      Fin      Bloques  Id  Sistema
/dev/sdb1               1      267350  2147483647+  ee  GPT

# parted
GNU Parted 2.1
Usando /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.

(parted) select /dev/sdb
Usando /dev/sdb

(parted) print
Model: DELL PERC 6/i (scsi)
Disk /dev/sdb: 6000GB
Sector size (logical/physical): 512B/512B
Partition Table: gpt
```

```
Numero  Inicio  Fin     Tamaño  Sistema de ficheros  Nombre  Banderas
 1      1049kB  6000GB  6000GB  ext4
```

En este sistema, se dispone de un *storage*, que se ve como un dispositivo */dev/sdb* con un total de 6 TeraBytes disponibles, en una partición */dev/sdb1* que es de tipo GPT, *fdisk* nos avisa de que no dispone de soporte de GPT y que usemos alternativamente *GNU parted*. En la segunda parte se invoca a *parted*, como herramienta de gestión de GPT; esta herramienta es interactiva (el comando *help* mostrará las opciones disponibles), seleccionamos el dispositivo que nos interesa (*select*) y visualizamos las particiones (*print*).

De los discos y particiones de que dispongamos, algunos se encontrarán montados en nuestro sistema de ficheros, o estarán preparados para montarse bajo demanda o bien montarse en el momento en que se disponga de medio (en el caso de dispositivos extraíbles).

Esta información la podemos obtener de diferentes maneras (lo veremos con más detalle en el taller final):

- **Fichero */etc/fstab***: indica dispositivos que están preparados para montarse en el arranque o los extraíbles que podrán ser montados. No tienen por qué estar todos los del sistema, sino solo aquellos que queramos tener en arranque. Los demás podemos montarlos bajo demanda con el comando *mount*, o desmontarlos con *umount*.

- **Comando *mount***: nos informa de los *filesystems* montados en ese momento (ya sean dispositivos reales o *filesystems* virtuales, como */proc*). Podemos obtener esta información también desde el fichero */etc/mtab*.

- **Comando *df -k***: nos informa de los *filesystems* de almacenamiento, y nos permite verificar el espacio usado y disponible. Se trata de un comando básico para controlar el espacio de disco disponible. Respecto a este comando *df -k*, una de nuestras tareas básicas de administración de la máquina es controlar los recursos de esta, y en este caso el espacio disponible en los *filesystems* utilizados. Estos tamaños hay que monitorizarlos con cierta frecuencia para evitar la caída del sistema; nunca tendría que dejarse un *filesystem* (sobre todo si es el */*) por debajo de un 10-15%, ya que hay muchos procesos (*daemons* de servicios) que están escribiendo información temporal o logs, que pueden consumir gran espacio. Un caso particular lo forman los ficheros *core,* generados por fallos de software y que continen información de depuración de los errores que los han provocado junto con la imagen del proceso, los cuales pueden suponer (dependiendo del proceso) tamaños muy grandes de archivo. Normalmente, habrá que tener algunas precauciones de "limpieza del sistema" si se detectan situaciones de saturación del *filesystem*:

- **Eliminar temporales antiguos.** Los directorios */tmp* y */var/tmp* suelen acumular muchos archivos generados por diferentes usuarios o aplicaciones. Algunos sistemas o distribuciones ya toman medidas de limpieza, como limpiar */tmp* en cada arranque del sistema.

- **Logs:** evitar su crecimiento excesivo, ya que según la configuración del sistema (por ejemplo, de *Syslogd*) la información generada de mensajes puede ser muy grande. Habrá que limpiar periódicamente al llegar a determinados tamaños, y en todo caso, si necesitamos la información para posteriores análisis, podemos realizar *backups* en medios extraíbles. Este proceso puede automatizarse mediante uso de *scripts cron*, o bien por medio de herramientas especializadas como *logrotate*.

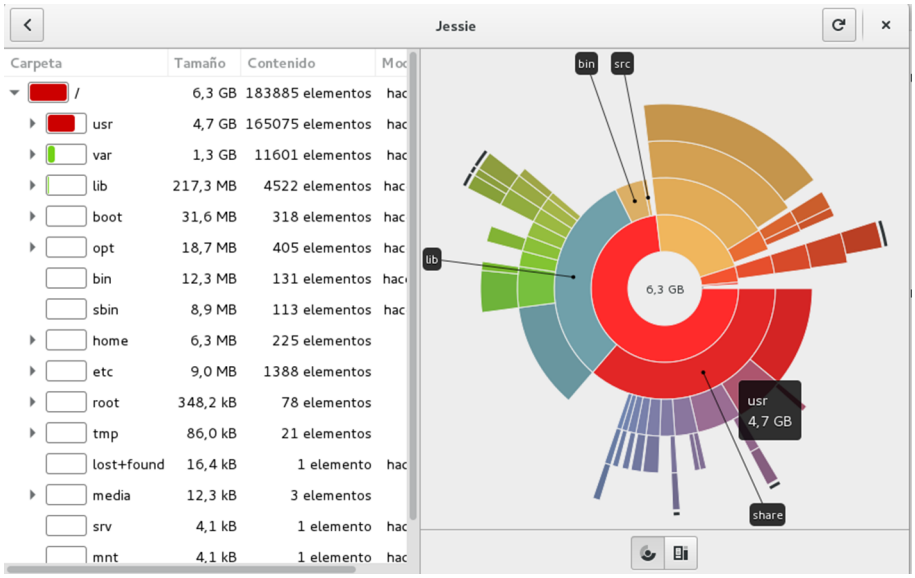
- Hay otros puntos del sistema que suelen crecer mucho, como pueden ser:
  - Ficheros *core* de los usuarios: podemos eliminarlos periódicamente o eliminar su generación.

  - El sistema de correo electrónico: almacena todos los correos enviados y recibidos; podemos pedir a los usuarios que hagan limpieza periódica, o bien poner sistemas de cuotas.

  - Las cachés de los navegadores u otras aplicaciones: también suelen tener tamaños grandes, otra limpieza que habrá que hacer periódicamente.

  - Las cuentas de los propios usuarios: pueden tener cuotas para no exceder los tamaños prefijados...

- Aparte de comandos (como `du`) para el análisis de espacio de los sistemas de ficheros, también existen diferentes herramientas para el análisis gráfico de la ocupación de los ficheros en los sistemas de archivos. Estas herramientas permiten una visión global del administrador, al integrar información tanto local como remota (integrando en la visualización sistemas de archivos por NFS, por ejemplo). Algunas herramientas que destacan para esta funcionalidad: *kdirstat* (o su substituta *k4dirstat*), *baobab*, *gdmap*, entre otras.



Análisis del *filesystem* raiz (/) en una distribución Debian con baobab

4.9. Upower, Udisks

Como casos especiales de control de dispositivos, semejantes a *udev*, pero no integrados en este, están los casos de gestión de dispositivos relacionados con energía, y los discos.

Estos dos componentes están integrados en la iniciativa freedesktop.org, que entre sus objetivos está el desarrollo de tecnologías interoperabales para compartir bases para desarrollar entornos de escritorio, para X Windows, en entornos de GNU/Linux y otros operativos de tipo UNIX. La idea es que hay muchos *frameworks* de desarrollo para X, pero el objetivo es intentar que las diferencias de desarrollo no sean visibles al usuario, con tal de utilizar una serie de tecnologías base comunes.

freedesktop.org

Podéis obtener más información sobre freedesktop.org, sus proyectos y especificaciones en:

http://www.freedesktop.org/wiki/Software/.

http://www.freedesktop.org/wiki/Specifications/.

Por lo que respecta al proyecto DeviceKit, podéis encontrar información en http://www.freedesktop.org/wiki/Software/DeviceKit.

Diversos proyectos de entornos de escritorio, como GNOME, KDE y Xfce, están colaborando con freedesktop.org (que también se conoce como X Desktop Group o XDG).

Asimismo, uno de los proyectos importantes que mantienen para GNU/Linux es D-bus, que es un mecanismo de comunicación interproceso (IPC), que permite a procesos que se ejecutan concurrentemente comunicarse entre ellos para intercambiar información o pedirse servicios. En GNU-Linux, es utilizado para comunicar aplicaciones de escritorio en ejecución en la misma sesión de

escritorio, así como comunicar la sesión de escritorio con el sistema operativo, sea el *kernel*, o *daemons* o procesos propios. D-Bus es utilizado en aplicaciones de escritorio de KDE, Gnome y Xfce. D-Bus es básicamente un sistema de *bus* para intercambiar mensajes, de manera que dos aplicaciones, o dos procesos pueden intercambiar mensajes una con otra, a través del *daemon* de paso de mensajes. En *systemd*, se reescribió el código de D-Bus y se hizo más eficiente, aumentando las prestaciones. También existe paralelamente un proyecto semejante denominado kdbus (*Linux Kernel D-Bus implementation*), que ofrecerá una alternativa en el futuro, mediante comunicaciones peer-to-peer para implementar IPC a través de mediación del *kernel*.

Información sobre D-Bus

D-Bus tutorial: http://dbus.freedesktop.org/doc/dbus-tutorial.html.

Howto use dbus-monitor/: https://wiki.ubuntu.com/DebuggingDBus.

La arquitectura D-Bus está compuesta principalmente de 3 capas:

- **Libdbus**. Una librería que permite a dos aplicaciones conectarse la una a la otra e intercambiar mensajes.

- **dbus-daemon**. Un *daemon* ejecutable que implementa el *bus* de mensajes y que, basándose en *libdbus*, permite que múltiples aplicaciones se conecten encaminando los mensajes de una aplicación a cero o más aplicaciones destinatarias, mediante mecanismos *publish/subscribe*.

- **Librerías construidas para *frameworks* de aplicaciones específicos**. Desde el punto de vista de administrador, desarrollador de aplicaciones, es útil la herramienta *dbus-monitor* que permite ver los mensajes que circulan por el *bus* de mensajes tanto en el caso de sistema como de sesión de escritorio.

En uno de los proyectos mantenidos por freedesktop.org, DeviceKit, entre otros se desarrollaron Upower y Udisks como interfaces y servicios, a través de D-Bus, para manejar la gestión de energía y los dispositivos de almacenamiento.

Upower nos permite enumerar dispositivos de energía (AC o baterías), escuchando por los eventos del dispositivo, y preguntar por su histórico o estadísticas. Algunos ejemplos:

- **# upower -e** Enumera dispositivos de batería o AC.

- **# upower -d** Proporciona información de batería.

- **# upower -i <nombre_dispositivo>** Proporciona información específica de un dispositivo (obtenido de -e).

Por ejemplo:

```
# upower -e
/org/freedesktop/UPower/devices/line_power_AC
```

```
# upower -d
Device: /org/freedesktop/UPower/devices/line_power_AC
  native-path:          AC
  power supply:         yes
  updated:              dom 20 jul 2014 17:57:59 CEST (23889 seconds ago)
  has history:          no
  has statistics:       no
  line-power
    online:             yes

Daemon:
  daemon-version:  0.9.23
  on-battery:      no
  on-low-battery:  no
  lid-is-closed:   no
  lid-is-present:  no
  is-docked:       no

# upower -i /org/freedesktop/UPower/devices/line_power_AC
  native-path:          AC
  power supply:         yes
  updated:              dom 20 jul 2014 17:57:59 CEST (23905 seconds ago)
  has history:          no
  has statistics:       no
  line-power
    online:             yes
```

En este caso encontramos una línea de AC, su estado y desde cuándo se encuentra activa; en casos de baterías, es posible hacer un seguimiento de su carga o descarga a partir de los eventos de energía que se generan, con:

```
# upower --monitor-detail
```

o directamente *-i <nombre_bateria>* para conocer las estadísticas generales.

Por otra parte udisks, se implementa un *daemon*, *udisksd*, que implementa interfaces D-Bus, que pueden ser usadas para la consulta y manipulación de los dispositivos de almacenamiento. Y con la utilidad de línea de comandos, *udisksctl* o *udisks* (dependiendo de la distribución, o incluso puede disponer de los dos), puede ser utilizada para preguntar y utilizar el *daemon* para consultar los discos, por ejemplo, con *udiskctl*, que es la herramienta de línea de comandos para interactuar con el proceso de *daemon* de *udisksd*.

Nota

Véase el manual de referencia Udisks en: https://udisks.freedesktop.org/docs/latest/. Si el comando `udisks` no está disponible puede usarse `udiskctl`: https://udisks.freedesktop.org/docs/latest/udisksctl.1.html.

Por otra parte, udisks como utilidad de línea de comandos si está presente (o sus equivalentes en *udiskctl*):

- **# udisks --enumerate** Lista id de los dispositivos presentes (como alternativa, *udiskctl status*, pero solo obtiene información de primer nivel, no particiones por ejemplo, en este caso *udiskctl dump* obtiene información más detallada).

- **# udisks --enumerate-device-files** Lista los dispositivos y sus identificadores conocidos (como los UUID entre otros; como alternativa, *udiskctl dump*).

- **# udisks --dump** Muestra información extensa de todos los dispositivos presentes (como alternativa *udiskctl dump*).

- **# udisks --show-info** Muestra información de un dispositivo particular (como alternativa *udiskctl -b dispositivo info*).

- **# udisks --monitor-detail** Monitoriza con detalle la actividad del *daemon* de discos (*udisksd*) (como alternativa *udiskctl monitor*).

Un ejemplo de una sesión donde se analiza */dev/sda* y */dev/sda1* de una determinada máquina, previa enumeración de dispositivos; además, en el caso del disco, se obtiene su información SMART, que nos puede ser útil para conocer la "salud" de su estado físico actual:

```
# udisks --enumerate
/org/freedesktop/UDisks/devices/sda5
/org/freedesktop/UDisks/devices/sr0
/org/freedesktop/UDisks/devices/sda
/org/freedesktop/UDisks/devices/sda1
/org/freedesktop/UDisks/devices/sda2

# udisks --enumerate-device-files
/dev/sda5
/dev/disk/by-id/ata-MB0500EBNCR_WMAP146173-part5
/dev/disk/by-id/scsi-SATA_MB0500EBNCR_WMAP146173-part5
/dev/disk/by-id/wwn-0x50014ee002c9780b-part5
/dev/disk/by-uuid/10628c07-51f6-4bb6-babe-082526be81d7
/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0-part5
/dev/sr0
/dev/disk/by-id/ata-DV-18S-A_11022334083404
/dev/disk/by-path/pci-0000:00:1f.5-scsi-0:0:0:0
/dev/sda
/dev/disk/by-id/ata-MB0500EBNCR_WMAP146173
/dev/disk/by-id/scsi-SATA_MB0500EBNCR_WMAP146173
/dev/disk/by-id/wwn-0x50014ee002c9780b
/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0
/dev/sda1
/dev/disk/by-id/ata-MB0500EBNCR_WMAP146173-part1
```

```
/dev/disk/by-id/scsi-SATA_MB0500EBNCR_WMAP146173-part1
/dev/disk/by-id/wwn-0x50014ee002c9780b-part1
/dev/disk/by-uuid/b16ebe2f-5072-4e00-a1e0-19aebf2be5e8
/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0-part1
/dev/sda2
/dev/disk/by-id/ata-MB0500EBNCR_WMAP146173-part2
/dev/disk/by-id/scsi-SATA_MB0500EBNCR_WMAP146173-part2
/dev/disk/by-id/wwn-0x50014ee002c9780b-part2
/dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0-part2

# udisks --show-info /dev/sda
Showing information for /org/freedesktop/UDisks/devices/sda
 native-path:            /sys/devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0/block/sda
  device:                     8:0
  device-file:                /dev/sda
    presentation:             /dev/sda
    by-id:                    /dev/disk/by-id/ata-MB0500EBNCR_WMAP14612773
    by-id:                    /dev/disk/by-id/scsi-SATA_MB0500EBNCR_WMAP1461773
    by-id:                    /dev/disk/by-id/wwn-0x50014ee002c9780b
    by-path:                  /dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0
  detected at:                lun 21 jul 2014 00:51:05 CEST
  system internal:            1
  removable:                  0
  has media:                  1 (detected at lun 21 jul 2014 00:51:05 CEST)
    detects change:           0
    detection by polling:     0
    detection inhibitable:    0
    detection inhibited:      0
  is read only:               0
  is mounted:                 0
  mount paths:
  mounted by uid:             0
  presentation hide:          0
  presentation nopolicy:      0
  presentation name:
  presentation icon:
  automount hint:
  size:                       500107862016
  block size:                 512
  job underway:               no
  usage:
  type:
  version:
  uuid:
  label:
  partition table:
    scheme:                   mbr
```

```
    count:                    3
  drive:
    vendor:                   ATA
    model:                    MB0500EBNCR
    revision:                 HPG0
    serial:                   WMAP146173
    WWN:                      50014ee002c9780b
    detachable:               0
    can spindown:             1
    rotational media:         Yes, at 7200 RPM
    write-cache:              disabled
    ejectable:                0
    adapter:                  /org/freedesktop/UDisks/adapters/0000_3a00_3a1f_2e2
    ports:
      /org/freedesktop/UDisks/adapters/0000_3a00_3a1f_2e2/host0
    similar devices:
    media:
      compat:
    interface:                ata
    if speed:                 (unknown)
    ATA SMART:                Updated at lun 21 jul 2014 01:21:05 CEST
      overall assessment:     Good
================================================================================
 Attribute       Current|Worst|Threshold  Status   Value       Type     Updates
================================================================================
 raw-read-error-rate         200|200| 51   good    0           Pre-fail Online
 spin-up-time                100|253| 21   good    0           Pre-fail Online
 start-stop-count            100|100|  0    n/a    6           Old-age  Online
 reallocated-sector-count    200|200|140   good    0 sectors   Pre-fail Online
 seek-error-rate             200|200| 51   good    0           Pre-fail Online
 power-on-hours               97| 95|  0    n/a    120,7 days  Old-age  Online
 spin-retry-count            100|253| 51   good    0           Pre-fail Online
 calibration-retry-count     100|253| 51   good    0           Pre-fail Online
 power-cycle-count           100|100|  0    n/a    5           Old-age  Online
 unused-reserved-blocks      200|200|100   good    0           Pre-fail Online
 end-to-end-error            100|100| 97   good    0           Pre-fail Online
 reported-uncorrect          100|100|  0    n/a    0 sectors   Old-age  Online
 command-timeout             100|100|  0    n/a    0           Old-age  Online
 airflow-temperature-celsius  70| 61| 45   good    30C / 86F   Old-age  Online
 power-off-retract-count     200|200|  0    n/a    4           Old-age  Online
 load-cycle-count            200|200|  0    n/a    1           Old-age  Online
 temperature-celsius-2       113|104|  0    n/a    30C / 86F   Old-age  Online
 hardware-ecc-recovered      200|200|  0    n/a    0           Old-age  Online
 reallocated-event-count     200|200|  0    n/a    0           Old-age  Online
 current-pending-sector      200|200|  0    n/a    0 sectors   Old-age  Online
 offline-uncorrectable       100|253|  0    n/a    0 sectors   Old-age  Offline
 udma-crc-error-count        200|200|  0    n/a    0           Old-age  Online
```

```
 multi-zone-error-rate       100|253|  0    n/a    0          Old-age  Offline

# udisks --show-info /dev/sda1
Showing information for /org/freedesktop/UDisks/devices/sda1
  native-path:                /sys/devices/pci0000:00/0000:00:1f.2/host0/target0:0:0/0:0:0:0/block
                                     /sda/sda1
  device:                     8:1
  device-file:                /dev/sda1
    presentation:             /dev/sda1
    by-id:                    /dev/disk/by-id/ata-MB0500EBNCR_WMAP146173-part1
    by-id:                    /dev/disk/by-id/scsi-SATA_MB0500EBNCR_WMAP146173-part1
    by-id:                    /dev/disk/by-id/wwn-0x50014ee002c9780b-part1
    by-id:                    /dev/disk/by-uuid/b16ebe2f-5072-4e00-a1e0-19aebf2be5e8
    by-path:                  /dev/disk/by-path/pci-0000:00:1f.2-scsi-0:0:0:0-part1
  detected at:                lun 21 jul 2014 00:51:05 CEST
  system internal:            1
  removable:                  0
  has media:                  1 (detected at lun 21 jul 2014 00:51:05 CEST)
    detects change:           0
    detection by polling:     0
    detection inhibitable:    0
    detection inhibited:      0
  is read only:               0
  is mounted:                 1
  mount paths:            /
  mounted by uid:             0
  presentation hide:          0
  presentation nopolicy:      0
  presentation name:
  presentation icon:
  automount hint:
  size:                       491527340032
  block size:                 512
  job underway:               no
  usage:                      filesystem
  type:                       ext4
  version:                    1.0
  uuid:                       b16ebe2f-5072-4e00-a1e0-19aebf2be5e8
  label:
  partition:
    part of:                  /org/freedesktop/UDisks/devices/sda
    scheme:                   mbr
    number:                   1
    type:                     0x83
    flags:                    boot
    offset:                   1048576
    alignment offset:         0
```

```
        size:                   491527340032
```

5. Sistema de ficheros

En cada máquina con un sistema GNU/Linux podemos ver sistemas de ficheros de diferentes tipos [Hin] [Soy12]. Para empezar, es habitual encontrarse con los propios sistemas de ficheros Linux creados en distintas particiones de los discos [Koe].

La configuración habitual suele ser de dos particiones:

1) la correspondiente a "/" (*root filesystem*) y

2) la correspondiente al fichero de intercambio o de *swap*.

Con todo, en configuraciones más profesionales suele ser habitual separar particiones con partes "diferenciadas" del sistema. Una técnica habitual es, por ejemplo (veremos más opciones después), crear particiones diferentes para:

```
/    /boot /home      /opt  /tmp  /usr  /var  swap
```

que seguramente se encontrarán montadas desde diferentes orígenes (diferentes discos, o incluso red en algunos casos).

Las particiones se realizan para separar claramente partes estáticas y dinámicas del sistema, para permitir de una forma más fácil, ante problemas de saturación, extender las particiones o aislar más fácilmente partes para la realización de *backups* (por ejemplo, las cuentas de los usuarios en la partición */home*).

El tipo de particiones *swap* es de tipo *Linux swap*, y la correspondiente a */* suele ser de alguno de los sistemas de ficheros estándar, ya sea *ext2* (el tipo por defecto hasta los *kernels* 2.4), *ext3* o el nuevo *ext4*, que son mejoras del *ext2* compatibles pero con *journaling*, lo que permite tener un log de lo que va pasando al sistema de ficheros, para recuperaciones más rápidas en caso de error. También pueden ser habituales otros sistemas de archivos, como Reiser (en desuso por falta de soporte), XFS, y algunos más extendidos con soporte importante como Btrfs y ZFS.

Otra configuración habitual puede ser de tres particiones: */*, *swap*, */home*, donde la */home* se dedicará a las cuentas de los usuarios. Esto permite separar las cuentas de los usuarios del sistema, aislando en dos particiones separadas, y podemos dar el espacio necesario para las cuentas en otra partición.

Otro esquema muy utilizado es el de separar en particiones las partes estáticas del sistema de las dinámicas, por ejemplo, una partición donde se coloca / con la parte estática (*/bin /sbin* y */usr* en algunos casos) que se espera que no va a crecer o lo va a hacer muy poco, y otra o varias con la parte dinámica (*/var /tmp /opt*), suponiendo que */opt*, por ejemplo, es el punto de instalación del software nuevo. Esto permite ajustar mejor el espacio de disco y dejar más espacio para las partes del sistema que lo necesiten.

Respecto a los sistemas de ficheros soportados debemos destacar la gran variedad de ellos; actualmente, podemos encontrar (entre otros):

- **Sistemas asociados a GNU/Linux**, como el estándar *ext2*, *ext3*, evolución del anterior con concepto de *journaling* (soporte de log de operaciones realizadas en el sistema de fichero que puede permitir su recuperación en caso de algún desastre que lo haga inconsistente). En las nuevas distribuciones tienen un peso importante *ext4*, como evolución de *ext3* (con mejoras en las prestaciones), y *btrfs* un nuevo diseño (aportando mayor tamaño de ficheros, y prestaciones adicionales en tolerancia a fallos y reparación), que se espera que sea el sistema de ficheros por defecto en la mayoría de ellas.

- **Compatibilidad con entornos no GNU/Linux**: *msdos*, *vfat*, *ntfs*, acceso a los diferentes sistemas de *fat16*, *fat32* y *ntfs*. En particular, cabe resaltar que el soporte *kernel*, en distribuciones antiguas, en el caso del *kernel* ntfs, estaba limitado a lectura. Pero como ya hemos dicho, existen soluciones en espacio de usuario (mediante FUSE, un componente que permite gestionar sistemas de ficheros en espacio de usuario), que la permiten, como el ya mencionado *ntfs-3g*. También se disponen de compatibilidad a otros entornos como Mac con *hfs* y *hfsplus*.

- **Sistemas asociados a soportes físicos**, caso de CD/DVD, como los *iso9660* y *udf*.

- **Sistemas usados en diferentes Unix**, que ofrecen generalmente mejor rendimiento (a veces, a costa de mayor consumo de recursos, en CPU por ejemplo), como JFS2 (IBM), XFS (SGI), ReiserFS, Zfs (Oracle Solaris).

- **Sistemas de ficheros en red** (más tradicionales): NFS, Samba (*smbfs*, *cifs*), permiten acceder a sistemas de ficheros disponibles en otras máquinas de forma transparente por red.

- **Sistemas distribuidos en red**: como GFS, Lustre, Ceph, HDFS o Coda.

- **Pseudosistemas de ficheros**, como procfs (*/proc*) o sysfs (*/sys*).

En la mayoría (excepto algún caso especial) de estos sistemas de ficheros, GNU/Linux nos permitirá crear particiones de estos tipos, construir el sistema de ficheros del tipo requerido y montarlas como parte integrante del árbol de directorios, ya sea de forma temporal o permanente.

5.1. Puntos de montaje

Aparte del *filesystem* principal / y de sus posibles divisiones en particiones extras (*/usr /var /tmp /home*), cabe tener en cuenta la posibilidad de dejar puntos de montaje preparados para el montaje de otros sistemas de ficheros, ya sea particiones de disco u otros dispositivos de almacenamiento.

En las máquinas en que GNU/Linux comparte la partición con otros sistemas operativos, mediante algún sistema de arranque (*lilo*, *grub* o *grub2*), pueden existir varias particiones asignadas a los diferentes operativos. Muchas veces es interesante compartir datos con estos sistemas, ya sea para leer sus ficheros o modificarlos. A diferencia de otros sistemas (que solo tienen en cuenta sus propios datos y sistemas de ficheros, y en los cuales en algunas versiones no se soportan algunos de sus propios sistemas de ficheros), GNU/Linux es capaz de tratar, como hemos visto, con una cantidad enorme de sistemas de ficheros de diferentes operativos y poder compartir la información.

Ejemplo

Si en los PC personales hemos instalado GNU/Linux, seguramente encontraremos más de un operativo, por ejemplo, otra versión de GNU/Linux con *ext2* o *3* de sistema de ficheros; podríamos encontrar un antiguo msdos con su sistema de ficheros FAT, un Windows98/ME/XP Home con FAT32 (o *vfat* para Linux) o un Windows NT/2000/XP/Vista/7/8x con sistemas NTFS (*ntfs* para Linux) y FAT32 (*vfat*) a la vez.

Nuestro sistema GNU/Linux puede leer datos (disponibles en paquetes como ficheros y directorios) de todos estos sistemas de ficheros y escribir en la mayoría de ellos.

En el caso de NTFS, hasta ciertos momentos existieron problemas en la escritura, que estaba en forma experimental en la mayoría de *drivers* de *kernel* aparecidos debido, principalmente, a las diferentes versiones que van apareciendo del sistema de ficheros, ya que existen dos versiones principales llamadas NTFS y NTFS2, y algunas extensiones como los llamados volúmenes dinámicos, o los sistemas de ficheros cifrados. Acceder con según que opción de *drivers* presentaba ciertas incompatibilidades, que podrían causar corrupciones de datos o errores en el sistema de ficheros.

Debido a FUSE, un módulo integrado en el *kernel* (a partir de 2.6.11), se ha permitido un desarrollo más flexible de sistemas de ficheros, directamente en espacio de usuario (de hecho, FUSE actúa como un "puente" entre las peticiones del *kernel* y el acceso que se hace desde el *driver*).

Gracias a las posibilidades de FUSE, se tiene un soporte más o menos completo de NTFS (mientras Microsoft no haga más cambios en la especificación), en especial desde la aparición del *driver* (basado en FUSE) ntfs-3g y la combinación con las utilidades *ntfsprogs*.

Para que se puedan leer o escribir los datos, la partición tiene que estar disponible dentro de nuestro sistema de ficheros raíz (*/*). Por lo tanto, hay que llevar a cabo un proceso de "montaje" del sistema de ficheros en algún punto de nuestro árbol de directorios.

Dependiendo de la distribución, se usan unos sistemas u otros, o también los podemos crear nosotros. Normalmente, suelen existir o bien como subdirectorios de la raíz, por ejemplo */cdrom /win /floppy*, o bien son subdirectorios dentro de */mnt*, el punto estándar de montaje (aparecen como */mnt/cdrom / mnt/floppy*...), o el directorio */media*, que es el preferido últimamente por las distribuciones. Según el estándar FHS, */mnt* se debería usar para montajes temporales de sistemas de archivo, mientras */media* se utilizaría para montar dispositivos removibles.

El proceso de montaje se realiza mediante la orden *mount* con el siguiente formato:

```
mount -t filesystem-type device mount-point
```

El tipo de *filesystem* puede ser: *msdos* (fat), *vfat* (fat32), *ntfs* (ntfs lectura), *iso9660* (para cdrom), *ext2*, *ext3*, *xfs*... (de los disponibles).

El dispositivo (*device*) es la entrada correspondiente en el directorio */dev* a la localización del dispositivo; los IDE tenían */dev/hdxy*, donde *x* es *a,b,c,* o *d* (1 master, 1 slave, 2 master, 2 slave) *e* y, el número de partición, los SCSI (*/dev/ sdx*), donde *x* es *a,b,c,d*... (según el ID SCSI asociado 0,1,2,3,4...).

Vamos a ver algunos casos:

```
mount -t iso9660 /dev/sdc /mnt/cdrom
```

montaría el CD-ROM (si es el dispositivo sdc, por ejemplo, si disponemos de otros dos discos previamente) en el punto */mnt/cdrom*.

```
mount -t iso9660 /dev/cdrom /mnt/cdrom
```

montaría el CD-ROM; */dev/cdrom* se usa como sinónimo (es un link) del dispositivo donde está conectado.

```
mount -t vfat /dev/fd0H1440 /mnt/floppy
```

montaría el disquete, */dev/fd0H1440*. Sería la disquetera A en alta densidad (1.44 MB). También puede usarse */dev/fd0*. Los disquetes prácticamente han desaparecido de los PC actuales, pero aún podemos encontrarlos en PC antiguos, o en algunos servidores antiguos.

```
mount -t ntfs /dev/sda2 /mnt/win8
```

montaría la segunda partición del primer dispositivo (la C:), de tipo NTFS (por ejemplo, un Windows 8.x, en una máquina que disponga de arranque dual).

Si estas particiones son más o menos estables en el sistema (o sea, no cambian frecuentemente) y las queremos utilizar, lo mejor será incluir los montajes para que se hagan en tiempo de ejecución, al iniciar el sistema, mediante la configuración del fichero */etc/fstab*:

```
# /etc/fstab: Información estática del sistema de ficheros
#
#<Sis. ficheros> <Punto montaje><Tipo>  <Opciones> <volcado><pasada>
/dev/sda2           /                ext3       errors = remountro        0          1
/dev/sdb3           none             swap       sw                        0          0
proc                /proc            proc       defaults                  0          0
/dev/fd0            /floppy          auto       user,noauto               0          0
/dev/cdrom          /cdrom           iso9660    ro,user,noauto            0          0
/dev/sdb1           /mnt/usb         vfat       user,noauto               0          0
```

Por ejemplo, esta configuración incluye algunos de los sistemas estándar, como la raíz en */dev/sda2*, la partición de *swap* que está en *sdb3*, el sistema *proc* (que utiliza el *kernel* para guardar su información). Y el disquete, el CD-ROM, y en este caso un disco USB de tipo *flash* (que se detecta como un dispositivo scsi). En algunos casos, se especifica *auto* como tipo de *filesystem*. Esto permite que se autodetecte el sistema de ficheros. Si se conoce, es mejor indicarlo en la configuración, y por otra parte, el *noauto* en las opciones permite que no sea montado de forma automática siempre, sino bajo petición (o acceso al directorio).

Nota

El formato de */etc/fstab*, en especial en la primera columna (*device*), depende del tipo de dispositivos que encontremos, ya sean particiones físicas, o especiales de tipo RAID software (*/dev/mdx*) o LVM (de tipo */dev/mapper*), o UUID, que son identificadores únicos asignados a discos o particiones, utilizados en tiempo de arranque. Podemos usar el comando `blkid` o acceder a */dev/disk/by-uuid* para conocer estos últimos UUID asociados.

Si tenemos esta información en el fichero, el proceso de montaje se simplifica mucho, ya que se hará o bien en ejecución, en arranque, o bien bajo demanda (para los *noauto*). Y puede hacerse ahora simplemente pidiendo que se monte el punto de montaje o el dispositivo:

```
mount /mnt/cdrom
mount /dev/fd0
```

dado que el sistema ya tiene el resto de la información.

El proceso contrario, el desmontaje, es bastante sencillo, el comando *umount* con punto o dispositivo:

```
umount /mnt/cdrom
umount /dev/fd0
```

En el caso de medios extraíbles, tipo CD-ROM (u otros), puede usarse *eject* para la extracción del soporte físico:

```
eject /dev/cdrom
```

o, en este caso, solo:

```
eject
```

Los comandos *mount* y *umount* montan o desmontan todos los sistemas disponibles. En el fichero */etc/mtab* se mantiene una lista de los sistemas montados. En un momento concreto se puede consultar o ejecutar *mount* sin parámetros para obtener esta información.

5.2. Permisos

Otro asunto que habrá que controlar, en el caso de los ficheros y directorios, es el de los permisos que queremos establecer en cada uno de ellos; hay que recordar que cada fichero puede disponer de la serie de permisos: *rwxrwxrwx*, donde se corresponden con *rwx* del propietario, *rwx* del grupo al que el usuario pertenece y *rwx* para otros usuarios. En cada uno se puede establecer el permiso de lectura (*r*), escritura (*w*) o ejecución (*x*). En el caso de un directorio, *x* denota el permiso para poder entrar en ese directorio (con el comando *cd*, por ejemplo).

Para modificar los derechos sobre un directorio o fichero, existen los comandos:

- ***chown***: cambiar propietario de los ficheros.
- ***chgrp***: cambiar grupo propietario de los ficheros.
- ***chmod***: cambiar permisos específicos (*rwx*) de los archivos.

Estos comandos también permiten la opción *-R*, que es recursiva si se trata de un directorio.

5.3. Sistemas de ficheros: Xfs, Zfs y Btrfs

Además de los clásicos sistemas de ficheros de Linux, *ext2* , 3 o 4, aparecen cada vez con más fuerza nuevos sistemas, muchos de ellos procedentes de otros sistemas UNIX, como XFS (creado por Silicon Graphics) o ZFS (creado por Sun Microsystems, ahora mantenido Oracle, fue originalmente creado con licencia abierta), así como *Btrfs*, que fue desarrollado por Oracle originalmente como nuevo sistema de ficheros para Linux, para sustituir a las series *ext2*, 3 y 4.

XFS fue creado en 1993 por SGI, como sistema de ficheros de 64 bits, de altas prestaciones, incluyendo funcionalidades de *journaling*, fue portado al *kernel* Linux en el 2001 y algunas distribuciones ya lo comienzan a incluir como sistema por defecto (RHEL y algunas versiones de CentOS). Entre las capacidades a destacar de XFS, se incluyen el ser un sistema de ficheros de 64 bits, lo que le permite llegar hasta 8 ExbiBytes de almacenamiento (2^{63} -1 bytes), que pueden generarse archivos de hasta 500 TB, su gran rendimiento en operaciones E/S en paralelo, la consistencia de los datos, gracias a su sistema de *journaling* (que facilita la recuperación delante de fallos) y varias optimizaciones para soportar amplios anchos de banda de E/S, en hardware dedicado.

En cuanto a la administración, algunos comandos:

 # mkfs.xfs

Crear un sistema de ficheros XFS en la unidad dada. Un parámetro importante es el tamaño de bloque, que se especifica con *-b size=*, siendo el rango válido normalmente entre 512 bytes y 4096 (4 KB), el bloque depende del tamaño total del *filesystem* y de la composición respecto al porcentaje de ficheros pequeños. Si se da un gran número de estos, se recomienda tamaños de bloque pequeños (512), pero si no, estamos creando problemas de prestaciones. El tamaño por defecto es de 4096, pero se puede incluir cualquier potencia de 2 en el tamaño de bloque.

La opción -l size= permite definir el tamaño reservado a los registros del *journal* que se guardarán en el *log* del sistema, esto requiere espacio de disco, que no se visualizará como espacio disponible (por ejemplo, con comandos df). El espacio puede estar en la misma partición, o en partición o volumen separado (se puede especificar con la opción -l), el defecto es en la misma partición. El tamaño máximo para un caso interno a la partición está entre 64K bloques.

Hay otras opciones útiles para definir los grupos de allocation y las unidades de *stripe*, que son dos parámetros que afectan a las prestaciones en función del patrón de accesos que se disponga. Los valores por defecto tendrían que ser

suficientes para un uso común. En todo caso consultar página man de mkfs.xfs para estos parámetros. Con *mount*, *umount* montaremos (o desmontaremos) las particiones creadas.

Un ejemplo de creación:

```
# mkfs.xfs -b size=1k -l size=10m /dev/sdc1
meta-data=/dev/sdc1 isize=256    agcount=18, agsize=4194304 blks
data     = bsize=1024   blocks=71687152, imaxpct=25
         = sunit=0      swidth=0 blks, unwritten=0
naming   = version 2             bsize=4096
log      = internal log          bsize=1024   blocks=10240, version=1
         =                       sunit=0 blks
realtime = none                  extsz=65536  blocks=0, rtextents=0

# mkdir -p /mnt/space

# mount /dev/sdc1 /mnt/space

(tambien podriamos incluir en /etc/fstab para que se montase en arranque):

/dev/sdc1  /mnt/space  xfs  defaults  0  0
```

Otros comandos:

 # xfs_quota Permite el control de cuotas de espacio para los usuarios.

 # xfs_growfs /mount/point -D size Permite el crecimiento de los sistemas de ficheros dentro del dispositivo de bloque donde residen (el espacio es indicado en -D en bloques), incluso encontrándose montado. Los sistemas de ficheros XFS no pueden reducirse *a posteriori*.

 # xfs_repair /dev/device Permite reparar el sistema de ficheros a partir del *log*; si este está corrupto, hay que inicializarlo con opciones como -L, aunque al perder el *log*, podemos tener alguna corrupción o pérdida de datos.

 # xfs_freeze -f | -u /mount/point Permite suspender las escrituras (-f) para, por ejemplo, realizar un *snapshot* (los cuales no soporta XFS, los delega en el gestor de volúmenes) del estado actual y, después, volver a activar (-u). Si el sistema XFS está sobre volúmenes LVM, no hace falta esta operación *xfs_freeze*, ya que los *snapshots* de LVM ya se encargan de suspender y reasumir después de realizar el *snapshot* LVM.

 # xfsdump -l 0 -f /dev/device /path/to/filesystem Permite un *backup* del *file system* sobre el (-f) dispositivo */dev/device* (una cinta, un fichero o un *backup* remoto). Con -l se especifica el nivel de volcado, 0 es un *full backup*, los siguientes 1-9 son *backups* incrementales a partir de un *full* previo realizado. Estos *backups* se pueden restaurar *a posteriori* con *xfsrestore*.

 # xfs_info Proporciona información sobre los *filesystems*.

Para el siguiente sistema de ficheros ZFS [Pow12], se trata de una combinación de sistema de ficheros y gestor de volúmenes lógicos, diseñado por Sun (y, posteriormente, mantenido por Oracle), incluye protecciones contra corrupción de datos, compresión de datos, *snapshots* y soporte de *copy-on-write*, COW, (técnica que optimiza múltiples copias de los mismos datos con una única fuente, con protección durante escrituras), RAID-Z (un modelo no estándar de RAID semejante a RAID 5) con soporte nativo en ZFS y soporte nativo de ACL (listas de acceso) para NFSv4.

Nota

ZFS on Linux http://zfsonlinux.org/

ZFS en Ubuntu: https://wiki.ubuntu.com/Kernel/Reference/ZFS

Para ZFS existe también un proyecto alternativo denominado OpenZFS (soporte ZFS-Like para múltiples UNIX y Linux). De las implementaciones posibles, ZFS original, no es comúnmente usado debido a licencias incompatibles; ZFS es licenciado en CDDL, que es incompatible con GPL. Una segunda posibilidad es Native ZFS on Linux, una versión realizada por el Lawrence Livermore Labs (LLNL), normalmente denominada *ZFS on Linux*, y que soporta directamente Ubuntu y Gentoo desde sus repositorios. También es posible encontrar soluciones basadas en FUSE, de manera que el sistema de ficheros se ejecute en espacio de usuario (de hecho, de forma semejante al caso de *ntfs-3g*). Pero en estos casos el soporte tiene prestaciones menores que un soporte nativo de *kernel*.

En el caso que utilicemos ZFS on Linux en una distribución Fedora, podemos consultar el proceso básico de instalación en http://zfsonlinux.org/fedora.html. Una vez realizado, disponemos ya de soporte nativo en *kernel* y podemos comenzar a administrar ZFS. En el caso de Ubuntu (a partir de la versión 15.10), ZFS está disponible como paquete en la distribución (podemos instalarlo con `sudo apt install zfsutils-linux`), y lo podemos utilizar como sistema de ficheros para cualquier disco o sistema de discos, a excepción del sistema raíz (aunque esta opción está disponible a partir de la versión 16.04).

Algunos comandos útiles, asumiendo que en esta sesión disponemos de discos sdb y sdc total libres para implementar un *pool* de almacenamiento ZFS:

```
# zpool create pdiscos /dev/sdb /dev/sdc
```

Puede que nos digan que los discos son de tipo MBR, en tal caso habrá que usar *parted* para ponerles una etiqueta de tipo GPT e inicializar la tabla de particiones nueva; y volvemos a proceder (al utilizar *parted*, hay que ir con cuidado de seleccionar, *select*, los discos adecuados a inicializar).

Nota

Los ejemplos siguientes destruyen completamente el contenido previo de los discos usados, se recomienda solo usar discos extras para hacer las pruebas, o por el contrario usar discos virtuales extras dentro de una maquina virtual.

ZFS también puede utilizar particiones, pero si se realiza sobre el disco entero, automáticamente se crean particiones y se utiliza el espacio entero. En el ejemplo anterior se ha creado un *pool* de almacenamiento denominado *pdiscos*, *zfs* automáticamente lo montará sobre */pdiscos*, que será una colección de los discos integrando el espacio de ambos discos (semejante en concepto a un RAID0).

```
# zpool create mirror pdiscos /dev/sdb /dev/sdc
```

Nos crearía un *mirror* entre los dos discos (semejante en concepto a un RAID1). O en el caso de un RAID-Z (con al menos tres discos), podemos hacer **# zpool create raidz pdiscos /dev/sdb /dev/sdc /dev/sdd**. Podemos conseguir algo semejante a RAID 6, con un raidz-2 i un disco más (4 discos mínimo).

```
# zpool status
```

Nos indica el estatus de los *pools* disponibles.

```
# zfs set mountpoint=/mnt/pdiscos pdiscos
```

Cambia el punto de montaje del *pool* (por defecto la /).

```
# zfs list pdiscos
```

Lista de subvolúmenes que incluye el *filesystem*.

```
# zfs create pdiscos/snaps
```

Crea un *sub-pool* de *pdiscos* denominado *snaps*.

```
# zfs snapshot pdiscos/snaps@hoy
```

Crea un *snapshot* denominado "hoy" (podríamos usar por ejemplo la fecha con *date* para identificarlo por estampa de tiempo. Se puede montar en cualquier momento para ver el estado del momento en que se realizó.

```
# mount -t zfs pdiscos/snaps@hoy /mnt/tmp
```

Monta un *snapshot* para examinarlo.

```
# zfs list
```

Lista *pools* y *subpools* presentes.

```
# zfs list -t snapshot
```

Lista de *snapshots* presentes.

```
# zfs destroy pdiscos/snaps@hoy
```

Borra *snapshot*.

```
# zpool destroy pdiscos
```

Borra el *pool* entero.

Btrfs (*B-tree file system*), el próximo sistema de ficheros, es un sistema de ficheros bajo licencia GPL que implementa *copy-on-write* para Linux. Se ha desarrollado entre otros por parte de Oracle, Fujitsu y Red Hat. La idea principal de este sistema es llegar a ser el sustituto de los *ext2*, 3 y 4 añadiendo tecnologías modernas que permitan *pooling*, *snapshots*, *checksums* y soporte de sistemas de fichero en multidispositivo, opciones que son vitales para necesidades de almacenamiento altas. La mayoría de distribuciones tienen ahora soporte nativo de Btrfs y es de esperar que, a medida que se incorporen nuevas funcionalidades planeadas en su desarrollo, se convierta en el *filesystem* por defecto en gran parte de ellas.

Para la administración, además del soporte del *kernel* que en muchos casos ya está activo (si no, *modprobre btrfs*), deberemos instalar el paquete *btrfs-progs*, que incluye varias utilidades para la gestión del sistema de ficheros, una vez las tengamos (suponiendo 2 discos */dev/sdb* y */dev/sdc* con partición primaria, previamente preparados con *fdisk* o *parted*):

Nota

Podéis encontrar más ejemplos extendidos de Btrfs en http://www.funtoo.org/BTRFS_Fun

```
# mkfs.btrfs -L/diskb /dev/sdb1
# mkfs.btrfs -L/diskc /dev/sdc1
```

Creamos sendos *filesystems* con las respectivas etiquetas (-L). También podríamos haber creado un *pool* con:

```
# mkfs.btrfs -L/disks /dev/sdb1 /dev/sdc1
```

O un *mirror* con:

```
# mkfs.btrfs -L/disks -draid1 /dev/sdb1 /dev/sdc1
```

Se soportan los diferentes raid0,1,5,6 y 10. Una vez dispongamos del filesystem creado, podemos examinarlo con:

```
# btrfs filesystem show /disks
```

Por ejemplo:

```
# btrfs filesystem show /disks
Label: '/disks'  uuid: 52eab416-66a7-4581-9487-243d84861331
 Total devices 2 FS bytes used 112.00KiB
 devid    1 size 4.40GiB used 929.00MiB path /dev/sdb1
 devid    2 size 4.40GiB used 909.00MiB path /dev/sdc
```

Que podemos montar después a partir del uuid que vemos con una línea */etc/fstab* como:

```
UUID=<campo_uuid_anterior>  /disks btrfs defaults 1 2
```

(colocando en /disks como punto de montaje, aunque podría ser cualquier otro punto).

Para crear *snapshots* podemos con (*/disks* aquí hace referencia al punto donde se encuentra montado el *filesystem*):

```
# btrfs subvolume snapshot /disks /disks/snapshot
```

Y listar los *snapshots* con:

```
# btrfs subvolume list /disks
```

O destruir los *snapshots* con:

```
# btrfs subvolume delete /disks/snapshot
```

Como hemos observado a lo largo de estos ejemplos, XFS, ZFS y Btrfs ofrecen bastantes posibilidades, muchas de ellas más o menos equivalentes y con grados diferentes de soporte, desde el consolidado XFS, hasta el experimental Btrfs, que seguramente se introduzca como *filesystem* por defecto en el futuro.

6. Usuarios y grupos

Los usuarios de un sistema GNU/Linux disponen de una cuenta asociada (definida con algunos de sus datos y preferencias), junto con el espacio en disco para que puedan desarrollar sus archivos y directorios. Este espacio está asignado al usuario, y solo puede ser usado por este (a no ser que los permisos especifiquen cosas diferentes).

Dentro de las cuentas asociadas a usuarios, podemos encontrar diferentes tipos:

- **La del administrador**, con identificador *root*, que solo es (o debería ser) utilizada para las operaciones de administración. El usuario *root* es el que dispone de más permisos y acceso completo a la máquina y a los archivos de configuración. Por lo tanto, también es el que más daño puede causar por errores u omisiones. Es mejor evitar usar la cuenta de *root* como si fuese un usuario más, por lo que se recomienda dejarla solo para operaciones de administración.

- **Cuentas de usuarios**: las cuentas normales para cualquier usuario de la máquina tienen los permisos restringidos al uso de ficheros de su cuenta, y a algunas otras zonas particulares (por ejemplo, los temporales en */tmp*), así como a utilizar algunos dispositivos para los que se les haya habilitado permisos.

- **Cuentas especiales de los servicios**: *lp*, *wheel*, *www-data*... cuentas que no son usadas por personas, sino por servicios internos del sistema, que los usa bajo estos nombres de usuario. Algunos de los servicios también son usados bajo el usuario de *root* (aunque por razones de seguridad debería evitarse).

Un usuario normalmente se crea mediante la especificación de un nombre (o identificador de usuario), una palabra de paso (*password*) y un directorio personal asociado (la cuenta).

La información de los usuarios del sistema (local, a no ser que se utilicen sistemas externos de directorio, como los YP NIS o NIS+, o LDAP, que veremos más adelante) está incluida en los siguientes archivos:

```
/etc/passwd
/etc/shadow
/etc/group
/etc/gshadow
```

Unas líneas del */etc/passwd* podrían ser:

```
juan:x:1000:1000:Juan Garcia,,,:/home/juan:/bin/bash
root:x:0:0:root:/root:/bin/bash
```

donde se indica (si aparecen :: seguidos, que el campo está vacío):

- *juan*: identificador de usuario en el sistema.
- *x*: palabra de paso del usuario codificada, si hay una "*x*" es que se encuentra en el fichero */etc/shadow*.
- *1000*: código del usuario; lo usa el sistema como código de identidad del usuario.
- *1000*: código del grupo principal al que pertenece; la información del grupo se encuentra en */etc/group*.
- *Juan García*: comentario; suele colocarse el nombre completo del usuario, o algún comentario para identificar el objetivo de la cuenta.
- */home/juan*: directorio personal asociado a su cuenta.
- */bin/bash*: *shell* interactivo que utilizará el usuario al interactuar con el sistema, en modo texto, o por *shell* gráfico. En este caso, el *shell* Bash de GNU, que es el utilizado por defecto. El fichero */etc/passwd* solía contener las palabras de paso de los usuarios de forma encriptada, pero el problema estaba en que cualquier usuario podía ver el fichero, y en su momento se diseñaron *cracks* que intentaban encontrar en forma bruta la palabra de paso, mediante la palabra de paso encriptada como punto de partida (palabra codificada con el sistema *crypt)*.

Para evitar esto, hoy en día ya no se colocan las palabras de paso en este archivo, solo una "*x*" que indica que se encuentran en otro fichero, que es solo de lectura para el usuario *root*, */etc/shadow,* cuyo contenido podría ser algo parecido a lo siguiente:

```
juan:a1gNcs82ICst8CjVJS7ZFCVnu0N2pBcn/:12208:0:99999:7:::
```

donde se encuentra el identificador del usuario junto con la palabra de paso encriptada. Además, aparecen (como campos separados por ":" con información respecto a la contraseña:

- Días desde el 1 de enero de 1970 en que la palabra de paso se cambió por última vez.

- Días que faltan para que se cambie (0 no hay que cambiarla).
- Días después en que hay que cambiarla (o sea, plazo de cambio).
- Días en que el usuario será avisado antes de que le expire.
- Días una vez expirado, en que se producirá la deshabilitación de la cuenta.
- Días desde el 1 de enero de 1970 en que la cuenta está deshabilitada.
- Y un campo reservado.

Además, las claves de encriptación pueden ser más difíciles, ya que ahora puede utilizarse diversos métodos como *md5*, *sha 256* o *512*, *blowfish*, o *DES* (suelen identificarse en la clave por un código x, donde x=1, 5, 6, 2a, u otro para los indicados) para proteger las palabras de paso de los usuarios.

En */etc/group* está la información de los grupos de usuarios:

```
jose:x:1000:
```

donde tenemos:

```
nombre-grupo:contraseña-grupo:identificador-del-grupo:lista-usuarios
```

La lista de usuarios del grupo puede estar presente o no, pues la información ya está en */etc/passwd*; no suele ponerse en */etc/group*. Si se pone, suele aparecer como una lista de usuarios separada por comas. Los grupos también pueden poseer una contraseña asociada (aunque no suele ser tan común), como en el caso de los de usuario, en que también existe un fichero de tipo shadow: */etc/gshadow*.

Otros ficheros interesantes son los del directorio */etc/skel*, donde se hallan los ficheros que se incluyen en cada cuenta de usuario al crearla. Recordad que, como vimos con los *shell* interactivos, podíamos tener unos *scripts* de configuración que se ejecutaban al entrar o salir de la cuenta. En el directorio *skel* se guardan los "esqueletos" que se copian en cada usuario al crearlo. Suele ser responsabilidad del administrador crear unos ficheros adecuados para los usuarios, poniendo los *path* necesarios de ejecución, inicialización de variables de sistema, variables que se necesiten para el software, etc.

A continuación, vamos a ver una serie de comandos útiles para esta administración de usuarios (mencionamos su funcionalidad y en el taller haremos algunas pruebas):

- ***useradd***: añadir un usuario al sistema.

- ***userdel***: borrar un usuario del sistema.

- ***usermod***: modificar un usuario del sistema.

- ***groupadd, groupdel, groupmod***: lo mismo para grupos.

- ***newusers, chpasswd***: pueden ser de utilidad en grandes instalaciones con muchos usuarios, ya que permiten crear varias cuentas desde la información introducida en un fichero (*newusers*) o bien cambiar las contraseñas a un gran número de usuarios (*chpasswd)*.

- ***chsh***: cambiar el *shell* de *login* del usuario.

- ***chfn***: cambiar la información del usuario, la presente en el comentario del fichero */etc/passwd*.

- ***passwd***: cambiar la contraseña de un usuario. Puede ejecutarse como usuario, y entonces pide la contraseña antigua y la nueva. En el caso de hacerlo, el *root* tiene que especificar el usuario al que va a cambiar la contraseña (si no, estaría cambiando la suya) y no necesita la contraseña antigua. Es quizás el comando más usado por el *root*, de cara a los usuarios cuando se les olvida la contraseña antigua.

- ***su***: una especie de cambio de identidad. Lo utilizan tanto usuarios como el *root* para cambiar el usuario actual. En el caso del administrador, es bastante utilizado para testear que la cuenta del usuario funcione correctamente. Hay diferentes variantes: *su* (sin parámetros, sirve para pasar a usuario *root*, previa identificación, permitiendo, cuando estamos en una cuenta de usuario, pasar a *root* para hacer alguna tarea). La sentencia *su iduser* cambia el usuario a *iduser*, pero dejando el entorno como está, o sea, en el mismo directorio. El mandato *su - iduser* hace una sustitución total, como si el segundo usuario hubiese entrado en el sistema haciendo un *login*.

Respecto a la administración de usuarios y grupos, lo que hemos comentado aquí hace referencia a la administración local de una sola máquina. En sistemas con múltiples máquinas que comparten los usuarios suele utilizarse otro sistema de gestión de la información de los usuarios. Estos sistemas, denominados genéricamente sistemas de información de red (o servicios de directorio), como NIS, NIS+ o LDAP, utilizan bases de datos para almacenar la información de los usuarios y grupos, de manera que se utilizan máquinas servidoras, donde se almacena la base de datos, y otras máquinas clientes, donde se consulta esta información. Esto permite tener una sola copia de los datos de los usuarios (o varias sincronizadas), y que estos puedan entrar en cualquier máquina disponible del conjunto administrado con estos sistemas. Además, estos sistemas incorporan conceptos adicionales de jerarquías, y/o dominios/zonas

de máquinas y recursos, que permiten representar adecuadamente los recursos y su uso en organizaciones con diferentes estructuras de organización interna de su personal y sus secciones internas.

Podemos comprobar si estamos en un entorno de tipo NIS si en las líneas *passwd* y *group* del archivo de configuración */etc/nsswitch.conf* aparece *files* en primer término, si estamos trabajando con los ficheros locales, o bien *nis* o *nisplus* según el sistema con que estemos trabajando. En general, para el usuario simple no supone ninguna modificación, ya que la gestión de las máquinas le es transparente, y más si se combina con ficheros compartidos por NFS que permite disponer de su cuenta sin importar con qué máquina trabaje. La mayor parte de los comandos anteriores pueden seguir usándose sin problema bajo NIS o NIS+; son equivalentes a excepción del cambio de contraseña, que en lugar de *passwd* se suele hacer con *yppasswd* (NIS) o *nispasswd* (NIS+), aunque suele ser habitual que el administrador los renombre (por un enlace) a *passwd*, con lo cual los usuarios no notarán la diferencia.

Veremos este y otros modos de configuración en las unidades de administración de red.

7. Servidores de impresión

Potencia y flexibilidad

Los sistemas UNIX disponen, quizás, de los sistemas de impresión más potentes y complejos, que aportan una gran flexibilidad a los entornos de impresión.

El sistema de impresión de GNU/Linux [Gt] [Smi02] está heredado de la variante BSD de UNIX. Este sistema se denominaba LPD (Line Printer Daemon). Es un sistema de impresión muy potente, ya que integra capacidades para gestionar tanto impresoras locales como de red y ofrece dentro del mismo tanto el cliente como el servidor de impresión. De forma semejante también UNIX ha dispuesto generalmente del System V Line Printer (o LPR), que era el sistema común en las otras variantes de UNIX. GNU/Linux ha integrado originalmente ambos sistemas, bien usando principalmente LPD y emulando LPR o dependiendo de la distribución integrando por defecto uno u otro.

LPD es un sistema bastante antiguo, puesto que se remonta a los orígenes de la rama BSD de UNIX (mediados de los ochenta). Por lo tanto, a LPD le suele faltar soporte para los dispositivos modernos, ya que en origen el sistema no estuvo pensado para el tipo de impresoras actuales. Tampoco fue concebido como un sistema basado en controladores de dispositivo, pues se producían solo impresoras serie o paralelas de escritura de caracteres texto.

Para la situación actual, el sistema LPD se combina con otro software común, como el sistema Ghostscript, que ofrece salida de tipo *postscript* para un rango muy amplio de impresoras para las que posee controladores. Además, se suele combinar con algún software de filtraje, que según el tipo de documento a imprimir, selecciona filtros adecuados para adaptar la impresión de documentos o formatos binarios, al sistema de impresión de destino. Así, normalmente el proceso que se sigue es (básicamente):

1) El trabajo es iniciado por un comando del sistema LPD.

2) El sistema de filtro identifica qué tipo de trabajo (o fichero) es utilizado y convierte el trabajo a un fichero *postscript* de salida, que es el que se envía a la impresora. En GNU/Linux y UNIX, la mayoría de aplicaciones suponen que la salida será hacia una impresora *postscript*, y muchas de ellas generan salida *postscript* directamente, y por esta razón se necesita el siguiente paso.

3) Ghostscript se encarga de interpretar el fichero *postscript* recibido, y según el controlador de la impresora a la que ha sido enviado el trabajo, realiza la conversión al formato propio de la impresora. Si es de tipo *postscript*, la impresión es directa; si no, habrá que realizar la traducción. El trabajo se manda a la cola de impresión.

Como hemos dicho, además del sistema de impresión LPD (con origen en los BSD UNIX), también existe el denominado sistema SystemV (de origen en la otra rama UNIX de SystemV) o LPR. Por compatibilidad, actualmente

la mayor parte de UNIX integra ambos, de manera que o bien uno u otro es el principal, y el otro se simula sobre el principal. En el caso de GNU/Linux, pasa algo parecido; según la instalación que hagamos podemos tener solo los comandos LPD de sistema de impresión, pero también será habitual disponer de los comandos SystemV. Una forma sencilla de identificar los dos sistemas (BSD o SystemV) es con el comando principal de impresión (el que envía los trabajos al sistema), en BSD es *lpr*, y en SystemV es *lp*.

Este era el panorama inicial de los sistemas de impresión de GNU/Linux, pero en los últimos años han surgido más sistemas, que permiten una mayor flexibilidad y una mayor disposición de controladores para las impresoras. Los dos principales sistemas son CUPS y, en menor grado, LPRng (de hecho, ya obsoleto, se utilizó en algunas versiones de Fedora, y ya no lo comentaremos en esta revisión del material; puede encontrase en ediciones anteriores). Últimamente es CUPS el estándar de facto para GNU/Linux, aunque los otros sistemas han de ser soportados por compatibilidad con sistemas UNIX existentes.

Los dos (tanto CUPS como LPRng) son una especie de sistema de más alto nivel, pero que no se diferencian en mucho de cara al usuario respecto a los BSD y SystemV estándar. Por ejemplo, se utilizan los mismos comandos clientes (o compatibles en opciones) para imprimir. Para el administrador sí que supondrá diferencias, ya que los sistemas de configuración son diferentes. En cierto modo, podemos considerar a LPRng y CUPS como nuevas arquitecturas de sistemas de impresión, que son compatibles de cara al usuario con los comandos antiguos.

En las distribuciones GNU/Linux actuales podemos encontrarnos con los diferentes sistemas de impresión. Si la distribución es antigua, puede que lleve incorporado tan solo el sistema BSD LPD. En las actuales, tanto Debian como Fedora/Red Hat utilizan CUPS. En algunas versiones de Red Hat existía una herramienta, *Print switch*, que permitia cambiar el sistema, conmutar de sistema de impresión, aunque últimamente solo está disponible CUPS. En Debian pueden instalarse ambos sistemas, pero son exclusivos, solo uno de ellos puede gestionar la impresión.

En el caso de Fedora, el sistema de impresión por defecto es CUPS (desapareciendo LPRng en Fedora Core 4), y la herramienta *Print switch* ya no existe por no ser necesaria. Se utiliza *system-config-printer* para la configuración de dispositivos. Debian, por defecto, utilizaba BSD LPD, pero ya es común instalar CUPS (y es la opción por defecto en nuevas versiones), y también puede utilizar LPRng. Además, cabe recordar que también teníamos la posibilidad (vista en la unidad de migración) de interaccionar con sistemas Windows mediante protocolos Samba, que permitían compartir las impresoras y el acceso a estas.

Respecto a cada uno de los sistemas [Gt]:

- **BSD LPD**: es uno de los estándares de UNIX, y algunas aplicaciones asumen que tendrán los comandos y el sistema de impresión disponibles, por lo cual, tanto LPRng como CUPS emulan el funcionamiento y los comandos de BDS LPD. El sistema LPD es utilizable, pero no muy configurable, sobre todo en el control de acceso; por eso las distribuciones se han movido a los otros sistemas más modernos.

- **LPRng**: se diseñó para ser un reemplazo del BSD; por lo tanto, la mayor parte de la configuración es parecida y solo difiere en algunos ficheros de configuración.

- **CUPS**: se trata de una desviación mayor del BSD original, y la configuración es propia. Se proporciona información a las aplicaciones sobre las impresoras disponibles (también en LPRng). En CUPS, tanto el cliente como el servidor tienen que disponer de software CUPS.

Los dos sistemas tienen emulación de los comandos de impresión de SystemV.

Para la impresión en GNU/Linux, hay que tener en cuenta varios aspectos:

- **Sistema de impresión que se utiliza**: BSD, CUPS, o LPRng (hoy prácticamente obsoleto).

Nota

Podéis encontrar información de las impresoras más adecuadas y de los controladores en: http://www.openprinting.org/printers

- **Dispositivo de impresión** (impresora): puede disponer de conexión local a una máquina o estar colocada en red. Las impresoras actuales pueden estar colocadas por conexiones locales a una máquina mediante interfaces serie, paralelo, USB, etc., o disponibles simplemente en red, como una máquina más, o con protocolos especiales propietarios. Las conectadas a red pueden actuar ellas mismas de servidor de impresión (por ejemplo, muchas láser son servidores BSD LPD), o bien pueden colgarse de una máquina que actúe de servidor de impresión para ellas.

- **Protocolos de comunicación** utilizados con la impresora o el sistema de impresión: ya sea TCP/IP directo (por ejemplo, una HP con LPD), o bien otros de más alto nivel sobre TCP/IP, como IPP (CUPS), JetDirect (algunas impresoras HP), etc. Este parámetro es importante, ya que lo debemos conocer para instalar la impresora en un sistema.

- **Sistema de filtros usado**: cada sistema de impresión soporta uno o varios.

- **Y los controladores de las impresoras**: en GNU/Linux hay bastantes tipos diferentes; podemos mencionar, por ejemplo, controladores de CUPS, propios o de los fabricantes (por ejemplo, HP y Epson los proporcionan); Gimp, el programa de retoque de imágenes, también posee controladores optimizados para la impresión de imágenes; Foomatic, un sistema de gestión de controladores que funciona con la mayoría de sistemas (CUPS,

LPD, LPRng y otros); los controladores de Ghostscript, etc. En casi todas las impresoras tienen uno o más controladores de estos conjuntos.

Respecto a la parte cliente del sistema, los comandos básicos son iguales para los diferentes sistemas. Estos son los comandos del sistema BSD (cada sistema soporta emulación de estos comandos):

- ***lpr***: envía un trabajo a la cola de la impresora por defecto (o a la que se selecciona); el *daemon* de impresión (*lpd*) se encarga de enviarlo a la cola correspondiente y asigna un número de trabajo, que será usado con los otros comandos. La impresora, por defecto, estaría indicada por una variable de sistema PRINTER, o se utilizará la primera que exista definida. En algunos sistemas se utiliza la cola *lp* (como nombre por defecto).

```
lpr -Pepson datos.txt
```

Esta instrucción mandaría el fichero *datos.txt* a la cola de impresión asociada a una impresora que hemos definido como "epson".

Nota

Si no se dispone de impresora, para probar el sistema de impresión, se puede instalar el paquete *cups-pdf* (si la distribución soporta CUPS), que instala una impresora virtual en el sistema con salida en ficheros PDF.

- ***lpq***: nos permite examinar los trabajos existentes en la cola.
  Este comando nos muestra los trabajos en cola, con el orden y tamaños de estos. Los ficheros pueden aparecer con nombres diferentes, ya que depende de si los hemos enviado con *lpr* o con otra aplicación que puede cambiar los trabajos de nombre al enviarlos, o si han tenido que pasar por algún filtro al convertirlos.

```
# lpq -P epson
Rank  Owner        Job   Files          Total Size
1st   juan         15    datos.txt      74578 bytes
2nd   marta        16    fpppp.F        12394 bytes
```

- ***lprm***: elimina trabajos de la cola. Podemos especificar un número de trabajo, o un usuario para cancelar los trabajos.

```
lprm -Pepson 15
```

Eliminar el trabajo con id 15 de la cola.

Respecto a la parte administrativa (en BSD), el comando principal sería *lpc*. Este comando permite activar y desactivar colas, mover trabajos en el orden de las colas y activar o desactivar las impresoras (se pueden recibir trabajos en las colas, pero no se envían a las impresoras).

Cabe mencionar asimismo que, para el caso de SystemV, los comandos de impresión suelen también estar disponibles, simulados sobre los de BSD. En el caso cliente, los comandos son *lp, lpstat, cancel* y, para temas de administración, *lpadmin, accept, reject, lpmove, enable, disable, lpshut.*

En las siguientes secciones veremos cómo hay que configurar un servidor de impresión para dos de los sistemas principales. Estos servidores sirven tanto para la impresión local como para atender las impresiones de clientes de red (si están habilitados).

7.1. BSD LPD

En el caso del servidor BSD LPD, hay dos ficheros principales para examinar: por una parte, la definición de las impresoras en */etc/printcap* y, por otra, los permisos de acceso por red en */etc/hosts.lpd*.

Respecto a los permisos, por defecto BSD LPD solo deja acceso local a la impresora, y por lo tanto, hay que habilitarlo expresamente en */etc/hosts.lpd*.

El fichero podría ser:

```
#fichero hosts.lpd
second
first.the.com
192.168.1.7
+@groupnis
-three.the.com
```

Indicaría que está permitida la impresión a una serie de máquinas, listadas bien por su nombre DNS o por la dirección IP. Se pueden añadir grupos de máquinas que pertenezcan a un servidor NIS (como en el ejemplo *groupnis*) o bien no permitir acceso a determinadas máquinas indicándolo con un guión "-".

En cuanto a la configuración del servidor en */etc/printcap*, se definen entradas, donde cada una representa una cola del sistema de impresión a la que pueden ir a parar los trabajos. La cola puede estar tanto asociada a un dispositivo local como a un servidor remoto, ya sea este una impresora u otro servidor.

En cada entrada, pueden existir las opciones:

- ***lp*=**: nos indica a qué dispositivo está conectada la impresora, por ejemplo *lp = /dev/lp0* indicaría el primer puerto paralelo. Si la impresora es de tipo LPD, por ejemplo, una impresora de red que acepta el protocolo LPD (como una HP), entonces podemos dejar el campo vacío y rellenar los siguientes.

- ***rm*=**: dirección con nombre o IP de la máquina remota que dispone de la cola de impresión. Si se trata de una impresora de red, será la dirección de esta.

- ***rp*=**: nombre de la cola remota, en la máquina indica antes con *rm*.

```
# Entrada de una impresora local
lp|epson|Epson C62:\
   :lp=/dev/lp1:sd=/var/spool/lpd/epson:\
   :sh:pw#80:pl#72:px#1440:mx#0:\
   :if = /etc/magicfilter/StylusColor@720dpi-filter:\filtro
   :af = /var/log/lp-acct:lf = /var/log/lp-errs:
# Entrada de impresora remota
hpremota|hpr|hp remota del departamento|:\
   :lp = :\
   :rm = servidor:rp = colahp:\
   :lf = /var/adm/lpd_rem_errs:\fichero de log.
   :sd = /var/spool/lpd/hpremota:spool local asociado
```

7.2. CUPS

CUPS es una nueva arquitectura para el sistema de impresión bastante diferente; tiene una capa de compatibilidad hacia BSD LPD, que le permite interaccionar con servidores de este tipo. Soporta también un nuevo protocolo de impresión llamado IPP (basado en http), pero solo disponible cuando cliente y servidor son de tipo CUPS. Además, utiliza un tipo de *drivers* denominados PPD que identifican las capacidades de la impresora. CUPS ya trae algunos de estos controladores, y algunos fabricantes también los ofrecen (como HP y Epson, entre otros).

CUPS tiene un sistema de administración completamente diferente, basado en diferentes ficheros: */etc/cups/cupsd.conf* centraliza la configuración del sistema de impresión, */etc/cups/printers.conf* controla la definición de impresoras y */etc/cups/classes.conf* los grupos de estas.

En */etc/cups/cupsd.conf* configuramos el sistema según una serie de secciones del archivo y las directivas de las diferentes acciones. El archivo es bastante grande; destacaremos algunas directivas importantes:

- ***Allow***: nos permite especificar qué máquinas podrán acceder al servidor, ya sean grupos o máquinas individuales, o segmentos IP de red.

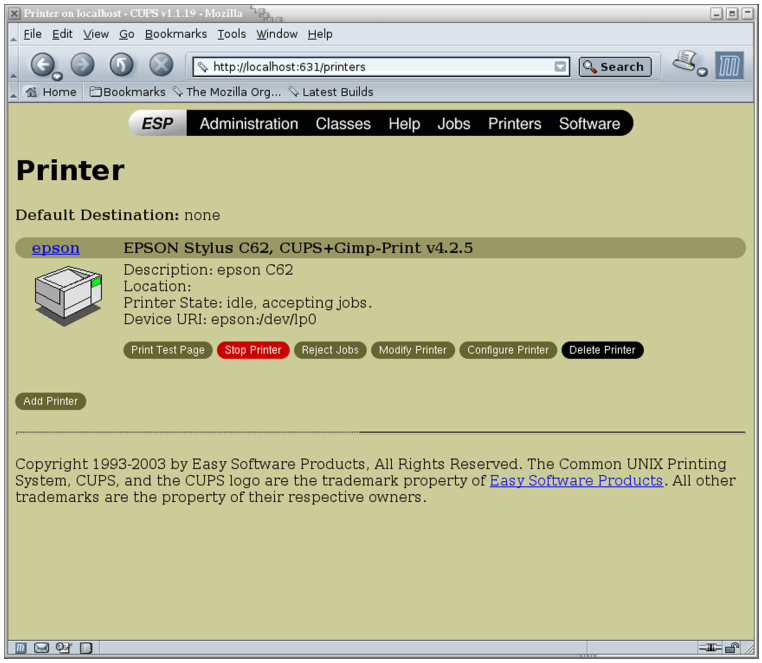
- ***AuthClass***: permite indicar si se pedirá que se autentifiquen los usuarios clientes o no.

- ***BrowseXXX***: hay una serie de directivas relacionadas con la posibilidad de examinar la red para encontrar impresoras servidas. Esta posibilidad está activada por defecto (*browsing* en *on*); por lo tanto, encontraremos disponibles todas las impresoras disponibles en la red. Podemos desactivarla, para solamente observar las impresoras que hayamos definido. Otra opción importante es *BrowseAllow*, que dice a quién le damos la posibilidad de preguntar por nuestras impresoras. Por defecto está habilitada, por lo que cualquiera puede ver nuestra impresora desde nuestra red.

Cabe señalar que CUPS, en principio, está pensado para que tanto clientes como el servidor funcionen bajo el mismo sistema; si los clientes utilizan LPD, hay que instalar un *daemon* de compatibilidad llamado *cups-lpd* (en paquetes

como *cupsys-bsd*). En este caso, CUPS acepta trabajos que provengan de un sistema LPD, pero no controla los accesos (*cupsd.conf* solo sirve para el propio sistema CUPS), por lo que habrá que implementar alguna estrategia de control de acceso, tipo *firewall*.

Para la administración desde línea de comandos, CUPS es un tanto peculiar, ya que acepta tanto comandos LPD como SystemV en los clientes, y la administración suele hacerse con el comando *lpadmin* de SystemV. En cuanto a herramientas gráficas, disponemos de *gnome-cups-manager*, *gtklp*, utilidades como *system-config-printers* o la interfaz por web que trae el mismo sistema CUPS, accesible en http://localhost:631.



Interfaz para la administración del sistema CUPS.

8. Discos y gestión *filesystems*

Respecto a las unidades de almacenamiento, como hemos visto, poseen una serie de dispositivos asociados, dependiendo del tipo de interfaz:

- **IDE** (solamente en PC antiguos): dispositivos como:
  /dev/hda disco master, primer conector IDE;
  /dev/hdb disco slave, del primer conector;
  /dev/hdc master, segundo conector;
  /dev/hdd slave, segundo conector.

- **SCSI**: dispositivos */dev/sda*, */dev/sdb*... siguiendo la numeración que tengan los periféricos en el Bus SCSI. Los discos SATA e IDE del mismo sistema también suelen seguir esta nomenclatura, debido a la capa de emulación scsi presente en el *kernel* para estos dispositivos.

- **Disquetes** (ya obsoletos pero posibles de encontrar en PC antiguos): dispositivos */dev/fdx*, con *x* número de disquetera (comenzando en 0). Hay diferentes dispositivos, dependiendo de la capacidad del disquete, por ejemplo, el disquete de 1.44 MB en la disquetera A sería */dev/fd0H1440*.

Respecto a las particiones presentes, el número que sigue al dispositivo representa el índice de la partición dentro del disco, y es tratado como un dispositivo independiente: */dev/hda1* primera partición del primer disco IDE, o */dev/sdc2*, segunda partición del tercer dispositivo SCSI. En el caso de los discos IDE (con MBR), estos permiten cuatro particiones denominadas primarias y un mayor número en extendidas (o lógicas). Así, si */dev/sdan*, *n* será inferior o igual a 4, se tratará de una partición primaria; si no, se tratará de una partición lógica con *n* superior o igual a 5.

Con los discos y los sistemas de ficheros (*filesystems*) asociados, los procesos básicos que podemos realizar se engloban en:

- **Creación de particiones**, o modificación de estas. Mediante comandos como *fdisk* o parecidos (*cfdisk*, *sfdisk*). Y *parted*, como vimos, para máquinas con soporte de GPT (nuevos PC con *firmware* UEFI).

- **Formateo de disquetes**: en caso de disquetes, pueden utilizarse diferentes herramientas: *fdformat (*formateo de bajo nivel), *superformat (*formateo a diferentes capacidades en formato msdos), *mformat (*formateo específico creando *filesystem* msdos estándar).

- **Creación de *filesystems* Linux**, en particiones, mediante el comando *mkfs*. Hay versiones específicas para crear *filesystems* diferentes: *mkfs.ext2*,

mkfs.ext3, y también *filesystems* no Linux: *mkfs.ntfs*, *mkfs.vfat, mkfs.msdos*, *mkfs.minix* u otros. Para CD-ROM como *mkisofs,* a la hora de crear los iso9660 (con extensiones *joliet* o *rock ridge*), que puedan ser una imagen de lo que después se acabará grabando sobre un CD/DVD, y junto con comandos como *cdrecord* (o *wodim*) permitirá finalmente crear/grabar los CD/DVD. Otro caso particular es la orden *mkswap,* que permite crear áreas de *swap* en particiones que, más tarde, se pueden activar o desactivar con *swapon* y *swapoff*.

- **Montaje de los *filesystems***: comandos *mount, umount*.

- **Verificación de estado**: la principal herramienta de verificación de *filesystems* Linux es el comando *fsck*. Este comando comprueba las diferentes áreas del sistema de ficheros para verificar la consistencia y comprobar posibles errores y, en los casos en que sea posible, corregirlos. El propio sistema activa automáticamente el comando *fsck* en el arranque cuando detecta situaciones donde se ha producido una parada incorrecta (un apagón eléctrico o accidental de la máquina), o bien pasado un cierto número de veces en que el sistema se ha arrancado. Esta comprobación suele comportar cierto tiempo, normalmente algunos minutos (dependiendo del tamaño de datos). También existen versiones particulares para otros sistemas de ficheros: *fsck.ext2*, *fsck.ext3*, *fsck.vfat*, *fsck.msdos*, etc. El proceso del *fsck* se realiza con el dispositivo en modo de "solo lectura" con particiones montadas. Se recomienda desmontar las particiones para realizar el proceso si se detectan errores y hay que aplicar correcciones. En determinados casos, por ejemplo, si el sistema por comprobar es la raíz / y se detecta algún error crítico, se nos pedirá que cambiemos de modo de ejecución del sistema (*runlevel*) hacia modo solo *root* y hagamos allí la verificación. En general, si hay que hacer la verificación, se recomienda hacer estas en modo superusuario (podemos conmutar de modo de *runlevel* con los comandos *init* o *telinit*).

- **Procesos de *backup***: ya sean del disco, bloques de disco, particiones, *filesystems*, ficheros... Hay varias herramientas útiles para ello: *tar* nos permite copiar ficheros hacia un fichero o a unidades de cinta; *cpio,* de forma parecida, puede realizar *backups* de ficheros hacia un fichero; tanto *cpio* como *tar* mantienen información de permisos y propietarios de los ficheros; *dd* permite copias, ya sea de ficheros, dispositivos, particiones o discos a fichero; es un poco complejo y hay que conocer información de bajo nivel, tipo, tamaños, número de bloques y/o sectores... Puede enviarse también a cintas.

- **Utilidades diversas**: ciertos comandos individuales (dependerá de utilidades exclusivas para un sistema de ficheros concreto), algunos de ellos utilizados por los procesos anteriores para hacer tratamientos distintos: *badblocks* para encontrar bloques defectuosos en el dispositivo; *dumpe2fs* para obtener información sobre *filesystems* Linux; *tune2fs* permite hacer

procesos de *tunning* de *filesystems* Linux de tipo ext2, ext3 o ext4 y ajustar diferentes parámetros de comportamiento.

A continuación, destacamos dos temas relacionados con la concepción del espacio de almacenamiento, que son utilizados en varios ambientes para la creación base del espacio de almacenamiento: el uso de RAID software y la creación de volúmenes dinámicos.

8.1. RAID software

La configuración de discos mediante esquemas RAID es uno de los esquemas de almacenamiento de alta disponibilidad más usados actualmente, cuando disponemos de varios discos para implementar nuestros sistemas de ficheros.

El enfoque principal de las diferentes técnicas existentes está en la tolerancia a fallos que se proporciona desde un nivel de dispositivo, el conjunto de discos, a diferentes tipos posibles de fallos tanto físicos como de sistema, para evitar las pérdidas de datos o los fallos de coherencia en el sistema. Así también algunos esquemas que están diseñados para aumentar las prestaciones del sistema de discos, ampliando el ancho de banda de estos disponible hacia el sistema y las aplicaciones.

Típicamente, hoy en día RAID en hardware (mediante tarjetas controladoras hardware), se puede encontrar en servidores empresariales (cuando comienzan a tener cierta presencia en equipos de escritorio, con placas base con capacidades para algunos RAID básicos), donde se hallan disponibles diferentes soluciones hardware que cumplen estos requerimientos de fiabilidad y de maximizar prestaciones. En particular, para ambientes con aplicaciones intensivas en disco, como *streaming* de audio y/o vídeo, o grandes bases de datos.

Conviene destacar un error común en el tema RAID, que proporciona ciertas capacidades de tolerancia a fallos, pero no evita la realización de *backups* de los datos disponibles de forma periódica. Si se supera la capacidad de tolerancia a fallos, también se pierden datos (en algunos casos, completamente).

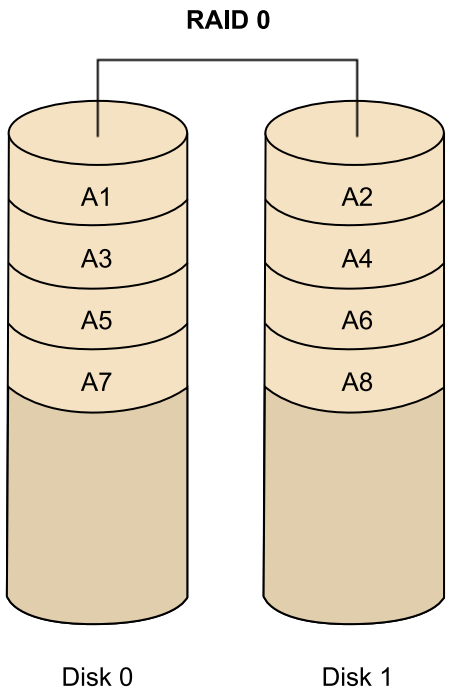
En general, este hardware se encuentra en forma de tarjetas (o integrado en la máquina) de tipo controladora RAID de discos, que implementan la gestión de uno o más niveles (de la especificación RAID), sobre un conjunto de discos (desde mínimo dos discos) administrado por esta controladora.

En RAID se distingue una serie de niveles (o configuraciones posibles) que pueden proporcionarse (cada fabricante de hardware, o el software concreto, puede soportar uno o varios de estos niveles). Cada nivel de RAID se aplica sobre un conjunto de discos, a veces denominado array RAID (o matriz de discos RAID), los cuales suelen ser (idealmente) discos iguales en tamaño (o iguales a tamaños por grupos). Por ejemplo, para realizar un caso de *array* podrían utilizarse cuatro discos de 500GB, o en otro caso, dos grupos (a 500GB) de

dos discos, uno de 150GB y otro de 350GB. En algunos casos de controladores hardware, no se permite que los discos (o en grupos) sean de diferentes tamaños; en otros pueden utilizarse, pero el array queda definido por el tamaño del disco (o grupo) más pequeño.

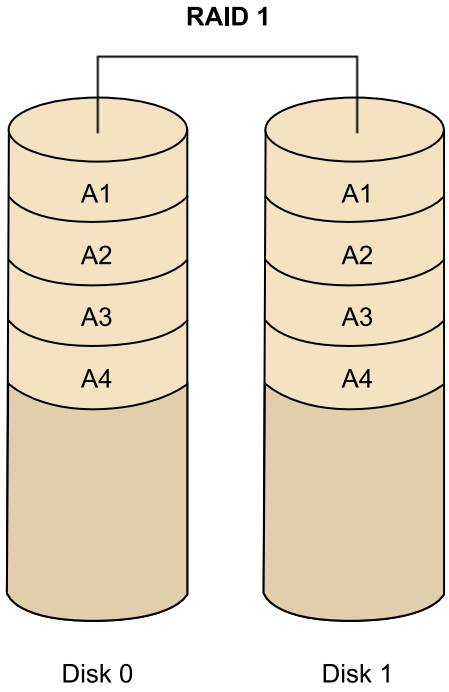
Describimos conceptos básicos de algunos niveles en la siguiente lista (tened en cuenta que, en algunos casos, la terminología no es plenamente aceptada, y puede depender de cada fabricante):

- **RAID 0**: se distribuyen los datos equitativamente entre uno o más discos sin información de paridad o redundancia; no se está ofreciendo tolerancia al fallo. Solo se están repartiendo datos; si el disco falla físicamente la información se pierde y debemos recuperarla desde copias de seguridad. Lo que sí aumenta es el rendimiento, dependiendo de la implementación de RAID0, ya que las operaciones de lectura y escritura se dividirán entre los diferentes discos.



- **RAID 1**: se crea una copia exacta (*mirror)* en un conjunto de dos o más discos (denominado array RAID). En este caso, resulta útil para el rendimiento de lectura (que puede llegar a incrementarse de forma lineal con el número de discos), y en especial por disponer de tolerancia al fallo de uno de los discos, ya que (por ejemplo, con dos discos) se dispone de la misma información. RAID 1 suele ser adecuado para alta disponibilidad, como entornos de 24x7, donde debemos disponer críticamente de los recursos. Esta configuración nos permite también (si el hardware lo soporta) el intercambio en caliente (*hot-swap*) de los discos. Si detectamos el fallo

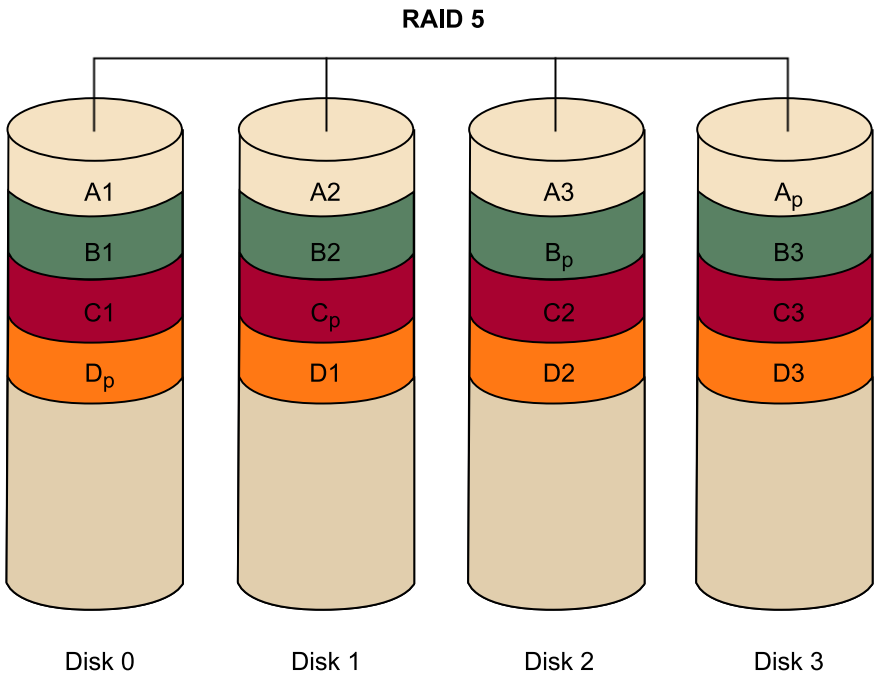
en uno de ellos, podemos sustituirlo, sin apagar el sistema, por un disco nuevo.



- **RAID 2**: en los anteriores se dividen los datos en bloques a repartir. Aquí se divide en bits y se utilizan códigos de redundancia para la corrección de datos. No es prácticamente utilizado, a pesar de las altas prestaciones que alcanzaría, ya que necesita idealmente un número muy alto de discos, uno por bit de datos y varios para el cálculo de la redundancia (por ejemplo, en un sistema de 32 bits, llegaría a usar 39 discos).

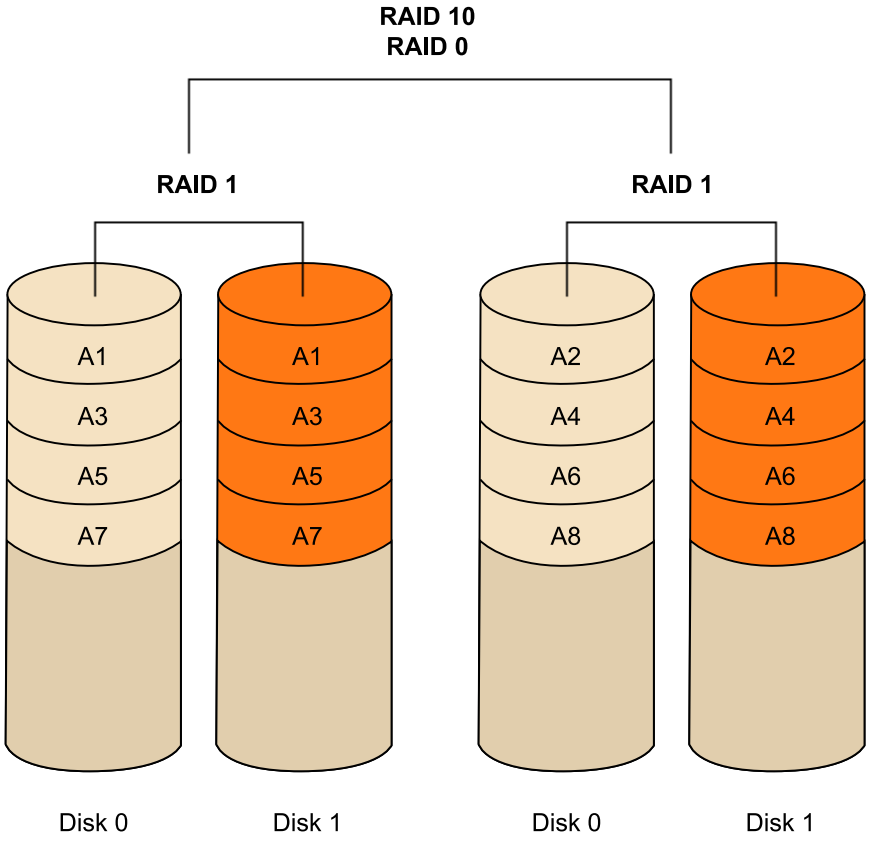
- **RAID 3**: utiliza división en bytes con un disco dedicado a la paridad de los bloques. Tampoco es muy utilizada, ya que según el tamaño de los datos y posiciones no permite accesos simultáneos. RAID 4 es semejante, aunque divide a nivel de bloques en lugar de bytes, lo que permite que sí se puedan servir peticiones simultáneas cuando se solicita un único bloque.

- **RAID 5**: Se usa división a nivel de bloques, distribuyendo la paridad entre los discos. Tiene amplio uso, debido al esquema sencillo de paridad, y a que este cálculo se implementa de forma sencilla por hardware, con buenas prestaciones.



- **RAID 0+1 (o 01)**: un *mirror* de divisiones es un nivel de RAID anidado. Se implementan, por ejemplo, dos grupos de RAID 0, los cuales son usados en RAID 1 para crear *mirror* entre ellos. Una ventaja es que, en caso de fallo, puede reconstruirse el nivel de RAID 0 usado gracias a la otra copia, pero si quieren añadirse discos hay que añadirlos a todos los grupos de RAID 0 de igual forma.

- **RAID 10 (1+0)**: división de *mirrors*, grupos de RAID 1 bajo RAID 0. Así, en cada grupo de RAID 1 puede llegar a fallar un disco sin que se pierdan datos. Claro que esto obliga a reemplazarlos, ya que si no el disco que quede en el grupo se convierte en posible punto de fallo de todo el sistema. Es una configuración que suele usarse para base de datos de altas prestaciones (por la tolerancia a fallos y la velocidad, al no estar basada en cálculos de paridad).



Algunas consideraciones a tener en cuenta sobre RAID en general:

- RAID mejora el *uptime* del sistema, ya que algunos de los niveles permiten que discos fallen y el sistema siga siendo consistente, y dependiendo del hardware, incluso puede cambiarse el hardware problemático en caliente sin necesidad de parar el sistema, cuestión especialmente importante en sistema críticos.

- RAID puede mejorar el rendimiento de las aplicaciones; en especial, en los sistemas con implementaciones de *mirror* es posible que la división de datos permita que las operaciones lineales de lectura se incrementen significativamente, debido a la posibilidad de que los discos ofrezcan, de modo simultáneo, partes de esta lectura, aumentando la tasa de transferencia de datos.

- RAID no protege los datos; evidentemente, la destrucción por otros medios (virus, mal funcionamiento general o desastres naturales) no está protegida. Hemos de basarnos en esquemas de copias de seguridad. Tengamos en cuenta que algunos esquemas protegen contra uno o dos fallos de discos del array, pero si hay más o dependiendo del esquema, directamente (caso RAID 0) se perderán datos.

- No se simplifica la recuperación de datos; si un disco pertenece a un array RAID, tiene que intentar recuperarse en ese ambiente. Se necesita software específico o los controladores hardware para acceder a los datos.

- Por el contrario, no suele mejorar aplicaciones típicas de usuario –sean de escritorio– debido a que estas aplicaciones tienen componentes altos de acceso aleatorio a datos, o a conjuntos de datos pequeños; puede que no se beneficien de lecturas lineales o de transferencias de datos sostenidas. En estos ambientes, es posible que no se note apenas mejoría de prestaciones.

- Algunos esquemas aumentan de velocidad las operaciones de lectura, pero por otra parte penalizan las de escritura (caso RAID 5 por el cálculo de paridad que hay que escribir). Si el uso es básicamente de escritura, habrá que buscar qué esquemas no penalizan o consiguen el ratio de escritura que necesitamos (algunos casos, como RAID 0,1, o algunas modalidades de RAID 10 son equivalentes a escribir en un disco único, o incluso aumentan las prestaciones en este sentido).

- No se facilita el traslado de información; sin RAID es bastante fácil trasladar datos, simplemente moviendo el disco de un sistema a otro. En el caso de RAID es casi imposible (a no ser que dispongamos del mismo hardware controlador) mover un array de discos a otro sistema.

En el caso de GNU/Linux, se da soporte al hardware RAID mediante varios módulos de *kernel*, asociados a diferentes conjuntos de fabricantes o circuitos base, *chipsets*, de estas controladoras RAID. Se permite así al sistema abstraerse de los mecanismos hardware y hacerlos transparentes al sistema y al usuario final. Así, estos módulos de *kernel* nos permiten el acceso a los detalles de estas controladoras, y a su configuración de parámetros de muy bajo nivel, que en algunos casos (especialmente en servidores que soportan carga elevada de E/S), pueden ser interesantes para procesos de *tunning* del sistema de discos que use el servidor. Se busca maximizar las prestaciones del sistema.

La otra posibilidad que analizaremos aquí es la realización de estos procesos mediante componentes software, en concreto el componente software RAID de GNU/Linux.

GNU/Linux dispone en *kernel* del *driver* llamado Multiple Device (*md*), que podemos considerar como el soporte a nivel *kernel* para RAID. Mediante este *driver* podemos implementar niveles de RAID generalmente 0,1,4,5,6 y anidados (por ejemplo, RAID 10) sobre diferentes dispositivos de bloque como discos IDE, SATA o SCSI. También dispone del nivel *linear*, como nivel donde se produce una combinación lineal de los discos disponibles (donde no importa que sean de diferentes tamaños), de manera que se escribe consecutivamente en los discos.

Para la utilización del RAID software en Linux, debemos disponer del soporte RAID en el *kernel*, y en su caso los módulos *md* activos (además de algunos *drivers* específicos según el nivel, ved *drivers* disponibles asociados a RAID, por ejemplo en Debian con *modconf*). El método preferido para la implementación de arrays de discos RAID, mediante el software RAID ofrecido por Linux, es mediante el proceso de instalación del sistema inicial, o bien mediante la utilidad *mdadm*. Esta utilidad nos permite crear los arrays y gestionarlos.

Webs recomendadas

Para consultar conceptos de software RAID, ved Linux RAID wiki:

https://raid.wiki.kernel.org/index.php/Linux_Raid

http://www.linuxfoundation.org/collaborate/workgroups/linux-raid

Ved también niveles RAID soportados:

https://raid.wiki.kernel.org/index.php/Introduction#The_RAID_levels

Observemos algunos casos prácticos. Supongamos unos discos SCSI */dev/sda*, */dev/sdb*... en los cuales disponemos de varias particiones disponibles para implementar RAID:

Creación de un array *linear:*

```
# mdadm --create --verbose /dev/md0 --level=linear --raid-devices=2 /dev/sda1 /dev/sdb1
```

donde se genera un array *linear* a partir de las particiones primeras de */dev/sda* y */dev/sdb*, creando el nuevo dispositivo */dev/md0*, que ya puede ser usado como nuevo disco (suponiendo que exista el punto de montaje */mnt/discoRAID*):

```
# mkfs.ext2fs /dev/md0
# mount /dev/md0 /mnt/discoRAID
```

Para un RAID0 o RAID1 podemos cambiar simplemente el nivel (*--level*) a *raid0* o *raid1*. Con *mdadm --detail /dev/md0* podremos comprobar los parámetros del nuevo array creado.

También podemos consultar la entrada *mdstat* en */proc* para determinar los arrays activos, así como sus parámetros. En especial, con los casos con *mirror* (por ejemplo, en los niveles 1, 5...) podremos observar en su creación la construcción inicial del array de las copias, en */proc/mdstat* indicará el nivel de reconstrucción (y el tiempo aproximado de finalización).

mdadm dispone de muchas opciones que nos permiten examinar y gestionar los diferentes arrays RAID software creados (podemos ver una descripción y ejemplos en *man mdadm*).

Otra cuestión importante es que los arrays RAID por software no son automáticamente reconocidos en arranque (como sí pasa con el RAID hardware soportado), ya que de hecho dependen de la construcción con *mdadm*. Para que la definición de un array software sea persistente, hace falta registrarla en el fichero de configuración */etc/mdadm.conf* (la ubicación puede depender de la distribución). Un paso sencillo es crearlo automáticamente a partir del resultado de la orden.

```
mdadm --detail --scan (puede añadirse tambien --verbose)
```

Otra consideración importante son las optimizaciones a que se pueden someter los arrays RAID para mejorar su rendimiento, tanto por monitorizar su comportamiento como por optimizar parámetros del sistema de ficheros, para realizar un uso más efectivo de los niveles RAID y sus características.

Nota

La optimización de los arrays RAID puede ser una fuente importante de sintonización del sistema. Examinad algunas cuestiones en https://raid.wiki.kernel.org/index.php/Performance o en la propia página man de *mdadm*.

Pasaremos a detallar otro caso de configuración RAID más elaborado basado en la elaboración de un RAID 5.

Este nivel RAID (de como mínimo tres discos) presenta una configuración distribuida de datos y pariedad en los discos que forman el array. Se presenta muy buen rendimiento en lectura, pero disminuye la escritura (respecto a un único disco) ya que hay que calcular la pariedad y distribuirla entre los discos (en casos RAID hardware, este caso de cálculo de pariedad se accelera mediante hardware). Otro dato a tener en cuenta es que la capacidad final obtenida es la suma de N-1 discos, y esta configuración es resistente a un fallo de disco presente en el array, el cual puede reemplazarse y volver a reconstruir el array. El nivel no soporta dos fallos de disco (probabilidad que de hecho puede aumentar si en el array RAID se integra un número elevado de discos), con lo cual, como ya comentamos, disponer de RAID no nos inhibe del proceso de *backup* de datos. Si quisiéramos disponer de más fiabilidad, podríamos movernos a RAID 6 (altamente recomendable frente a RAID5 por fiabilidad, a costa de disponer de un mínimo de 4 discos), que posee cálculo con pariedad dual, a costa de bajar un poco el rendimiento, o a configuraciones con RAID 10, que tienen un buen compromiso entre rendimiento y fiabilidad.

Realizaremos, en este caso, el proceso de construcción con cuatro discos SATA de alta capacidad 1,5TB, obteniendo un almacenamiento final de 4,5TB (aproximadamente). El array RAID es complementario al sistema existente; en casos que tenga que incluir particiones de *boot*, el proceso es más delicado (ved las recomendaciones de la RAID software wiki, comentada anteriormente).

Utilizaremos cuatro discos SATA presentes en el sistema como */dev/sdb*, *sdc*, *sdd*, *sde*. El disco */dev/sda* se supone que incluye el sistema y las particiones de arranque, y no forma parte del RAID. Primero hemos de pasar por la inicializacion de los discos (este proceso borra cualquier contenido previo); definiremos una partición única (en este caso, vamos a integrar todo el almacenamiento disponible; en otros esquemas podrían realizarse varias particiones),

creada con *fdisk*. Fundamentalmente, seleccionamos nueva particion primaria, requisitos por defecto y cambiamos el tipo de partición a "*Linux Raid autodetect*" (código *fd*), haciendo el proceso con *fdisk*, sea *x* cada disco diferente del array, y el paréntesis incluye las opciones mediante teclado de *fdisk*:

```
# fdisk /dev/sdx            (d n p 1 ENTER ENTER t fd w)
```

(para nuestro caso 4 veces con *x=b,c,d,e*)

Respecto a esta inicialización, hay que tener cuidado con los tamaños de disco, ya que *fdisk* con particiones tipo MBR solo soporta particiones hasta 2TB. Si los discos fueran mayores deberíamos movernos a otras herramientas de particionado, como *gparted/parted,* que soportan nuevos tipos de particiones como GPT, que ya no disponen de estas restricciones de tamaño.

Una vez realizada esta inicialización, ya podemos pasar a construir el array:

```
# mdadm --create /dev/md0 --level=6 --verbose --force --chunk=512
--raid-devices=4 /dev/sdb1 /dev/sdc1 /dev/sdd1 /dev/sde1
```

Un parámetro importante en RAID más avanzados es el tamaño del parámetro *chunk* (por defecto, 64 unidad en kb), que tiene especial relevancia en las prestaciones del RAID. Existen estudios que determinan tamaños adecuados de *chunk* en función de la funcionalidad final del RAID (escritura o lectura, tamaño promedio de los accesos, acceso principalmente aleatorio o secuencial, etc.). En general, en RAID6 se recomiendan tamaños de 128 en adelante, e incluso más elevados 512-1024, dependiendo del tamaño promedio de los archivos. Si disponemos de más discos que los mínimos necesarios, también podríamos añadir discos como *spare*, en el comando anterior con, por ejemplo, *--spare-devices=1*. Los discos de respaldo tipo *spare* no forman parte del array RAID inicial, pero si uno de los discos activos falla, cuando el fallo es detectado, el disco es marcado como *bad*, se quita del *array* y la reconstrucción del *array* comienza inmediatamente en el primer disco *spare* disponible.

Una vez lanzado el comando anterior, comenzará la construcción del RAID en segundo plano. Podemos ir consultando */proc/mdstat*, que nos mencionará la velocidad de construcción y el porcentaje, así como una estimación de tiempo para finalizar. Los tiempos de construcción son bastante grandes para altas capacidades de espacio, y pueden ir desde algunas horas hasta días, dependiendo de las capacidades de los discos y máquina.

Podríamos estar viendo en */proc/mdstat*:

```
# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid6 sdb1[0] sde1[3] sdd1[2] sdc1[1]
   4395406848 blocks level 6, 512k chunk, algorithm 2 [4/3] [UUU_]
```

```
   [==>..................] recovery = 12.6% (37043392/292945152) finish=127.5min
speed=33440K/sec

unused devices: <none>
```

Y una vez finalizado:

```
# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid6 sdb1[0] sde1[3] sdd1[2] sdc1[1]
   4395406848 blocks level 6, 512k chunk, algorithm 2 [4/4] [UUUU]

unused devices: <none>
```

El siguiente paso es crear un *filesystem* en nuestro recientemente creado array, visto por el sistema como el dispositivo */dev/md0* (este paso podría iniciarse simultáneamente durante la reconstrucción, pero por velocidad y seguridad en algunos pasos críticos iniciales se recomienda realizarlo al final del proceso).

En nuestro caso, vamos a optimizar ligeramente esta creación examinando algunos parámetros; crearemos un *filesystem ext3* en el array completo (es recomendable usar un gestor de volúmenes primero, como LVM, que veremos a continuación, porque nos permitirá crear varias particiones lógicas, que puedan en el futuro expandirse o contraerse según las necesidades).

Para la creación del *filesystem* hay algunas recomendaciones generales que provienen a partir del tamaño de *chunk* (número de datos consecutivos que reside en un disco en KB), que de hecho nos define cómo se accede al array, y de los discos presentes, un posible esquema de creación de un *filesystem ext3*:

```
mkfs.ext3 -v -b 4096 -E stride=128,stripe-width=384 /dev/md0
```

Donde *-b 4096* es el tamaño de bloque de disco, recomendado para *filesystems* muy grandes, y *-E* define opciones del *filesystem* para optimizar su rendimiento. Estas opciones pueden variarse después con el comando *tune2fs*.

En estas opciones suele sugerirse un cálculo relacionado con el *chunk*, que puede ser en nuestro ejemplo:

- *chunk size* = 512kB (colocado en el comando *mdadm* en la creación del array)
- *block size* = 4kB (recomendado para grandes *filesystems*)
- *stride* = *chunk / block* = 512kB / 4k = 128kB
- *stripe-width* = *stride* * ((*n* discos en el raid6) - 2) = 128kB * ((4) - 2) = 128kB * 2 = 256kB

Esto nos permite ajustar los parámetros del sistema de ficheros al uso del *chunk* escogido.

Una vez creado el sistema de ficheros, ya tenemos el array disponible para que pueda ser montado en una ubicación del sistema (suponiendo un directorio previo */mnt/raid6*):

```
# mount -t ext3 /mnt/raid6 /dev/md0
```

Y ya lo tenemos disponible en el sistema. Si queremos hacerlo fijo, recordad introducir los parámetros en */etc/fstab* para que se monte en arranque. Y en este caso de array software es recomendable (aunque algunos *kernels* recientes ya soportan autodetección de RAID software), colocar la información RAID en el fichero */etc/mdadm.conf*, mediante consulta del identificador del array por medio de los comandos *# mdadm –detail –scan* (con o sin *–verbose* si se necesita la identificación de dispositivos):

```
#mdadm --detail --scan --verbose
ARRAY        /dev/md0         level=raid6        num-devices=4         metadata=0.90
UUID=dcc77e17:94093185:66731ad6:6353ec0b
  devices=/dev/sdb1,/dev/sdc1,/dev/sdd1,/dev/sde1
```

En el siguiente reinicio del sistema ya dispondremos del array montado en arranque. También, como punto final, podemos obtener la información del array con *mdadm –detail*:

```
# mdadm --detail /dev/md0
/dev/md0:
   Version : 0.90
 Creation Time : Sat May 8 09:33:06 2010
   Raid Level : raid6
   Array Size : 4395406848 (4191.79 GiB 4500.90 GB)
 Used Dev Size : 1465135616 (1397.26 GiB 1500.30 GB)
   Raid Devices : 4
  Total Devices : 4
Preferred Minor : 0
   Persistence : Superblock is persistent
```

```
    Update Time : Fri May 21 12:14:31 2010
       State : clean
  Active Devices : 4
Working Devices : 4
   Failed Devices : 0
   Spare Devices : 0
   Layout : left-symmetric
   Chunk Size : 512K

   UUID : dcc77e17:94093185:66731ad6:6353ec0b (local to host kaoscore)
   Events : 0.125

  Number  Major  Minor  RaidDevice  State
   0       8      17         0      active sync  /dev/sdb1
   1       8      33         1      active sync  /dev/sdc1
   2       8      49         2      active sync  /dev/sdd1
   3       8      65         3      active sync  /dev/sde1
```

Respecto al funcionamiento del sistema, hay que examinar tanto este informe como el proporcionado por */proc/mdstat* para controlar de forma periódica (manualmente, mediante *cron*, o por notificaciones mediante mail) el estado del RAID (en este caso activo), para determinar si se produce algun fallo de disco. En este estado el array pasaría a *degraded,* y el sistema perdería su capacidad de tolerancia a un siguiente fallo. Es entonces necesario detectar qué disco está fallando y sustituirlo si no se tienen discos *spare* en el *array*, en cuyo caso comenzaría automáticamente la reconstrucción. Si no se dispone de *spare*, examinamos qué disco es para sustituirlo, para que comience la reconstrucción del *array* (mediante *mdadm* puede eliminarse un disco del *array*y añadir, una vez realizado el cambio de hardware, el nuevo disco). También es posible mediante *mdadm* simular degradaciones o fallos, lo que nos permite testear nuestro *array* delante de condiciones extremas.

Finalmente, destacamos que la creación de arrays RAID es muy interesante para la realización de grandes soportes de discos para entornos empresariales donde se intenta maximizar la tolerancia a fallos, la recuperación rápida delante de problemas y un soporte continuo de servicios sin interrupciones. Es un campo también donde se necesita un entorno cuidadoso de análisis de rendimiento de las soluciones, de los requisitos iniciales del servicio y experimentación con las diferentes soluciones posibles.

8.2. Volúmenes lógicos (LVM)

En un momento determinado, surge la necesidad de abstraerse del sistema físico de discos, y de su configuración y número de dispositivos, para que el sistema (operativo) se encargue de este trabajo y no nos tengamos que preocupar

de estos parámetros directamente. En este sentido, puede verse el sistema de volúmenes lógicos como una capa de virtualización del almacenamiento que permite una visión más simple que facilite la utilización fluida y sencilla.

En el *kernel* Linux se dispone de LVM (*logical volume manager*) [War13], que se basó en ideas desarrolladas de gestores de volúmenes de almacenamiento usados en HP-UX (una versión UNIX propietaria de HP). Actualmente existen dos versiones, de las que la LVM2 es la más utilizada, por una serie de prestaciones añadidas que incorpora.

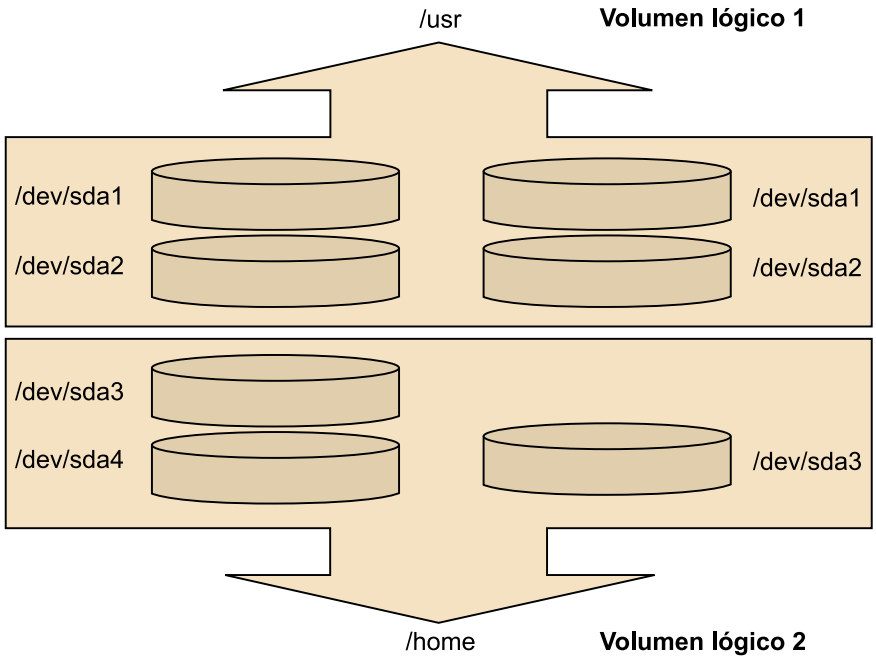
La arquitectura de una LVM consiste típicamente en los componentes (principales):

1) Volúmenes físicos (PV): son los discos duros, o particiones de estos, o cualquier otro elemento que aparezca como un disco duro de cara al sistema (por ejemplo, un RAID software o hardware).

2) Volúmenes lógicos (LV): es el equivalente a la partición del disco físico. Esta LV es visible en el sistema como un dispositivo de bloques (absolutamente equivalente a una partición física), y puede contener un sistema de ficheros (por ejemplo, el */home* de los usuarios). Los volúmenes tienen más sentido para los administradores, ya que pueden usarse nombres para identificarlos (así podemos utilizar un dispositivo lógico, llamado "*stock*" o "*marketing*" en lugar de *hda6* o *sdc3*).

3) Grupos de volúmenes (VG): es el elemento de la capa superior. La unidad administrativa que engloba nuestros recursos, ya sean volúmenes lógicos (LV) o físicos (PV). En esta unidad se guardan los datos de los PV disponibles, y cómo se forman las LV a partir de los PV. Evidentemente, para poder utilizar un grupo VG, hemos de disponer de soportes físicos PV, que se organizen en diferentes unidades lógicas LV.

En la siguiente figura observamos un grupo de volúmenes, donde disponemos de siete PV, en forma de particiones de discos, que se han agrupado para formar dos volúmenes lógicos (que se han acabado utilizando para formar los sistemas de ficheros de */usr* y */home*):



Esquema de un ejemplo de LVM.

Con el uso de los volúmenes lógicos, permitimos un tratamiento más flexible del espacio en el sistema de almacenamiento (que podría tener un gran número de discos y particiones diferentes), según las necesidades que nos aparezcan. Podemos gestionar el espacio, tanto por identificadores más adecuados como por operaciones que nos permitan adecuar las necesidades al espacio disponible en cada momento.

Los sistemas de gestión de volúmenes nos permiten:

1) Redimensionar dinámicamente grupos y volúmenes lógicos, aprovechando nuevos PV, o extrayendo algunos de los disponibles inicialmente.

2) Instantáneas (*snapshots*) del sistema de archivos (lectura en LVM1 y lectura y/o escritura en LVM2). Esto facilita la creación de un nuevo dispositivo que sea una instantánea en el tiempo de la situación de una LV. Se puede, por ejemplo, crear la instantánea, montarla, probar varias operaciones o configuración nueva de software, u otros elementos, y si no funciona como esperábamos, devolver el volumen original a su estado antes de las pruebas.

3) RAID 0 de volúmenes lógicos.

En LVM no se implementan configuraciones de RAID de tipos 1 o 5, si son necesarias (o sea redundancia y tolerancia a fallo). El proceso es utilizar software de RAID o controladora hardware RAID que lo implemente en un determinado nivel de RAID, y luego colocar LVM como capa superior al RAID creado previamente.

Hagamos un breve ejemplo de creación típica (en muchos casos, el instalador de la distribución realiza un proceso parecido, si permitimos un LVM como sistema inicial de almacenamiento). Básicamente, se realiza: 1) la creación de los volumenes físicos (PV), 2) creación del grupo lógico (VG), 3) creación del volumen lógico (LV), y 4) utilización para creación de un sistema de ficheros, y posterior montaje:

1) Disponemos, por ejemplo, de tres particiones de diferentes discos. Creamos tres PV (cuidado: los pasos iniciales borran cualquier contenido de los discos) e inicializamos el contenido:

```
# dd if=/dev/zero of=/dev/sda1 bs=1k count=1
# dd if=/dev/zero of=/dev/sda2 bs=1k count=1
# dd if=/dev/zero of=/dev/sdb1 bs=1k count=1

# pvcreate /dev/sda1
Physical volume "/dev/sda1" successfully created
# pvcreate /dev/sda2
Physical volume "/dev/sda2" successfully created
# pvcreate /dev/sdb1
Physical volume "/dev/sdb1" successfully created
```

2) Colocamos en un VG creado de los diferentes PV:

```
# vgcreate grupo_discos /dev/sda1 /dev/sda2 /dev/sdb1
Volume group "grupo_discos" successfully created
```

3) Creamos la LV (en este caso, de tamaño de 1GB) a partir de los elementos que tenemos en el grupo VG (*-n* indica el nombre del volumen):

```
# lvcreate -L1G -n volumen_logico grupo_discos
lvcreate -- doing automatic backup of "grupo_discos"
lvcreate -- logical volume "/dev/grupo_discos/volumen_logico" successfully created
```

4) Y finalmente, creamos un sistema de ficheros (un *xfs* en este caso):

```
# mkfs.xfs /dev/grupo_discos/volumen_logico
```

El cual, por ejemplo, podríamos colocar de espacio de *backup*:

```
mkdir /mnt/backup
mount -t xfs /dev/grupo_discos/volumen_logico /mnt/backup
```

Y disponemos finalmente del dispositivo como un volumen lógico que implementa un sistema de ficheros.

Pongamos otro ejemplo de escenario de LVM, donde cambiamos los esquemas de disco usados por un sistema dinámicamente, porque no conocemos qué imagen final tendrá el sistema a lo largo de su historia.

Supongamos, por ejemplo, un entorno de portátil, donde tenemos de partida un esquema de particiones:

1) C:
2) D: (incluyendo datos)
3) Partición invisible de recuperación del sistema.

Analizando una posible instalación de GNU/Linux, determinamos que la mejor solución es usar la partición 2 previamente habiendo guardado los datos. Pero GNU/Linux suele necesitar como mínimo 2 particiones (*/root* y *swap*), si quisiéramos con este esquema alterarlo, deberíamos definir nuevas particiones, poner alguna como extendida, y mover el resto, o copiar particiones.

En este esquema, para evitar estos problemas decidimos generar volúmenes sobre la partición física 2, que nos permitirán colocar *a posteriori* las particiones de *root* y *swap* asociadas a los volúmenes correspondientes.

Con esto el curso de la acción sería:

1) Generamos un PV con la /dev/sda2.
2) Generamos un VG "grupo" con la PV anterior.
3) Generamos 2 volúmenes a partir del grupo anterior: LV "system" y LV "swap".

La secuencia de comandos podría ser:

```
# pvcreate /dev/sda2
# vgcreate grupo /dev/sda2
# lvcreate -n system -L <tamaño_system>
# lvcreate -n swap -L <tamaño_swap>
```

Con esto */dev/sda2* ahora se nos presentará como dos volúmenes: */dev/grupo/system* y */dev/grupo/swap* que podemos usar en una instalación de GNU/Linux para realizar las particiones *root* y *swap* (siempre que la distribución soporte instalación sobre volúmenes LVM). Con lo cual, estas particiones, una vez formateadas, y *filesystems* de *root* (posiblemente un *ext4*) y *swap*, podríamos proceder a la instalación de un sistema GNU/Linux que residirá sobre los dos volúmenes, que a su vez están sobre la partición física */dev/sda2*.

Ahora supongamos que nos cambia la imagen del sistema final, por ejemplo, a lo largo del tiempo nos damos cuenta de que queríamos utilizar el disco completo para la distribución GNU/Linux que instalamos en el pasado, con lo cual, podríamos recuperar las particiones */dev/sda1* y *sda3*.

Con LVM, esto lo podemos plantear con:

- 1) Nuevos volúmenes físicos PVs con */dev/sda1* y */dev/sda3*.
- 2) Extender el "grupo" VG con los nuevos PV.
- 3) Extender los LV con el nuevo espacio disponible, y los sistemas de ficheros existentes.

Básicamente:

```
# pvcreate /dev/sda1
# pvcreate /dev/sda3
# vgextend group /dev/sda1
# vgextend group /dev/sda3
```

Antes de desmontar los *filesystems* para las redimensiones, y utilizar un liveCD o USB, podríamos en el sistema:

```
# lvextend /dev/grupo/system +<tamaño_sda1>G
# lvextend /dev/grupo/system +<tamaño_sda3>G
```

Añadiendo así todo el tamaño disponible al LV "system", aunque no tendríamos por qué añadir todo, podríamos dejar algo, previendo nuevos volúmenes LV necesarios en el futuro, etc.

Y, finalmente, se debería redimensionar el sistema de ficheros (suponiendo que sea de tipo *ext4*).

```
# resize2fs /dev/grupo/system <tamaño total>
```

En todo caso, como medida de precaución, sería recomendable realizar un *backup* de los datos, antes de proceder a este tipo de procedimientos, ya sea de los datos sensibles a pérdidas, o directamente de los *filesystems* enteros o el mismo disco.

En el siguiente caso de uso utilizando volúmenes, en una distribución Fedora, combinaremos LVM2 con XFS para realizar operaciones de creación de volúmenes, expansión de sistemas de archivos, y uso de *snapshots* de LVM2.

Nota

Al igual que en otros casos, las operaciones de este caso son destructivas de los datos anteriores (en nuestro caso utilizando el disco */dev/sdb*).

Comenzamos examinando los dispositivos de bloque disponibles:

```
# lsblk
NAME       MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda          8:0    0   20G  0 disk
├─sda1       8:1    0  500M  0 part /boot
└─sda2       8:2    0 19,5G  0 part
  ├─fedora-root
  │          253:0    0 17,5G  0 lvm  /
  └─fedora-swap
             253:1    0    2G  0 lvm  [SWAP]
sdb          8:16   0    8G  0 disk
```

El disco */dev/sdb* no está en uso: crearemos una partición, con todo el espacio, donde borraremos los datos.

```
# fdisk /dev/sdb

Bienvenido a fdisk (util-linux 2.28).
Los cambios solo permanecerán en la memoria, hasta que decida escribirlos.
Tenga cuidado antes de utilizar la orden de escritura.



Orden (m para obtener ayuda): n
Tipo de partición
   p   primaria (0 primaria(s), 0 extendida(s), 4 libre(s))
   e   extendida (contenedor para particiones lógicas)
Seleccionar (valor predeterminado p): p
Número de partición (1-4, valor predeterminado 1): 1
Primer sector (2048-16777215, valor predeterminado 2048):
Último sector, +sectores o +tamaño{K,M,G,T,P} (2048-16777215, valor predeterminado 16777215):

Crea una nueva partición 1 de tipo 'Linux' y de tamaño 8 GiB.

Orden (m para obtener ayuda): w
Se ha modificado la tabla de particiones.
Llamando a ioctl() para volver a leer la tabla de particiones.
Se están sincronizando los discos.

# dd if=/dev/zero of=/dev/sdb1 bs=1k count=1
1+0 registros leídos
1+0 registros escritos
1024 bytes (1,0 kB, 1,0 KiB) copied, 0,00265721 s, 385 kB/s
```

Pasamos a crear un dispositivo físico, un grupo de volúmenes *volgroup*, y un volumen dentro de él con el 50 % de espacio disponible (4 GB en nuestro caso):

```
# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created.
```

```
# vgcreate volgroup /dev/sdb1
  Volume group "volgroup" successfully created

# lvcreate -L4G -n vol volgroup
  Logical volume "vol" created.
```

Creamos un sistema de ficheros XFS sobre el volumen y lo montamos en */mnt/extra*.

```
# mkfs.xfs /dev/volgroup/vol
meta-data=/dev/volgroup/vol      isize=512    agcount=4, agsize=262144 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=1, sparse=0
data     =                       bsize=4096   blocks=1048576, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=1
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0

# mkdir /mnt/extra

# mount -t xfs /dev/volgroup/vol /mnt/extra

# df -k
S.ficheros               bloques de 1K  Usados Disponibles Uso% Montado en
/dev/mapper/volgroup-vol       4184064   32960     4151104   1% /mnt/extra
```

Podemos realizar mientras tanto algunas operaciones con nuestro sistema de ficheros, como copiar archivos temporales/de prueba, para disponer de algunos archivos en el sistema de ficheros.

Procedemos a crear un volumen *snapshot* de LVM, con 1 GB asignado (¡Cuidado!, el tamaño para evitar problemas debería ser el mismo que el *filesystem* original. En este caso estamos presuponiendo que en estos momentos no llega a estar ocupado un 25 % del sistema original. Si el sistema de ficheros origen estuviera lleno deberíamos crear un *snapshot* también de 4 GB):

```
# lsblk /dev/sdb
NAME             MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdb                8:16   0    8G  0 disk
└─sdb1             8:17   0    8G  0 part
  └─volgroup-vol 253:2    0    4G  0 lvm  /mnt/extra

# lvcreate --size 1G -s -n vol_snap /dev/volgroup/vol
  Logical volume "vol_snap" created.
```

```
# lvdisplay
  --- Logical volume ---
  LV Path                /dev/volgroup/vol
  LV Name                vol
  VG Name                volgroup
  LV UUID                yJiHYl-onqI-QR6M-NK85-0Dr1-2Qre-Eu1LtB
  LV Write Access        read/write
  LV Creation host, time localhost.localdomain, 2016-07-11 13:18:09 +0200
  LV snapshot status     source of
                         vol_snap [active]
  LV Status              available
  # open                 1
  LV Size                4,00 GiB
  Current LE             1024
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:2

  --- Logical volume ---
  LV Path                /dev/volgroup/vol_snap
  LV Name                vol_snap
  VG Name                volgroup
  LV UUID                LCZqyU-aWb3-mwvV-8yzx-cS4t-KdAH-Jl31wy
  LV Write Access        read/write
  LV Creation host, time localhost.localdomain, 2016-07-11 13:26:01 +0200
  LV snapshot status     active destination for vol
  LV Status              available
  # open                 0
  LV Size                4,00 GiB
  Current LE             1024
  COW-table size         1,00 GiB
  COW-table LE           256
  Allocated to snapshot  0,00%
  Snapshot chunk size    4,00 KiB
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:5


# lsblk
NAME                   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdb                      8:16   0    8G  0 disk
```

```
└─sdb1                    8:17   0    8G  0 part
  ├─volgroup-vol-real   253:3    0    4G  0 lvm
  │ ├─volgroup-vol      253:2    0    4G  0 lvm  /mnt/extra
  │ └─volgroup-vol_snap 253:5    0    4G  0 lvm
  └─volgroup-vol_snap-cow
                        253:4    0    1G  0 lvm
    └─volgroup-vol_snap 253:5    0    4G  0 lvm
```

Creamos punto de montaje y montamos el *snap*. En este caso puede incorporarse una opción de solo lectura *-ro*, si el objetivo es solo hacer una copia de seguridad; la opción *nouuid* es necesaria, sino *mount* no dejaría montar dos sistemas de archivos XFS que tienen el mismo *uuid*. Una alternativa es realizar el comando `xfs_admin -U generate /dev/volgroup/vol_snap` para generar un nuevo *uuid*, pero es recomendable utilizar la primera opción para evitar modificar el volumen de *snapshot*.

```
# mkdir /mnt/snap
# mount -o nouuid /dev/volgroup/vol_snap /mnt/snap

# lvs
  LV       VG       Attr       LSize  Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  vol      volgroup owi-aos---  4,00g
  vol_snap volgroup swi-aos---  1,00g      vol    0,20

# lsblk
NAME                    MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdb                       8:16   0    8G  0 disk
└─sdb1                    8:17   0    8G  0 part
  ├─volgroup-vol-real   253:3    0    4G  0 lvm
  │ ├─volgroup-vol      253:2    0    4G  0 lvm  /mnt/extra
  │ └─volgroup-vol_snap 253:5    0    4G  0 lvm  /mnt/snap
  └─volgroup-vol_snap-cow
                        253:4    0    1G  0 lvm
    └─volgroup-vol_snap 253:5    0    4G  0 lvm  /mnt/snap
```

Ahora podemos realizar una copia de seguridad de nuestra partición aprovechando el *snapshot* creado (asumimos que la partición donde está */tmp* tiene espacio suficiente para esta copia de seguridad):

```
# tar -cvzf /tmp/backup_vol.tgz /mnt/snap
```

Cuando la copia de seguridad esté realizada podemos desmontar el *snapshot*, y quitarlo del sistema, para evitarnos la pérdida de prestaciones por mantener la copia original y el *snapshot* activos a la vez.

```
# umount /mnt/snap
# lvremove /dev/volgroup/vol_snap
```

```
Do you really want to remove active logical volume vol_snap? [y/n]: y
  Logical volume "vol_snap" successfully removed
```

Ahora realizamos una extensión del volumen *vol* para aumentar en 2G su tamaño:

```
# lvextend -L+2G /dev/volgroup/vol
Size of logical volume volgroup/vol changed from 4,00 GiB (1024 extents) to 6,00 GiB (1536 extents).
  Logical volume vol successfully resized.
```

Ahora podemos extender el sistema de archivos existente (en nuestro caso XFS) con:

```
# xfs_growfs /mnt/extra

# lsblk /dev/sdb
NAME             MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sdb                8:16   0    8G  0 disk
└─sdb1             8:17   0    8G  0 part
  └─volgroup-vol 253:2    0    6G  0 lvm  /mnt/extra

# df -hT /mnt/extra
S.ficheros               Tipo Tamaño Usados  Disp Uso% Montado en
/dev/mapper/volgroup-vol xfs    6,0G    33M  6,0G   1% /mnt/extra
```

9. Software: actualización

Ved también

En el apartado 1, "Herramientas básicas para el administrador", desarrollamos en profundidad estos conceptos.

Para la administración de instalación o actualización de software en nuestro sistema, vamos a depender en primera instancia del tipo de paquetes software que utilice nuestro sistema:

- **RPM**: paquetes que utiliza la distribución Fedora/Red Hat (y derivadas). Se suelen manejar mediante el comando rpm. Contienen información de dependencias del software con otros. A alto nivel la utilidad principal de actualización es yum.

- **DEB**: paquetes de Debian, se suelen manejar con un conjunto de herramientas que trabajan a diferentes niveles con paquetes individuales o grupos. Entre estas, cabe mencionar *dselect, tasksel, dpkg, apt-get* y *aptitude*.

- **Tar**, o bien los tgz (también *tar.gz*): son puramente paquetes de ficheros unidos y comprimidos mediante comandos estándar como *tar* y *gzip* (se usan los mismos para la descompresión). Estos paquetes no contienen información de dependencias y pueden instalarse en diferentes lugares, si no es que llevan información de ruta (*path*) absoluta.

Para manejar estos paquetes existen varias herramientas gráficas, como RPM: *Kpackage;* DEB: *Synaptic, Gnome*-apt; Tgz: *Kpackage* o desde el propio gestor de ficheros gráficos (en Gnome o KDE). También suelen existir utilidades de conversiones de paquetes. Por ejemplo, en Debian tenemos el comando *alien,* que permite convertir paquetes RPM a DEB. Aunque hay que tomar las debidas precauciones para que el paquete, por tener una distribución destino diferente, no modifique algún comportamiento o fichero de sistema no esperado.

Dependiendo del uso de los tipos de paquetes o herramientas, la actualización o instalación de software de nuestro sistema se podrá producir de diferentes maneras:

1) Desde los propios CD de instalación del sistema. Todas las distribuciones buscan el software en sus CD. Pero hay que tener en cuenta que este software no sea antiguo y no incluya, por esta razón, algunos parches como actualizaciones, o nuevas versiones con más prestaciones, con lo cual, si se instala a partir de CD, es bastante común verificar después que no exista alguna versión más reciente disponible en los repositorios oficiales de la distribución, o en alternativos que proporcionen software adicional (siempre con las medidas de seguridad oportunas para establecer que sean repositorios de confianza).

2) Mediante servicios de actualización o búsqueda de software, ya sea de forma gratuita, como el caso de la herramienta *apt-get* de Debian, o yum/dnf en Fedora, o servicios de suscripción (de pago o con facilidades básicas), como el Red Hat Network de las versiones Red Hat comerciales.

3) Por repositorios de software que ofrecen paquetes de software preconstruidos para una distribución determinada.

4) Por el propio creador o distribuidor del software, que ofrece una serie de paquetes de instalación de su software. Podemos no encontrar el tipo de paquetes necesario para nuestra distribución.

5) Software sin empaquetamiento o con un empaquetamiento de solo compresión sin ningún tipo de dependencias.

6) solo código fuente, en forma de paquete o bien fichero comprimido.

En las actualizaciones basadas en puntos como 3, 4, 5 y 6, es imprescindible mantener cierta seguridad en la distribución, para asegurarse que son paquetes "oficiales", o de una fuente de reputación conocida, así como comprobar la autenticidad de los paquetes mediante firmas o comprobaciones de *hash*. Asimismo, es posible que los paquetes dispongan de actualizaciones frecuentes que tendremos que integrar periódicamente, por ejemplo, las diferentes correcciones o updates, relacionados con seguridad, con correcciones de vulnerabilidades detectadas.

Hay que recordar, además, que Snap/Flatpack, como nuevos formatos de empaquetado nos permitirían instalar paquetes de software en principio independientes de la distribución, y que podremos incorporar a cualquier distribución que soporte el sistema utilizado (sea Snappy o Flatpak). Cabrá esperar cuál es la evolución futura de estos sistemas de empaquetado "universal".

10. Trabajos no interactivos

En las tareas de administración, suele ser necesaria la ejecución a intervalos temporales de ciertas tareas, ya sea por programar las tareas para realizarlas en horarios de menor uso de la máquina, o bien por la propia naturaleza periódica de las tareas que se quieran desarrollar.

Para realizar este tipo de trabajos "fuera de horas", como servicios periódicos o programados, hay varios sistemas que nos permiten construir un tipo de agenda de tareas (planificación de ejecución de tareas):

- ***nohup***: es quizás el caso más simple utilizado por los usuarios. Les permite la ejecución de una tarea no interactiva una vez hayan salido de su cuenta. Al salir de la cuenta, el usuario pierde sus procesos, y *nohup* permite dejarlos en ejecución, a pesar de que el usuario se desconecte.

- ***at***: nos deja lanzar una acción para más tarde, programando un determinado instante en el que va a iniciarse, especificándose la hora (*hh:mm*) y fecha, o bien si se hará hoy (*today*) o mañana (*tomorrow*).

```
# at 10pm tarea
```

realizar la tarea a las diez de la noche.

```
# at 2am tomorrow tarea
```

realizar la tarea a las dos de la madrugada.

- ***cron***: permite establecer una lista de trabajos por realizar con su programación; esta configuración se guarda en */etc/crontab*. Concretamente, en cada entrada de este fichero tenemos: minutos y hora en que se va a efectuar la tarea, qué día del mes, qué mes, qué día de la semana, junto con qué (ya sea una tarea, o bien un directorio donde estarán las tareas a ejecutar).

De forma estándar, el contenido es parecido a:

```
25 6 * * * root test -e /usr/sbin/anacron ||
run-parts --report /etc/cron.daily
47 6 * * 7 root test -e /usr/sbin/anacron ||
run-parts --report /etc/cron.weekly
52 6 1 * * root test -e /usr/sbin/anacron ||
run-parts --report /etc/cron.monthl
```

donde se está programando que una serie de tareas van a hacerse: cada día ("*" indica "cualquiera"), semanalmente (el séptimo día de la semana), o mensualmente (el primero de cada mes). Los trabajos serían ejecutados por el comando *crontab,* pero el sistema *cron* supone que la máquina está siempre encendida. Si no es así, es mejor utilizar *anacron,* que verifica si la acción no se realizó cuando habría debido hacerlo, y la ejecuta. En cada línea del anterior fichero se verifica que exista el comando *anacron* y se ejecutan los *scripts* asociados a cada acción, que en este caso están guardados en unos directorios asignados para ello, los *cron*.

También pueden existir unos ficheros *cron.allow, cron.deny*, para limitar quién puede colocar (o no) trabajos en *cron*. Mediante el comando *crontab,* un usuario puede definir trabajos en el mismo formato que hemos visto antes, que se guardarán en */var/spool/cron/crontabs*. En algunos casos, existe también un directorio */etc/cron.d* donde se pueden colocar trabajos y que es tratado como si fuera una extensión del fichero */etc/crontab*. En algunas versiones, el sistema ya especifica sus trabajos periódicos de sistema directamente sobre subdirectorios del */etc* como *cron.hourly/, cron.daily/, cron.weekly y cron.montly/,* donde se colocan los trabajos de sistema que necesitan esta periodicidad.

En sistemas con arranque systemd, este ofrece *systemd-cron*, como un servicio basado en sus posibilidades de temporizadores y calendarios, que le permite usar los directorios estandard *cron.hourly*, *daily*, *monthly* y con *$ systemctl start crond.service* o *stop*, podemos arrancar o parar el servicio. Dependiendo de la distribución y el soporte de systemd, se utiliza esta capa de compatibilidad con el *cron* clásico, o pueden utilizarse la definición de *timers* propios de systemd, para crear tareas usando directamente systemd únicamente. Se recomienda consultar las páginas man de la distribución, referentes a systemd, *systemd.timer*, *cron* y *crond* para conocer los métodos habilitados.

11. Taller: prácticas combinadas de los apartados

Comenzaremos por examinar el estado general de nuestro sistema. Vamos a realizar los diferentes pasos en un sistema Debian. Aunque se trata de un sistema Debian, los procedimientos son en su mayor parte trasladables a otras distribuciones, como Fedora/Red Hat (mencionaremos algunos de los cambios más importantes). El hardware consiste en una antigua máquina Pentium 4 a 2.66Mhz con 768 MB y varios discos, DVD y grabador de CD, además de otros periféricos, pero ya iremos obteniendo esta información paso a paso.

En este caso particular hemos seleccionado una máquina bastante antigua para los estándares actuales, precisamente una de las versatilidades de GNU/Linux nos permite recuperar máquinas obsoletas (según otros entornos operativos), y convertirlas en funcionales para tareas determinadas: servidores de ficheros, impresoras, servidores web, *streaming* audio/vídeo, etc.

Veamos primero cómo ha arrancado nuestro sistema la última vez:

```
# uptime

17:38:22 up 2:46, 5 users, load average: 0.05, 0.03, 0.04
```

Este comando nos da el tiempo que lleva el sistema funcionando desde que se arrancó la última vez, 2 horas 46 minutos. En nuestro caso tenemos cinco usuarios. Estos no tienen por qué ser cinco usuarios diferentes, sino que serán las sesiones de usuario abiertas (por ejemplo, mediante un terminal). El comando *who* permite listar estos usuarios. El *load average* es la carga media del sistema en los últimos 1, 5 y 15 minutos.

Veamos el log del arranque del sistema (comando *dmesg*), las líneas que se iban generando en la carga del sistema (se han suprimido diferentes líneas por claridad):

```
Linux version 2.6.20-1-686 (Debian 2.6.20-2) (waldi@debian.org) (gcc version
4.1.2 20061115 (prerelease) (Debian 4.1.1-21)) #1 SMP Sun Apr 15 21:03:57 UTC
BIOS-provided physical RAM map:
 BIOS-e820: 0000000000000000 - 000000000009f800 (usable)
 BIOS-e820: 000000000009f800 - 00000000000a0000 (reserved)
 BIOS-e820: 00000000000ce000 - 00000000000d0000 (reserved)
 BIOS-e820: 00000000000dc000 - 0000000000100000 (reserved)
 BIOS-e820: 0000000000100000 - 000000002f6e0000 (usable)
 BIOS-e820: 000000002f6e0000 - 000000002f6f0000 (ACPI data)
 BIOS-e820: 000000002f6f0000 - 000000002f700000 (ACPI NVS)
 BIOS-e820: 000000002f700000 - 000000002f780000 (usable)
```

```
 BIOS-e820: 000000002f780000 - 0000000030000000 (reserved)
 BIOS-e820: 00000000ff800000 - 00000000ffc00000 (reserved)
BIOS-e820: 00000000fffffc00 - 0000000100000000 (reserved)
0MB HIGHMEM available.

759MB LOWMEM available.
```

Estas primeras líneas ya nos indican varios datos interesantes: la versión del *kernel* Linux es la 2.6.20-1-686, una versión 2.6 revisión 20 a revisión 1 de Debian, y para máquinas 686 (arquitectura Intel x86 32bits). Indica también que estamos arrancando un sistema Debian, con este *kernel* que fue compilado con un compilador GNU gcc versión 4.1.2. A continuación, existe un mapa de zonas de memoria usadas (reservadas) por la BIOS, y a continuación el total de memoria detectada en la máquina: 759 MB, a las que habría que sumar el primer 1 MB, total de 760 MB.

```
Kernel command line: BOOT_IMAGE=LinuxNEW ro root=302 lang=es acpi=force
Initializing CPU#0
Console: colour dummy device 80x25
Memory: 766132k/777728k available (1641k kernel code, 10968k reserved, 619k data,
208k init, 0k highmem)
Calibrating delay using timer specific routine.. 5320.63 BogoMIPS (lpj=10641275)
```

Aquí nos refiere cómo ha sido el arranque de la máquina, qué línea de comandos se le ha pasado al *kernel* (pueden pasarse diferentes opciones, por ejemplo desde el *lilo* o *grub*). Y estamos arrancando en modo consola de 80 x 25 caracteres (esto se puede cambiar). Los *BogoMIPS* son una medida interna del *kernel* de la velocidad de la CPU; hay arquitecturas donde es difícil detectar a cuántos Mhz o GHz funciona la CPU, y por eso se utiliza esta medida de velocidad. Después nos da más datos de la memoria principal, y nos dice para qué se está usando en este momento del arranque.

```
CPU: Trace cache: 12K uops, L1 D cache: 8K
CPU: L2 cache: 512K
CPU: Hyper-Threading is disabled
Intel machine check architecture supported.
Intel machine check reporting enabled on CPU#0.
CPU0: Intel P4/Xeon Extended MCE MSRs (12) available
CPU0: Intel(R) Pentium(R) 4 CPU 2.66GHz stepping 09
```

Además, nos proporciona datos varios de la CPU, el tamaño de las caché de primer nivel, caché interna CPU, L1 dividida en una *TraceCache* del Pentium4 (o *Instruction Cache*), y la caché de datos, y caché unificada de segundo nivel (*L2*), tipo de CPU, velocidad de esta y del bus del sistema.

```
PCI: PCI BIOS revision 2.10 entry at 0xfd994, last bus=3
Setting up standard PCI resources
```

```
...
NET: Registered protocol
IP route cache hash table entries: 32768 (order: 5, 131072 bytes)
TCP: Hash tables configured (established 131072 bind 65536)
checking if image is initramfs... it is
Freeing initrd memory: 1270k freed
fb0: VESA VGA frame buffer device
Serial: 8250/16550 driver $Revision: 1.90 $ 4 ports, IRQ sharing enabled
serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
00:09: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A
RAMDISK driver initialized: 16 RAM disks of 8192K size 1024 blocksize
PNP: PS/2 Controller [PNP0303:KBC0,PNP0f13:MSE0] at 0x60,0x64 irq 1,12
i8042.c: Detected active multiplexing controller, rev 1.1.
serio: i8042 KBD port at 0x60,0x64 irq 1
serio: i8042 AUX0 port at 0x60,0x64 irq 12
serio: i8042 AUX1 port at 0x60,0x64 irq 12
serio: i8042 AUX2 port at 0x60,0x64 irq 12
serio: i8042 AUX3 port at 0x60,0x64 irq 12
mice: PS/2 mouse device common for all mice
```

Continúan las inicializaciones del *kernel* y dispositivos, menciona la inicialización de protocolos de red. Los terminales, los puertos serie *ttyS0* (sería el *com1*), *ttyS01* (el *com2*), da información de los discos RAM que usamos y detecta dispositivos PS2, teclado y ratón.

```
ICH4: IDE controller at PCI slot 0000:00:1f.1


   ide0: BM-DMA at 0x1860-0x1867, BIOS settings: hda:DMA, hdb:pio
   ide1: BM-DMA at 0x1868-0x186f, BIOS settings: hdc:DMA, hdd:pio
Probing IDE interface ide0...
hda: FUJITSU MHT2030AT, ATA DISK drive
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
Probing IDE interface ide1...
hdc: SAMSUNG CDRW/DVD SN-324F, ATAPI CD/DVD-ROM drive
ide1 at 0x170-0x177,0x376 on irq 15
SCSI subsystem initialized
libata version 2.00 loaded.
hda: max request size: 128KiB
hda: 58605120 sectors (30005 MB) w/2048KiB Cache, CHS=58140/16/63<6>hda: hw_config=600b
, UDMA(100)
hda: cache flushes supported
hda: hda1 hda2 hda3
kjournald starting. Commit interval 5 seconds
EXT3-fs: mounted filesystem with ordered data mode.
hdc: ATAPI 24X DVD-ROM CD-R/RW drive, 2048kB Cache, UDMA(33)
Uniform CD-ROM driver Revision: 3.20
```

```
Addinf 618492 swap on /dev/hda3.
```

Detección de dispositivos IDE, detecta el chip IDE en el bus PCI e informa de que está controlando dos dispositivos: *hda* y *hdc*, que son respectivamente un disco duro (fujitsu), un segundo disco duro, un DVD Samsung y una grabadora de CD (ya que en este caso se trata de una unidad combo). Indica particiones activas en el disco. Más adelante, detecta el sistema principal de ficheros de Linux, un *ext3* con *journal*, que activa y añade el espacio de *swap* disponible en una partición.

```
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
input: PC Speaker as /class/input/input1
USB Universal Host Controller Interface driver v3.0
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
uhci_hcd 0000:00:1d.1: UHCI Host Controller
uhci_hcd 0000:00:1d.1: new USB bus registered, assigned bus number 2
uhci_hcd 0000:00:1d.1: irq 11, io base 0x00001820
usb usb2: configuration #1 chosen from 1 choice
hub 2-0:1.0: USB hub found
hub 2-0:1.0: 2 ports detected
hub 4-0:1.0: USB hub found
hub 4-0:1.0: 6 ports detected
```

Más detección de dispositivos, USB en este caso (y los módulos que corresponden). Ha detectado dos dispositivos *hub* (con un total de 8 USB *ports*).

```
parport: PnPBIOS parport detected.
parport0: PC-style at 0x378 (0x778), irq 7, dma 1 [PCSPP,TRISTATE,COMPAT,EPP,ECP,DMA]
input: ImPS/2 Logitech Wheel Mouse as /class/input/input2
ieee1394: Initialized config rom entry `ip1394'
eepro100.c:v1.09j-t 9/29/99 Donald Becker
Synaptics Touchpad, model: 1, fw: 5.9, id: 0x2e6eb1, caps: 0x944713/0xc0000
input: SynPS/2 Synaptics TouchPad as /class/input/input3


agpgart: Detected an Intel 845G Chipset
agpgart: Detected 8060K stolen Memory
agpgart: AGP aperture is 128M
eth0: OEM i82557/i82558 10/100 Ethernet, 00:00:F0:84:D3:A9, IRQ 11.
Board assembly 000000-000, Physical connectors present: RJ45
e100: Intel(R) PRO/100 Network Driver, 3.5.17-k2-NAPI
usbcore: registered new interface driver usbkbd
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
```

```
lp0: using parport0 (interrupt-driven).

ppdev: user-space parallel port driver
```

Y la detección final del resto de dispositivos: puerto paralelo, modelo de ratón, puerto *firewire* (ieee1394) tarjeta de red (Intel), un panel táctil, la tarjeta de vídeo AGP (i845). Más datos de la tarjeta de red una intel pro 100, registro de usb como *mass storage* (indica un dispositivo de almacenamiento por usb, como un disco externo) y detección del puerto paralelo.

Toda esta información que hemos visto mediante el comando *dmesg,* también la podemos encontrar volcada en el log principal del sistema */var/log/*messages. En este log, entre otros, encontraremos los mensajes del *kernel* y de los *daemons*, y errores de red o dispositivos, los cuales comunican sus mensajes a un *daemon* especial llamado *syslogd,* que es el encargado de escribir los mensajes en este fichero. Si hemos arrancado la máquina recientemente, observaremos que las últimas líneas contienen exactamente la misma información que el comando *dmesg,* por ejemplo, si nos quedamos con la parte final del fichero (suele ser muy grande):

```
tail 200 /var/log/messages
```

Observamos las líneas de antes, y también algunas informaciones más, como por ejemplo:

```
shutdown[13325]: shutting down for system reboot
kernel: usb 4-1: USB disconnect, address 3
kernel: nfsd: last server has exited
kernel: nfsd: unexporting all filesystems
kernel: Kernel logging (proc) stopped.
kernel: Kernel log daemon terminating.

exiting on signal 15
syslogd 1.4.1#20: restart.

kernel: klogd 1.4.1#20, log source = /proc/kmsg started.
Linux version 2.6.20-1-686 (Debian 2.6.20-2) (waldi@debian.org) (gcc version
4.1.2 20061115 (prerelease) (Debian 4.1.1-21)) #1 SMP Sun Apr 15 21:03:57

kernel: BIOS-provided physical RAM map:
```

La primera parte corresponde a la parada anterior del sistema; nos informa de que el *kernel* ha dejado de colocar información en */proc*, se está parando el sistema... Al principio del arranque nuevo, se activa el *daemon Syslogd* que genera el log y comienza la carga del sistema, que nos dice que el *kernel* comenzará a

escribir información en su sistema, */proc*. Vemos las primeras líneas de *dmesg* de mención de la versión que se está cargando de *kernel*, y luego encontraremos lo que hemos visto con *dmesg*.

En este punto, otro comando útil para saber cómo se ha producido la carga es *lsmod*, que nos permitirá saber qué módulos dinámicos se han cargado con el *kernel* (versión resumida):

```
# lsmod
Module            Size      Used by
nfs               219468    0
nfsd              202192    17
exportfs          5632      1 nfsd
lockd             58216     3 nfs,nfsd
nfs_acl           3616      2 nfs,nfsd
sunrpc            148380    13 nfs,nfsd,lockd,nfs_acl
ppdev             8740      0
lp                11044     0
button            7856      0
ac                5220      0
battery           9924      0
md_mod            71860     1
dm_snapshot       16580     0
dm_mirror         20340     0
dm_mod            52812     2 dm_snapshot,dm_mirror
i810fb            30268     0
vgastate          8512      1 i810fb
eeprom            7184      0
thermal           13928     0
processor         30536     1 thermal
fan               4772      0
udf               75876     0
ntfs              205364    0
usb_storage       75552     0
hid               22784     0
usbkbd            6752      0
eth1394           18468     0
e100              32648     0
eepro100          30096     0
ohci1394          32656     0
ieee1394          89208     2 eth1394,ohci1394
snd_intel8x0      31420     1
snd_ac97_codec    89412     1 snd_intel8x0
ac97_bus          2432      1 snd_ac97_codec
parport_pc        32772     1
snd               48196     6 snd_intel8x0,snd_ac97_codec,snd_pcm,snd_timer
ehci_hcd          29132     0
```

```
ide_cd            36672    0
cdrom             32960    1 ide_cd
soundcore         7616     1 snd
psmouse           35208    0
uhci_hcd          22160    0
parport           33672    3 ppdev,lp,parport_pc
intelfb           34596    0
serio_raw         6724     0
pcspkr            3264     0
pci_hotplug       29312    1 shpchp
usbcore           122312   6 dvb_usb,usb_storage,usbkbd,ehci_hcd,uhci_hcd
intel_agp         22748    1
agpgart           30504    5 i810fb,drm,intelfb,intel_agp
ext3              121032   1
jbd               55368    1 ext3
ide_disk          15744    3
ata_generic       7876     0
ata_piix          15044    0
libata            100052   2 ata_generic,ata_piix
scsi_mod          133100   2 usb_storage,libata
generic           4932     0 [permanent]
piix              9540     0 [permanent]
ide_core          114728   5 usb_storage,ide_cd,ide_disk,generic,piix
```

Vemos que disponemos de los controladores para el hardware que hemos detectado, y de otros relacionados o necesarios por dependencias.

Ya tenemos, pues, una idea de cómo se han cargado el *kernel* y sus módulos. En este proceso puede que ya hubiésemos observado algún error; si hay hardware mal configurado o módulos del *kernel* mal compilados (no eran para la versión del *kernel* adecuada), inexistentes, etc.

El paso siguiente será la observación de los procesos en el sistema, con el comando *ps* (Process Status), por ejemplo (solo se han listado los procesos de sistema, no los de los usuarios):

```
#ps -ef

UID PID PPID C STIME TTY TIME CMD
```

Información de los procesos, *UID*, usuario que ha lanzado el proceso (o con qué identificador se ha lanzado), *PID*, código del proceso asignado por el sistema, son consecutivos a medida que se lanzan los procesos. El primero siempre es el 0, que corresponde al proceso de *init*. *PPID* es el id del proceso padre del actual. *STIME*, tiempo en que fue arrancado el proceso, *TTY*, terminal asignado al proceso (si tiene alguno), *CMD*, línea de comando con que fue lanzado.

```
root     1    0 0 14:52 ?     00:00:00 init [2]
root     3    1 0 14:52 ?     00:00:00 [ksoftirqd/0]
root    143   6 0 14:52 ?     00:00:00 [bdflush]
root    145   6 0 14:52 ?     00:00:00 [kswapd0]
root    357   6 0 14:52 ?     00:00:01 [kjournald]
root    477   1 0 14:52 ?     00:00:00 udevd --daemon
root    719   6 0 14:52 ?     00:00:00 [khubd]
```

Varios *daemons* de sistema, como *kswapd daemon*, que controla el intercambio de páginas con memoria virtual. Gestión de *buffers* del sistema (*bdflush*). Gestión de *journal* de *filesystem* (*kjournald*), gestión de USB (*khubd*). O el demonio de *udev* que controla la conexión en caliente de dispositivos. En general (no siempre) los *daemons* suelen identificarse por una *d* final, y si llevan un *k* inicial, normalmente son hilos (*threads*) internos del *kernel*.

```
root    1567  1 0 14:52 ?     00:00:00 dhclient -e -pf ...
root    1653  1 0 14:52 ?     00:00:00 /sbin/portmap
root    1829  1 0 14:52 ?     00:00:00 /sbin/syslogd
root    1839  1 0 14:52 ?     00:00:00 /sbin/klogd -x
root    1983  1 0 14:52 ?     00:00:09 /usr/sbin/cupsd
root    2178  1 0 14:53 ?     00:00:00 /usr/sbin/inetd
```

Tenemos: *dhclient*, lo que indica que esta máquina es cliente de un servidor DHCP, para obtener su IP; *Syslogd*, *daemon* que envía mensajes al log; el *daemon* de *cups*, como vimos, está relacionado con el sistema de impresión; e *inetd*, que, como veremos en la parte de redes, es una especie de "superservidor" o intermediario de otros *daemons* relacionados con servicios de red.

```
root      2154  1 0 14:53 ?     00:00:00 /usr/sbin/rpc.mountd
root      2241  1 0 14:53 ?     00:00:00 /usr/sbin/sshd
root      2257  1 0 14:53 ?     00:00:00 /usr/bin/xfs -daemon
root      2573  1 0 14:53 ?     00:00:00 /usr/sbin/atd
root      2580  1 0 14:53 ?     00:00:00 /usr/sbin/cron
root      2675  1 0 14:53 ?     00:00:00 /usr/sbin/apache
www-data  2684 2675 0 14:53 ?   00:00:00 /usr/sbin/apache
www-data  2685 2675 0 14:53 ?   00:00:00 /usr/sbin/apache
```

También está *sshd*, servidor de acceso remoto seguro (una versión mejorada que permite servicios compatibles con telnet y ftp); *xfs* es el servidor de fuentes (tipos de letra) de X Window. Los comandos *atd* y *cron* sirven para manejar tareas programadas en un momento determinado. Apache es el servidor web, que puede tener varios hilos activos (*threads*) para atender diferentes peticiones.

```
root    2499 2493 0 14:53 ?         00:00:00 /usr/sbin/gdm
root    2502 2499 4 14:53 tty7      00:09:18 /usr/bin/X :0 -dpi 96 ...
root    2848   1  0 14:53 tty2      00:00:00 /sbin/getty 38400 tty2
```

```
root    2849   1  0 14:53 tty3      00:00:00 /sbin/getty 38400 tty3
root    3941 2847 0 14:57 tty1      00:00:00 -bash
root    16453 12970 0 18:10 pts/2   00:00:00 ps -ef
```

Así, *gdm* es el login gráfico del sistema de escritorio Gnome (la entrada que nos pide el *login* y contraseña); los procesos *getty* son los que gestionan los terminales virtuales de texto (los que podemos ver con las teclas Alt+Fn, o Ctrl +Alt+Fn si estamos en modo gráfico); *X* es el proceso de servidor gráfico de X Window System, imprescindible para que se ejecute cualquier entorno de escritorio por encima de él. Un *shell* abierto (*bash*), y finalmente el proceso que hemos generado al pedir este *ps* desde la línea de comandos.

El comando *ps* tiene muchas opciones de línea de comandos para ajustar la información que queremos de cada proceso, ya sea tiempo que hace que se ejecuta, porcentaje de CPU usado, memoria utilizada, etc. (ved man de *ps*). Otro comando muy interesante es *top*, que hace lo mismo que *ps* pero de forma dinámica; se actualiza cada cierto intervalo; podemos clasificar los procesos por uso de CPU, de memoria, y también nos da información del estado de la memoria global.

Nota

Ver man de los comandos para interpretar las salidas.

Otros comandos útiles para uso de recursos son *free* y *vmstat*, que nos dan información sobre la memoria utilizada y el sistema de memoria virtual:

```
# free
             total       used       free     shared    buffers     cached
Mem:        767736     745232      22504          0      89564     457612
-/+ buffers/cache:     198056     569680
Swap:       618492       1732     616760

# vmstat
procs -----------memory----------   ---swap-- -----io-- --system-- ----cpu----
r  b   swpd free     buff     cache   si so bi bo    in   cs us sy   id wa
1  0   1732 22444    89584    457640 0  0  68 137   291  418 7 1    85 7
```

En el comando *free* también se puede observar el tamaño del *swap* presente, aproximadamente de unas 600 MB, que de momento no se está usando intensamente por tener suficiente espacio de memoria física. Todavía quedan unas 22 MB libres (lo que indica una alta utilización de la memoria física y un uso de *swap* próximamente). El espacio de memoria y el *swap* (a partir de los *kernels* 2.4) son aditivos para componer el total de memoria del sistema, haciendo en este caso un total de 1,4 GB de memoria disponible.

Actividades

1. Haced una lectura básica del estándar FHS, que nos servirá para tener una buena guía a la hora de buscar archivos por nuestra distribución.

Nota

Ver FHS en:

http://www.pathname.com/fhs

2. Para los paquetes RPM, ¿cómo haríais algunas de las siguientes tareas?

a) Conocer qué paquete instaló un determinado comando.

b) Obtener la descripción del paquete que instaló un comando.

c) Borrar un paquete cuyo nombre completo no conocemos.

d) Mostrar todos los archivos que estaban en el mismo paquete que un determinado archivo.

3. Efectuad las mismas tareas que en la actividad anterior, pero para paquetes Debian, usando herramientas *APT*.

4. Actualizad una distribución Debian (o Fedora).

5. Instalad en nuestra distribución alguna herramienta genérica de administración, como *Webmin*. ¿Qué nos ofrece? ¿Es fácil comprender las tareas ejecutadas y los efectos que provocan?

6. El espacio de *swap* permite complementar la memoria física para disponer de mayor memoria virtual. Dependiendo de los tamaños de memoria física y *swap*, ¿puede llegar a agotarse la memoria? ¿Podemos solucionarlo de otro modo, que no sea añadiendo más memoria física?

7. Supongamos que tenemos un sistema con dos particiones Linux, una */* y la otra de *swap*. ¿Cómo solucionaríais el caso de que las cuentas de los usuarios agotasen el espacio de disco? Y en el caso de tener una partición */home* aislada que se estuviera agotando, ¿cómo lo solucionaríais?

8. Instalad el sistema de impresión CUPS, definid nuestra impresora para que funcione con CUPS y probad la administración vía interfaz web. Tal como está el sistema, ¿sería recomendable modificar de algún modo la configuración que trae por defecto CUPS? ¿Por qué?

9. Analizad el sistema Upstart, o systemd, presentes en una distribución Debian, Ubuntu o Fedora. ¿Qué eventos y *jobs* trae predefinidos? ¿Hay compatibilidad con el *init* de SystemV (*sysvinit*)?

10. Examinad la configuración por defecto que traiga el sistema GNU/Linux de trabajos no interactivos por *cron* (o systemd-cron). ¿Qué trabajos y cuándo se están realizando? ¿Alguna idea para nuevos trabajos que haya que añadir?

11. Reproducid el análisis del taller (más los otros apartados de la unidad) sobre una máquina disponible; ¿se observan en el sistema algunos errores o situaciones anómalas? En tal caso, ¿cómo orientamos su resolución?

Bibliografía

[Bai03] Bailey, E. C. (2003). RedHat Maximum RPM. Ofrece una amplia visión de los sistemas de paquetes de software de las distribuciones Debian y Fedora/Red Hat.

[Bas] Mike, G. "BASH Programming - Introduction HOWTO". *The Linux Documentation Project*.Ofrece una amplia introducción (y conceptos avanzados) a la programación de *shell scripts* en *bash*, así como numerosos ejemplos.

[Coo] Cooper, M. (2006). "Advanced bashScripting Guide". *The Linux Documentation Project* (guías). Ofrece una amplia introducción (y conceptos avanzados) a la programación de *shell scripts* en *bash*, así como numerosos ejemplos.

[Deb] Debian. "Sitio Seguridad de Debian". Ofrece una amplia visión de los sistemas de paquetes de software de las distribuciones Debian y Fedora/Red Hat.

[Debb] Comunidad Debian. "Distribución Debian".

[Fed] The Fedora Project. <http://fedoraproject.org>

[Fri02] Frisch, A. (2002). *Essential System Administration*. O'Reilly. Administración de GNU/Linux y UNIX. Comenta de forma amplia aspectos de administración local y gestión de sistemas de impresión.

[Gt] Taylor, G.; Allaert, D. "The Linux Printing HOWTO". *The Linux Documentation Project*. Ofrece información actualizada de los sistemas de impresión y su configuración, así como detalles de algunas impresoras. Para detalles concretos de modelos de impresora y controladores, podéis dirigiros a <http://www.linuxprinting.org/>

[Hin00] Hinner, M. "Filesystems HOWTO". *The Linux Documentation Project*. Información sobre los diferentes sistemas de ficheros disponibles y los esquemas de creación de particiones para la instalación del sistema.

[Koe] Koehntopp, K. "Linux Partition HOWTO". *The Linux Documentation Project*. Información sobre los diferentes sistemas de ficheros disponibles y los esquemas de creación de particiones para la instalación del sistema.

[Lin04b] (2004). *Filesystem Hierarchy Standard (FHS)*. http://www.pathname.com/fhs

[Lin11c] Linux Standard Base (specifications Archive).

[Linc] Linux Standards Base project. <http://www.linux-foundation.org/en/LSB>

[Mor03] Morill, D. (2003). *Configuración de sistemas Linux*. Anaya Multimedia.

[Nem06] Nemeth, E.; Snyder, G.; Hein, T. R. (2006). "Linux Administration Handbook" (2.ª ed.). Prentice Hall. Trata de forma amplia la mayoría de aspectos de administración y es una buena guia genérica para cualquier distribución.

[Pow12] Powell, Howard (2012). "ZFS and Btrfs: a Quick introduction to Modern Filesystems". *Linux Journal* (issue 218, pág. 104-111).

[Qui01] Quigley, E. (2001). *Linux shells by Example*. Prentice Hall. Comenta los diferentes *shells* de programación en GNU/Linux, así como sus semejanzas y diferencias.

[Sob10] Sobell, M. G. (2010). *A Practical Guide to Fedora and Red Hat Enterprise Linux*. Prentice Hall. Es una buena guía de administración local para distribuciones Red Hat y Fedora.

[SM02] Schwartz, M. y otros (2002). *Multitool Linux - Practical Uses for Open Source Software*. Addison Wesley.

[Smi02] Smith, R. (2002). *Advanced Linux Networking*. Addison Wesley. Administración de GNU/Linux y UNIX. Comenta de forma amplia aspectos de administración local y gestión de sistemas de impresión.

[Soy12] Soyinka, W. (2012). *Linux Administration. A Beginner's Guide* (6th Edition). McGraw Hill.

[Stu] Stutz, M. "The Linux Cookbook: Tips and Techniques for Everyday Use". *The Linux Documentation Project* (guías). Es una amplia introducción a las herramientas disponibles en GNU/Linux.

[War13] Ward, Allan (2013). "How To Use the Logic Volume Manager". *Full Circle Magazine* (issue #80, pág. 12-18).

[Wm02] Welsh, M. y otros (2002). *Running Linux 4th edition*. O'Reilly. Administración de GNU/Linux y UNIX. Comenta de forma amplia aspectos de administración local y gestión de sistemas de impresión.