Administración avanzada de GNU/Linux

Josep Jorba Esteve
Remo Suppi Boldrito

XP07/M2103/02279

UOC
www.uoc.edu

Josep Jorba Esteve

Ingeniero superior y doctor en Informática por la UAB. Profesor de los Estudios de Informática, Multimedia y Telecomunicaciones de la UOC.

Remo Suppi Boldrito

Ingeniero en Telecomunicaciones. Doctor en Informática por la UAB. Profesor del Departamento de Arquitectura de Computadores y Sistemas Operativos UAB.

Primera edición: septiembre 2007
Fundació per a la Universitat Oberta de Catalunya
Av. Tibidabo, 39-43, 08035 Barcelona
Material realizado por Eureca Media, SL

Depósito legal: B-31.591-2007

Agradecimientos

Los autores agradecen a la Fundación para la Universitat Oberta de Catalunya (http://www.uoc.edu) la financiación de la primera edición de esta obra y las posteriores revisiones realizadas, enmarcada en el máster internacional de Software Libre ofrecido por la citada institución.

Presentación

Los sistemas GNU/Linux han llegado a un grado de madurez importante, que los hacen válidos para integrarlos en cualquier ambiente de trabajo, ya sea desde el escritorio del PC personal, hasta el servidor de una gran empresa.

El objetivo principal de este curso es introducirnos en el mundo de la administración de los sistemas GNU/Linux.

Aprenderemos cómo proporcionar desde GNU/Linux los servicios necesarios a diferentes ambientes de usuarios y máquinas. El campo de la administración de sistemas es enorme, hay muchas tareas, muchos problemas por tratar, hay que tener grandes conocimientos de hardware y software, y no está de más un poco de psicología para tratar con los usuarios finales de los sistemas.

El curso no pretende abordar una distribución GNU/Linux particular, pero se han escogido un par de ellas para tratar los ejemplos: Debian y Fedora (y derivadas de Red Hat). Respecto al campo de la administración, ésta se intentará gestionar desde el nivel más bajo posible, normalmente la línea de comandos y los ficheros de configuración. Se comentarán, en su caso, herramientas de más alto nivel, pero hay que tener cuidado con estas últimas, ya que suelen ser fuertemente dependientes de la distribución utilizada e incluso de la versión de ésta; además, estas herramientas suelen variar mucho entre versiones. La administración de bajo nivel suele ser mucho más dura, pero sabemos qué estamos haciendo y dónde podemos ver los resultados, además de que nos aporta muchos conocimientos extra sobre las diferentes tecnologías utilizadas.

Las distribuciones escogidas han sido: Debian GNU/Linux Etch (4.0), y Fedora Core 7 (basada en Red Hat), utilizadas en el momento de revisar este documento (la primera edición a finales del 2003 estaba basada en Debian Gnu/Linux Woody 3.0 y Red Hat 9). La distribución Debian es un paradigma dentro del movimiento Open Source, por no pertenecer a ninguna empresa y estar confeccionada sólo por las aportaciones de los voluntarios distribuidos por todo el mundo. Debian, además, integra exclusivamente software libre (pueden añadirse otros aparte).

Fedora Core por otra parte, es la distribución que cuenta con el soporte comunitario amplio, y es base de las distribuciones de una de las empresas más solventes en el panorama comercial, Red Hat, y por eso sea quizás la que otorgue más soporte a nivel empresarial (mediante servicios de pago). En Debian y Fedora, el soporte depende de los voluntarios y del conocimiento compartido de los usuarios.

Siendo la administración de sistemas un campo tan amplio, este manual sólo pretende introducirnos en este apasionante (y cómo no, también a veces frus-

trante) mundo. Veremos algunas de las tareas típicas, y cómo tratar las problemáticas; pero la administración es un campo que se aprende día a día, con el trabajo diario. Y desde aquí advertimos de que este manual es un trabajo abierto, que con sus aciertos y los más que probables errores, se puede ver complementado con los comentarios de sus (sufridores) usuarios. De modo que son bienvenidos cualquier tipo de comentarios y sugerencias de mejora de los materiales.

Comentamos, por último, que el contenido del manual refleja el estado de las distribuciones y de las herramientas de administración en el momento de su confección (a finales del 2003 la primera edición y en primavera de 2007 la segunda edición).

Contenidos

Módulo didáctico 5

Administración local

Josep Jorba Esteve

1. Distribuciones: particularidades
2. Niveles de arranque y servicios
3. Observar el estado del sistema
4. Sistemas de ficheros
5. Usuarios y grupos
6. Servidores de impresión
7. Discos y gestión filesystems
8. Software: actualización
9. Trabajos no interactivos
10. Taller: prácticas combinadas de los diferentes apartados

Módulo didáctico 6

Administración de red

Remo Suppi Boldrito

1. Introducción a TCP/IP (TCP/IP suite)
2. Conceptos en TCP/IP
3. ¿Cómo se asigna una dirección Internet?
4. ¿Cómo se debe configurar la red?
5. Configuración del DHCP
6. IP *aliasing*
7. IP Masquerade
8. NAT con el *kernel* 2.2 o superiores
9. ¿Cómo configurar una conexión DialUP y PPP?
10. Configuración de la red mediante *hotplug*
11. Virtual *private network* (VPN)
12. Configuraciones avanzadas y herramientas

Módulo didáctico 7

Administración de servidores

Remo Suppi Boldrito

1. *Domain name system* (DNS)
2. NIS (YP)
3. Servicios de conexión remota: telnet y ssh
4. Servicios de transferencia de ficheros: FTP
5. Servicios de intercambio de información a nivel de usuario
6. Servicio de Proxy: Squid
7. OpenLdap (Ldap)
8. Servicios de archivos (NFS)

Módulo didáctico 8

Administración de datos

Remo Suppi Boldrito

1. PostgreSQL
2. Mysql
3. *Source code control system* (CVS y Subversion)
4. Subversion

Módulo didáctico 9

Administración de seguridad

Josep Jorba Esteve

1. Tipos y métodos de los ataques
2. Seguridad del sistema
3. Seguridad local
4. SELinux
5. Seguridad en red
6. Detección de intrusiones
7. Protección mediante filtrado (*wrappers* y *firewalls*)
8. Herramientas de seguridad
9. Análisis logs
10. Taller: análisis de la seguridad mediante herramientas

Módulo didáctico 10

Configuración, sintonización y optimización

Remo Suppi Boldrito

1. Aspectos básicos

Módulo didáctico 11

Clustering

Remo Suppi Boldrito

1. Introducción al HPC
2. OpenMosix
3. *Metacomputers, grid computing*

Introducción al sistema operativo GNU/Linux

Josep Jorba Esteve

P07/M2103/02280

UOC
www.uoc.edu

Índice

Introducción

Los sistemas GNU/Linux [Joh98] ya no son una novedad, cuentan con una amplia variedad de usuarios y de ámbitos de trabajo donde son utilizados.

Su origen se remonta al mes de agosto de 1991, cuando un estudiante finlandés llamado Linus Torvalds anunció en una lista de *news* que había creado su propio núcleo de sistema operativo y lo ofrecía a la comunidad de desarrolladores para que lo probara y sugiriera mejoras para hacerlo más utilizable. Éste sería el origen del núcleo (o *kernel*) del operativo que más tarde se llamaría Linux.

Por otra parte, la FSF (Free Software Foundation), mediante su proyecto GNU, producía software (desde 1984) que podía ser utilizado libremente. Debido a lo que Richard Stallman (miembro de la FSF) consideraba software libre, es decir, como aquél del que podíamos conseguir sus fuentes (código), estudiarlas y modificarlas, y redistribuirlo sin que nos obliguen a pagar por ello. En este modelo, el negocio no está en la ocultación del código, sino en el software complementario añadido, en la adecuación del software a los clientes y en los servicios añadidos, como el mantenimiento y la formación de usuarios (el soporte que les demos), ya sea en forma de material, libros y manuales, o en cursos de formación.

La combinación (o suma) del software GNU y del *kernel* Linux, es el que nos ha traído a los actuales sistemas GNU/Linux. Actualmente, los movimientos Open Source, desde diferentes organizaciones (como FSF) y empresas como las que generan las diferentes distribuciones Linux (Red Hat, Mandrake, SuSe...), pasando por grandes empresas como HP, IBM o Sun que proporcionan apoyo, han dado un empujón muy grande a los sistemas GNU/Linux hasta situarlos al nivel de poder competir, y superar, muchas de las soluciones propietarias cerradas existentes.

Los sistemas GNU/Linux no son ya una novedad. El software GNU se inició a mediados de los ochenta, el *kernel* Linux, a principios de los noventa. Y Linux se apoya en tecnología probada de UNIX, con más de 30 años de historia.

En esta unidad introductoria repasaremos algunas ideas generales de los movimientos Open Source y Free software, así como un poco de historia de Linux, y de sus orígenes compartidos con UNIX, de donde ha heredado más de 30 años de investigación en sistemas operativos.

1. Software Libre y Open Source

Bajo la idea de los movimientos (o filosofías) de Software Libre y Open Source [OSIc] [OSIb] (también llamado de código abierto o software abierto), se encuentran varias formas de software, no todas del mismo tipo, pero sí compartiendo muchas ideas comunes.

La denominación de un producto de software como 'de código abierto' conlleva como idea más importante la posibilidad de acceder a su código fuente, y la posibilidad de modificarlo y redistribuirlo de la manera que se considere conveniente, estando sujeto a una determinada licencia de código abierto, que nos da el marco legal.

Frente a un código de tipo propietario, en el cual un fabricante (empresa de software) encierra su código, ocultándolo y restringiéndose los derechos a sí misma, sin dar posibilidad de realizar ninguna adaptación ni cambios que no haya realizado previamente la empresa fabricante, el código abierto ofrece, entre otras consideraciones:

a) Acceso al código fuente, ya sea para estudiarlo (ideal para educación) o modificarlo, sea para corregir errores, adaptarlo o añadir más prestaciones.

b) Gratuidad: normalmente, el software, ya sea en forma binaria o en la forma de código fuente, puede obtenerse libremente o por una módica cantidad en concepto de gastos de empaquetamiento, distribución y valores añadidos.

c) Evitar monopolios de software propietario: no depender de una única opción o único fabricante de nuestro software. Esto es más importante cuando se trata de una gran organización, ya sea una empresa o estado, los cuales no pueden (o no deberían) ponerse en manos de una determinada única solución y pasar a depender exclusivamente de ella.

d) Un modelo de avance, no basado en la ocultación de información, sino en la compartición del conocimiento (semejante al de la comunidad científica), para lograr progresos de forma más rápida, con mejor calidad, ya que las elecciones tomadas están basadas en el consenso de la comunidad, y no en los caprichos de empresas desarrolladoras de software propietario.

Crear programas y distribuirlos junto al código fuente no es nuevo. Ya desde los inicios de la informática y en los inicios de la red Internet se había hecho así. Sin embargo, el concepto de *código abierto* como tal, la definición y la redacción de las condiciones que tenía que cumplir datan de mediados de 1997.

Eric Raymond y Bruce Perens fueron los que divulgaron la idea. Raymond [Ray98] era autor del ensayo titulado "La catedral y el Bazar", que hablaba sobre las técnicas de desarrollo de software utilizadas por la comunidad Linux, encabezada por Linus Torvalds, y la comunidad GNU de la Free Software Foundation (FSF), encabezada por Richard Stallman. Por su parte, Bruce Perens era en aquel momento el jefe del proyecto Debian, que trabajaba en la creación de una distribución de GNU/Linux integrada únicamente con software libre.

Nota

Ver versión española en:
http://es.tldp.org/Otros/catedral-bazar/cathedral-es-paper-00.html

Nota

Dos de las comunidades más importantes son la FSF, con su proyecto de software GNU, y la comunidad Open Source, cuyo máximo exponente de proyecto es Linux. GNU/Linux es el resultado de la unión de sus trabajos.

Una distinción importante entre estas comunidades son las definiciones de *código abierto* y *software libre*. [Deba] [PS02]

El Software Libre (*Free Software*) [FSF] es un movimiento que parte de las ideas de Richard Stallman, que considera que hay que garantizar que los programas estuviesen al alcance de todo el mundo de forma gratuita, se tuviese acceso libre a éstos y pudieran utilizarse al antojo de cada uno. Una distinción importante, que causó ciertas reticencias a las empresas, es el término *free*. En inglés, este término tiene el doble significado de 'gratuito' y 'libre'. La gente de la FSF buscaba las dos cosas, pero era difícil vender ambas cosas a las empresas; la pregunta típica era: ¿cómo se podía ganar dinero con esto? La respuesta vino de la comunidad Linux (con Linus Torvalds en cabeza), cuando consiguieron tener una cosa que todavía no había logrado la comunidad GNU y la FSF: tener un sistema operativo libre con código fuente disponible. En este momento es cuando a la comunidad se le ocurrió juntar las diversas actividades que había en la filosofía del Software Libre bajo la nueva denominación de *código abierto* (*open source*).

Open Source se registró como una marca de certificación, a la que podían adherirse los productos software que respetasen sus especificaciones. Esto no gustó a todo el mundo y suele haber cierta separación y controversias entre los dos grupos del Open Source y la FSF (con GNU), pero son más las cosas que los unen que las que los separan.

En cierta manera, para los partidarios del software libre (como la FSF), el código abierto (u *open source*) representa un paso en falso, ya que representa una cierta "venta" al mercado de sus ideales, y deja la puerta abierta a que se vaya haciendo propietario el software que era libre. Los partidarios de *open source* ven la oportunidad de promocionar el software que de otra manera estaría en una utilización minoritaria, mientras que con la divulgación y la puesta en común para todo el mundo, incluidas empresas que quieran participar en código abierto, entramos con suficiente fuerza para plantar cara al software propietario.

Sin embargo, la idea que persiguen ambas filosofías es la de aumentar la utilidad del software libre, ofreciendo así una alternativa a las soluciones únicas que las grandes empresas quieren imponer. Las diferencias son más que prácticas.

Una vez establecidas las ideas básicas de la comunidad del código abierto, llegamos al punto en que había que concretar de manera clara qué criterios tenía que cumplir un producto de software para considerarse de código abierto. Había que contar con una definición de código abierto [OSIb], que inicialmente escribió Bruce Perens en junio de 1997 como resultado de comentarios de los desarrolladores de la distribución Debian Linux, y que posteriormente fue reeditada (con modificaciones menores) por la organización OSI (Open Source Initiative). Esta organización está encargada de regular la definición y controlar las licencias de código abierto.

Nota

Ver la definición original de Open Source en:
http://www.opensource.org/docs/definition.php
Y la reedición en:
http://www.opensource.org

Nota

El código abierto está regulado por una definición pública que se utiliza como base de la redacción de sus licencias de software.

Un pequeño resumen (interpretación) de la definición: Un *open source software* [OSIb], o software de código fuente abierto, debe cumplir los requisitos siguientes:

1. Se puede copiar, regalar o vender a terceros el software, sin tener que pagar a nadie por ello. Se permite copiar el programa.

2. El programa debe incluir el código fuente y tiene que permitir la distribución tanto en forma compilada, como en fuente. O, en todo caso, hay que facilitar algún modo de obtener los códigos fuente (por ejemplo, descarga desde Internet). No está permitido ocultar el código o darlo en representaciones intermedias. Garantiza que se pueden hacer modificaciones.

3. La licencia del software tiene que permitir que se puedan realizar modificaciones y trabajos que se deriven, y que entonces se puedan distribuir bajo la misma licencia que la original. Permite reutilizar el código original.

4. Puede requerirse la integridad del código del autor, o sea, las modificaciones se pueden presentar en forma de parches al código original, o se puede pedir que tengan nombres o números distintos a los originales. Esto protege al autor de qué modificaciones puedan considerarse como suyas. Este punto depende de lo que diga la licencia del software.

5. La licencia no debe discriminar a ninguna persona o grupo. No se debe restringir el acceso al software. Un caso aparte son las restricciones por ley, como las de las exportaciones tecnológicas fuera de USA a terceros países. Si existen restricciones de este tipo, hay que mencionarlas.

6. No discriminar campos laborales. El software puede utilizarse en cualquier ambiente de trabajo, aunque no se haya pensado para él. Otra lectura es permitir fines comerciales, nadie puede impedir que el software se utilice con fines comerciales.

7. La licencia es aplicable a todo el mundo que reciba el programa.

8. Si el software forma parte de producto mayor, debe permanecer con la misma licencia. Esto controla que no se separen partes para formar software propietario (de forma no controlada). En el caso de software propietario, hay que informar que hay partes (y cuáles) de software de código abierto.

9. La licencia no debe restringir ningún software incorporado o distribuido conjuntamente, o sea, incorporarlo no debe suponer ninguna barrera para otro producto de software distribuido conjuntamente. Éste es un punto "polémico", ya que parece contradecirse con el anterior, básicamente dice que cualquiera puede coger software de código abierto y añadirlo al suyo sin que afecte a las condiciones de su licencia (por ejemplo propietaria), aunque sí que, según el punto anterior, tendría que informar de que existen partes de código abierto.

10. La licencia tiene que ser tecnológicamente neutra. No deben mencionarse medios de distribución únicos, o excluirse posibilidades. Por ejemplo, no puede limitarse (por licencia) que se haga la distribución en forma de CD, ftp o mediante web.

Esta definición de *código abierto* no es por sí misma una licencia de software, sino más bien una especificación de qué requisitos debería cumplir una licencia de software de código abierto.

La licencia que traiga el programa tiene que cumplir las especificaciones anteriores para que el programa se considere de código abierto. La organización OSI se encarga de comprobar que las licencias cumplen las especificaciones. En la página web de Open Source Licenses se puede encontrar la lista de las licencias [OSIa], siendo una de las más famosas y utilizadas, la GPL (GNU Public License).

Nota

Open Source Licences:
http://www.opensource.org/licenses/index.html

Bajo GPL, el software puede ser copiado y modificado, pero las modificaciones deben hacerse públicas bajo la misma licencia, y se impide que el código se mezcle con código propietario, para evitar así que el código propietario se haga con partes abiertas. Hay una licencia LGPL que es prácticamente igual, pero permite que software con esta licencia sea integrado en software propietario. Un ejemplo clásico es la biblioteca (*library*) C de Linux (con licencia LGPL); si ésta fuera GPL, sólo podría desarrollarse software libre, con la LGPL se permite usar para desarrollar software propietario.

Muchos proyectos de software libre, o con parte de código abierto y parte propietario, tienen su propia licencia: Apache (basada en BSD), Mozilla (MPL y NPL de Netscape), etc. Básicamente, a la hora de poner el software como *open source* podemos poner nuestra propia licencia que cumpla la definición anterior (de código abierto), o podemos escoger licenciar bajo una licencia ya establecida, o como en el caso de la GPL, nos obliga a que nuestra licencia también sea GPL.

Una vez vistos los conceptos de *código abierto* y sus licencias, nos queda por tratar hasta qué punto es rentable para una empresa trabajar o producir código abierto. Si no fuera atrayente para las empresas, perderíamos a la vez tanto un potencial cliente como uno de los principales productores de software.

El código abierto es también atrayente para las empresas, con un modelo de negocio donde se prima el valor añadido al producto.

En el código abierto existen diferentes rentabilidades atrayentes de cara a las empresas:

a) Para las empresas desarrolladoras de software, se crea un problema, ¿cómo es posible ganar dinero sin vender un producto? Hay mucho dinero gastado en desarrollar un programa y después es necesario obtener beneficios. Bien, la respuesta no es simple, no se puede conseguir con cualquier software, la rentabilidad se encuentra en el tipo de software que puede generar beneficios más allá de la simple venta. Normalmente, hay que hacer un estudio de si la aplicación se tornará rentable al desarrollarla como software abierto (la mayoría sí que lo hará), basándose en las premisas de que tendremos un descenso de gasto en desarrollo (la comunidad nos ayudará), reducción de mantenimiento o corrección de errores (la comunidad puede ofrecer esto muy rápido), y tener en cuenta el aumento de número de usuarios que nos proporcionará el código abierto, así como las necesidades que tendrán de nuestros servicios de apoyo o documentación. Si la balanza es positiva, entonces será viable prescindir de los ingresos generados por las ventas.

b) Aumentar la cuota de usuarios.

c) Obtener mayor flexibilidad de desarrollo, cuantas más personas intervienen, más gente habrá para detectar errores.

d) Los ingresos en su mayor parte vendrán por el lado del apoyo, formación de usuarios y mantenimiento.

e) En empresas que utilizan software, hay que considerar muchos parámetros a la hora de escoger el software para el desarrollo de las tareas, hay que tener en cuenta cosas como: rendimiento, fiabilidad, seguridad, escalabilidad y coste monetario. Y aunque parece que el código abierto ya supone de por sí una

elección por el coste económico, hay que decir que existe software abierto que puede competir con el propietario (o incluso superarlo) en cualquiera de los otros parámetros. Además, hay que vigilar mucho con las opciones o sistemas propietarios de un único fabricante, no podemos depender únicamente de ellos (podemos recordar casos, en otros ámbitos, como los vídeos beta de Sony frente a VHS, o en los PC la arquitectura MicroChannel de IBM). Tenemos que evitar el uso de monopolios con lo que éstos suponen: falta de competencia en los precios, servicios caros, mantenimiento caro, poca (o nula) variedad de opciones, etc.

f) Para los usuarios particulares ofrece gran variedad de software adaptado a tareas comunes, ya que mucho del software ha sido pensado e implementado por personas que querían hacer esas mismas tareas pero no encontraban el software adecuado. Normalmente, en el caso del usuario particular un parámetro muy importante es el coste del software, pero la paradoja es que en el usuario doméstico es donde se hace más uso de software propietario. Normalmente, los usuarios domésticos hacen uso de productos de software con copias ilegales, algunas estadísticas recientes indican índices del 60-70% de copias ilegales domésticas. El usuario siente que sólo por tener el ordenador doméstico PC ya tiene "derecho" a disponer de software para usarlo. En estos casos estamos bajo situaciones "ilegales" que, aunque no han sido perseguidas, pueden serlo en su día, o bien se intentan controlar por sistemas de licencias (o activaciones de productos). Además, esto tiene unos efectos perjudiciales indirectos sobre el software libre, debido a que si los usuarios hacen un uso amplio de software propietario, esto obliga a quien se quiera comunicar con ellos, ya sean bancos, empresas o administraciones públicas, a hacer uso del mismo software propietario, y ellos sí que abonan las licencias a los productos. Una de las "batallas" más importantes para el software libre es la posibilidad de captar a los usuarios domésticos.

Nota

Las copias ilegales domésticas son también denominadas en ocasiones *copias piratas*.

g) Por último, los estados, como caso particular, pueden obtener beneficios importantes del software de código abierto, ya que pueden disponer de software de calidad a precios "ridículos" comparados con el enorme gasto de licencias de software propietario (miles o decenas de miles). Además de que el software de código abierto permite integrar fácilmente a las aplicaciones cuestiones culturales (de cada país) como, por ejemplo, su lengua. Este último caso es bastante problemático, ya que en determinadas regiones, estados pequeños con lengua propia, los fabricantes de software propietario se niegan a adaptar sus aplicaciones, o instan a que se les pague por hacerlo.

2. UNIX. Un poco de historia

Como antecesor de nuestros sistemas GNU/Linux [Sta02], vamos a recordar un poco la historia de UNIX [Sal94] [Lev]. En origen, Linux se pensó como un clon de Minix (una implementación académica de UNIX para PC) y de algunas ideas desarrolladas en los UNIX propietarios; pero, a su vez, se desarrolló en código abierto, y con orientación a los PC domésticos. Veremos, en este apartado dedicado a UNIX y el siguiente dedicado a GNU/Linux, cómo esta evolución nos ha llevado hasta los sistemas GNU/Linux actuales que pueden competir con cualquier UNIX propietario, y que están disponibles para un amplio número de arquitecturas hardware, desde el simple PC hasta los supercomputadores.

Linux puede ser utilizado en un amplio rango de máquinas. En la lista TOP500, pueden encontrarse varios supercomputadores con GNU/Linux (ver lista en sitio web top500.org): por ejemplo, el MareNostrum, en el Barcelona Supercomputing Center, un cluster, diseñado por IBM, con 10240 CPUs PowerPC con sistema operativo GNU/Linux (adaptado para los requisitos de tales máquinas). En las estadísticas de la lista podemos observar que los supercomputadores con GNU/Linux ocupan en general un 75% de la lista.

Nota

Podemos ver la lista TOP500 de los supercomputadores más rápidos en:
http://www.top500.org

UNIX se inició hacia el año 1969 (tenemos ya casi 40 años de historia) en los laboratorios BTL (Bell Telephone Labs) de AT&T. Éstos se acababan de retirar de la participación de un proyecto llamado MULTICS, cuyo objetivo era crear un sistema operativo con el cual un gran ordenador pudiera dar cabida a un millar de usuarios simultáneos. En este proyecto participaban los BTL, General Electric, y el MIT. Pero falló, en parte, por ser demasiado ambicioso para su época.

Mientras se desarrollaba este proyecto, dos ingenieros de los BTL que participaban en MULTICS: Ken Thompson y Dennis Ritchie, encontraron un ordenador que no estaba utilizando nadie, un DEC PDP7, que sólo tenía un ensamblador y un programa cargador. Thompson y Ritchie desarrollaron como pruebas (y a menudo en su tiempo libre) partes de UNIX, un programa ensamblador (del código máquina) y el núcleo rudimentario del sistema operativo.

Ese mismo año, 1969, Thompson tuvo la idea de escribir un sistema de ficheros para el núcleo creado, de manera que se pudiesen almacenar ficheros de forma ordenada en un sistema de directorios jerárquicos. Después de unas cuantas discusiones teóricas (que se alargaron unos dos meses) se implementó el sistema en un par de días. A medida que se avanzaba en el diseño del sistema, en el cual se incorporaron algunos ingenieros más de los BTL, la máquina original se les quedó pequeña, y pensaron en pedir una nueva (en aquellos

días costaban cerca de 100.000 dólares, era una buena inversión). Tuvieron que inventarse una excusa (ya que el sistema UNIX era un desarrollo en tiempo libre) y dijeron que la querían para crear un nuevo procesador de texto (aplicación que daba dinero en aquellos tiempos), y se les aprobó la compra de una PDP11.

UNIX se remonta al año 1969, cuenta con más de 30 años de tecnologías desarrolladas y utilizadas en todo tipo de sistemas.

Cuando les llegó la máquina, sólo les llegó la CPU y la memoria, pero no el disco ni el sistema operativo. Thompson, sin poder esperarse, diseñó un disco RAM en memoria y utilizó la mitad de la memoria como disco, y la otra para el sistema operativo que estaba diseñando. Una vez llegó el disco, se siguió trabajando tanto en UNIX como en el procesador de textos prometido (la excusa). El procesador de textos fue un éxito (se trataba de Troff, un lenguaje de edición, que posteriormente fue utilizado para crear las páginas man de UNIX), y los BTL comenzaron a utilizar el rudimentario UNIX con el nuevo procesador de texto, convirtiéndose así los BTL en el primer usuario de UNIX.

En aquellos momentos comenzaron a presentarse varios principios filosóficos de UNIX [Ray02a]:

- Escribir programas para hacer una cosa y hacerla bien.
- Escribir programas para que trabajaran juntos.
- Escribir programas para que manejaran flujos de texto.

Otra idea muy importante fue que UNIX fue uno de los primeros sistemas pensados para ser independiente de la arquitectura hardware, y que ha permitido portarlo con éxito a un gran número de arquitecturas hardware diferentes.

La necesidad de documentar lo que se estaba haciendo, ya que había usuarios externos, dio lugar en noviembre de 1971 al *UNIX Programmer's Manual*, que firmaron Thompson y Richie. En la segunda edición (junio 1972), denominada V2 (se hacía corresponder la edición de los manuales con el número de versión UNIX), se decía que el número de instalaciones de UNIX ya llegaba a las 10. Y el número siguió creciendo hasta unas 50 en la V5.

Entonces se decidió (finales de 1973) presentar los resultados en un congreso de sistemas operativos. Y como resultado, varios centros informáticos y universidades pidieron copias de UNIX. AT&T no daba apoyo ni mantenimiento de UNIX, lo que hizo que los usuarios necesitaran unirse y compartir sus conocimientos para formar comunidades de usuarios de UNIX. AT&T decidió ceder UNIX a las universidades, pero tampoco les daba apoyo, ni corrección de errores. Los usuarios comenzaron a compartir sus ideas, información de programas, *bugs*, etc. Se creó una asociación denominada USENIX como agrupación de usuarios de UNIX. Su primera reunión (mayo de 1974) tuvo una docena de asistentes.

Nota

Ver: http://www.usenix.org

Una de las universidades que había obtenido una licencia de UNIX fue la Universidad de California en Berkeley, donde había estudiado Ken Thompson. En 1975, Thompson volvió como profesor a Berkeley, y trajo consigo la última versión de UNIX. Dos estudiantes graduados recién incorporados, Chuck Haley y Bill Joy (hoy en día uno de los vicepresidentes de SUN Microsystems) comenzaron a trabajar en una implementación de UNIX.

Una de las primeras cosas que les decepcionó eran los editores; Joy perfeccionó un editor llamado EX, hasta transformarlo en el VI, un editor visual a pantalla completa. Y los dos escribieron un compilador de lenguaje Pascal, que añadieron a UNIX. Hubo cierta demanda de esta implementación de UNIX, y Joy lo comenzó a producir como el BSD, *Berkeley Software Distribution* (o UNIX BSD).

BSD (en 1978) tenía una licencia particular sobre su precio: decía que estaba acorde con el coste de los medios y la distribución que se tenía en ese momento. Así, los nuevos usuarios acababan haciendo algunos cambios o incorporando cosas, vendiendo sus copias "rehechas" y, al cabo de un tiempo, los cambios se incorporaban en la siguiente versión de BSD.

Joy también realizó en su trabajo del editor VI algunas aportaciones más, como el tratamiento de los terminales de texto, de manera que el editor fuera independiente del terminal en que se utilizase; creó el sistema TERMCAP como interfaz genérica de terminales con controladores para cada terminal concreto, de manera que en la realización de los programas ya nos podíamos olvidar de los terminales utilizando la interfaz.

Un siguiente paso fue adaptarlo a diferentes arquitecturas. Hasta el año 1977 sólo se podía ejecutar en máquinas PDP; en ese año se comenzaron a hacer adaptaciones para máquinas del momento como las Interdata e IBM. La versión 7 (V7 en junio 1979) de UNIX fue la primera portable. Esta versión trajo muchos avances, ya que contenía: *awk*, *lint*, *make*, *uucp*; el manual ya tenía 400 páginas (más dos apéndices de 400 cada uno). Se incluía también el compilador de C diseñado en los BTL por Kernighan y Ritchie, que se había creado para reescribir la mayor parte de UNIX, inicialmente en ensamblador y luego pasado a C con las partes de ensamblador que fuesen sólo dependientes de la arquitectura. Se incluyeron también una *shell* mejorada (*shell* de Bourne) y comandos como: *find*, *cpio* y *expr*.

La industria UNIX comenzó también a crecer, empezaron a aparecer versiones (implementaciones) de UNIX por parte de compañías como: Xenix, colaboración entre Microsoft (en los orígenes también trabajó con versiones de UNIX) y SCO para máquinas Intel 8086 (el primer PC de IBM); nuevas versiones BSD de Berkeley...

Pero apareció un nuevo problema, cuando AT&T se dio cuenta de que UNIX era un producto comercial valioso, en la licencia de la V7 se prohibió el estu-

dio en centros académicos, para proteger el secreto comercial. Muchas universidades utilizaban hasta el momento el código fuente de UNIX para docencia de sistemas operativos, y dejaron de usarlo para dar sólo teoría.

Sin embargo, cada uno solucionó el problema a su modo. En Amsterdam, Andrew Tanenbaum (autor de prestigio de libros de teoría de sistema operativos) decidió escribir desde el principio un nuevo sistema operativo compatible con UNIX sin utilizar una sola línea de código de AT&T; llamó a este nuevo operativo Minix. Éste sería el que posteriormente le serviría en 1991 a un estudiante finlandés para crear su propia versión de UNIX, que llamó Linux.

Bill Joy, que continuaba en Berkeley desarrollando BSD (ya estaba en la versión 4.1), decidió marcharse a una nueva empresa llamada SUN Microsystems, en la cual acabó los trabajos del 4.2BSD, que posteriormente modificaría para crear el UNIX de SUN, el SunOS (hacia 1983). Cada empresa comenzó a desarrollar sus versiones: IBM con AIX, DEC con Ultrix, HP con HPUX, Microsoft/SCO con Xenix, etc. UNIX comenzó (desde el año 1980) su andadura comercial, AT&T sacó una ultima versión llamada UNIX SystemV (SV), de la cual derivan, junto con los 4.xBSD, los UNIX actuales, ya sea de la rama BSD o de la SystemV. La SV tuvo varias revisiones, por ejemplo, la SV Release 4 fue una de las más importantes. La consecuencia de estas últimas versiones es que más o menos todos los UNIX existentes se adaptaron uno al otro; en la práctica son versiones del SystemV R4 de AT&T o del BSD de Berkeley, adaptadas por cada fabricante. Algunos fabricantes lo especifican y dicen que su UNIX es de tipo BSD o SV, pero la realidad es que todos tienen un poco de las dos, ya que posteriormente se hicieron varios estándares de UNIX para intentar uniformizarlos; entre ellos encontramos los IEEE POSIX, UNIX97, FHS, etc.

Con el tiempo, UNIX se dividió en varias ramas de sistema, siendo las dos principales, la que derivaba del AT&T UNIX o System V, y la de la universidad de California, el BSD. La mayoría de UNIX actuales derivan de uno u otro, o son una mezcla de los dos.

Pero AT&T en aquellos momentos (SVR4) pasó por un proceso judicial por monopolio telefónico (era la principal, si no la única, compañía telefónica en Estados Unidos), que hizo que se dividiera en múltiples empresas más pequeñas, y los derechos de UNIX originales comenzaron un baile de propietarios importante: en 1990 los tenían a medias el Open Software Foundation (OSF) y UNIX International (UI), después, UNIX Systems Laboratories (USL), que denunció a la Universidad de Berkeley por sus copias del BSD, pero perdió, ya que la licencia original no imponía derechos de propiedad al código de UNIX. Más tarde, los derechos UNIX se vendieron a la empresa Novell, ésta cedió parte a SCO, y hoy en día no está muy claro quién los tiene: por diferentes frentes los reclaman Novell, la OSF y SCO. Un ejemplo reciente de esta problemática puede ser el caso de SCO, que puso una demanda legal a IBM porque ésta, según SCO, había cedido parte del código UNIX a versiones del *kernel* Linux, que

supuestamente incluyen algún código UNIX original. El resultado a día de hoy es que el asunto continúa en los tribunales, con SCO convertida en un "paria" de la industria informática que amenaza a los usuarios Linux, IBM, y otros UNIX propietarios, con la afirmación de que tienen los derechos UNIX originales, y que los demás tienen que pagar por ellos. Habrá que ver cómo evoluciona este caso, y el tema de los derechos UNIX con él.

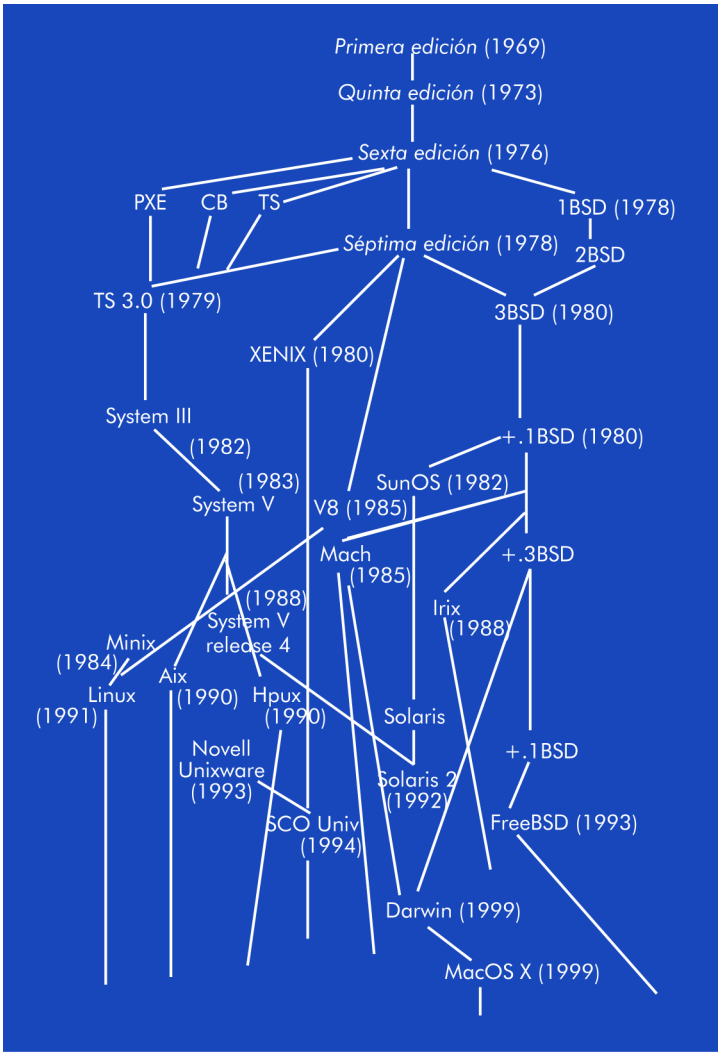

Figura 1. Resumen histórico de varias versiones UNIX

El panorama actual de UNIX ha cambiado mucho desde la aparición de Linux (1991), que a partir de los años 1995-99 comenzó a convertirse en una alternativa seria a los UNIX propietarios, por la gran cantidad de plataformas hardware que soporta y el amplio apoyo de la comunidad internacional y empresas en el avance. Hay diferentes versiones UNIX propietarias que siguen sobreviviendo en el mercado, tanto por su adaptación a entornos industriales o por ser el mejor operativo existente en el mercado, como porque hay necesidades que sólo pueden cubrirse con UNIX y el hardware adecuado. Además, algunos de los UNIX propietarios todavía son mejores que GNU/Linux en cuanto a fiabilidad y rendimiento aunque cada vez acortando distancias, ya que las mismas empresas que tienen sus UNIX propietarios se interesan cada vez más en GNU/Linux, y aportan parte de sus desarrollos para incorporarlos a Linux. Es

de esperar una muerte más o menos lenta de las versiones propietarias de UNIX hacia distribuciones basadas en Linux de los fabricantes adaptadas a sus equipos.

Un panorama general de estas empresas:

Nota

Muchas de las empresas que disponen de UNIX propietarios participan en GNU/Linux y ofrecen algunos de sus desarrollos a la comunidad.

- SUN: dispone de su implementación de UNIX llamada Solaris (evolución del SunOS). Comenzó como un sistema BSD, pero ahora es mayoritariamente SystemV y partes de BSD; es muy utilizado en las máquinas Sun con arquitectura Sparc, y en máquinas multiprocesador (hasta unos 64 procesadores). Promocionan GNU/Linux como entorno de desarrollo para Java, y disponen de una distribución de GNU/Linux denominada Java Desktop System, que ha tenido una amplia aceptación en algunos países. Además, ha comenzado a usar Gnome como escritorio, y ofrece apoyo financiero a varios proyectos como Mozilla, Gnome y OpenOffice. También cabe destacar la iniciativa tomada, con su última versión de su UNIX Solaris, para liberar su código casi totalmente, en la versión Solaris 10. Creando una comunidad para las arquitecturas Intel y Sparc, denominada OpenSolaris, que ha permitido la creación de distribuciones libres de Solaris. Aparte tenemos que señalar iniciativas recientes (2006) para liberar la plataforma Java bajo licencias GPL.

- IBM: tiene su versión de UNIX propietaria denominada AIX, que sobrevive en algunos segmentos de estaciones de trabajo y servidores de la firma. Por otra parte, presta apoyo firme a la comunidad Open Source, promoviendo entornos de desarrollo libres (eclipse.org) y tecnologías Java para Linux, incorpora Linux a sus grandes máquinas y diseña campañas publicitarias (marketing) para promocionar Linux. Además, tiene una repercusión importante en la comunidad por el ambiente judicial de su caso defendiéndose de la firma SCO, que la acusa de violación de propiedad intelectual UNIX, por haber integrado supuestamente componentes en GNU/Linux.

- HP: tiene su UNIX HPUX, pero da amplio soporte a Linux, tanto en forma de código en Open Source, como instalando Linux en sus máquinas. Se dice que es la compañía que ha ganado más dinero con Linux.

- SGI: Silicon Graphics tiene un UNIX llamado IRIX para sus máquinas gráficas, pero últimamente tiende a vender máquinas con Windows, y puede que algunas con Linux. Ha pasado por algunas deficiencias empresariales, por las que estuvo a punto de quebrar. A la comunidad Linux le ofrece soporte de OpenGL (tecnología de gráficos 3D) y de diferentes sistemas de ficheros y control de dispositivos periféricos.

- Apple: se incorporó recientemente (a partir de mediados de los noventa) al mundo UNIX, cuando decidió sustituir su operativo por una variante UNIX. El núcleo llamado Darwin proviene de una versión 4.4BSD; este núcleo Open Source será el que, sumado a unas interfaces gráficas muy potentes, dé a Apple

su sistema operativo MacOS X. Considerado hoy en día como uno de los mejores UNIX y, como mínimo, uno de los más "bellos" en aspecto gráfico. También emplea gran cantidad de software GNU como utilidades de sistema.

- Distribuidores Linux: tanto comerciales como organizaciones, mencionaremos a empresas como Red Hat, SuSe, Mandriva (previamente conocida como Mandrake), y organizaciones no comerciales como Debian, etc. Entre éstas (las distribuciones con mayor despliegue) y las más pequeñas, se llevan el mayor desarrollo de GNU/Linux, y tienen el apoyo de la comunidad Linux y de la FSF con el software GNU, además de recibir contribuciones de las citadas empresas.

- BSD: aunque no sea una empresa como tal, mencionaremos cómo desde Berkeley y otros intermediarios se continúa el desarrollo de las versiones BSD, así como otros proyectos libres clones de BSD como los operativos FreeBSD, netBSD, OpenBSD (el UNIX considerado más seguro), TrustedBSD, etc., que también, más tarde o más temprano, suponen mejoras o incorporaciones de software a Linux. Además, una aportación importante es el *kernel* Darwin proveniente de 4.4BSD, y que desarrolló Apple como núcleo Open Source de su sistema operativo MacOS X.

- Microsoft: aparte de entorpecer el desarrollo de UNIX y GNU/Linux, poniendo trabas con incompatibilidades en diferentes tecnologías, no tiene participación directa en el mundo UNIX/Linux. Aunque en sus orígenes desarrolló Xenix (1980) para PC, a partir de una licencia AT&T de UNIX, que si bien no vendió directamente, sí que lo hizo por medio de intermediarios, como SCO que se hizo con el control en 1987, renombrado como SCO UNIX (1989). Como nota curiosa, posteriormente, compró parte de derechos de la licencia UNIX a SCO (ésta los había obtenido a su vez a través de Novell), no están claros los motivos de Microsoft a la hora de realizar esta adquisición, aunque algunos sugieren que existe alguna relación con el hecho de proporcionar apoyo a SCO en su juicio contra IBM. Además, recientemente (2006), Microsoft llegó a acuerdos con Novell (actual proveedora de la distribución SuSe y la comunidad OpenSuse), en una serie de decisiones bilaterales para promocionar empresarialmente ambas plataformas. Pero parte de la comunidad GNU/Linux se mantiene escéptica, por las posibles implicaciones sobre la propiedad intelectual de Linux y los temas que podrían incluir problemas judiciales por uso de patentes.

Otra anécdota histórica curiosa es que, junto a una empresa llamada UniSys, se dedicaron a realizar marketing de cómo convertir sistemas UNIX a sistemas Windows; y aunque el objetivo podía ser más o menos loable, lo curioso era que el servidor original de la web empresarial estaba en una máquina FreeBSD con Apache. En ocasiones, también paga a algunas empresas "independien-

Nota

Carta abierta de Novell a la comunidad GNU/Linux
http://www.novell.com/linux/microsoft/community_open_letter.html

tes" (algunos opinan que bastante poco) para que lleven a cabo estudios de rendimiento comparativos entre UNIX/Linux y Windows.

Como resumen general, algunos comentarios que suelen aparecer en la bibliografía UNIX apuntan a que: UNIX es técnicamente un sistema sencillo y coherente diseñado con buenas ideas que se supieron llevar a la práctica, pero que no hay que olvidar que algunas de estas ideas se consiguieron gracias al apoyo entusiasta que brindó una gran comunidad de usuarios y desarrolladores que colaboraron entre sí, compartiendo una tecnología y gobernando su evolución.

Y como la historia se suele repetir, en este momento la evolución y el entusiasmo continúan con los sistemas GNU/Linux.

3. Sistemas GNU/Linux

Hace unos veinte años los usuarios de los primeros ordenadores personales no disponían de muchos sistemas operativos donde elegir. El mercado de los ordenadores personales lo dominaba un DOS de Microsoft. Otra posibilidad era los MAC de Apple, pero a unos precios desorbitados (en comparación) con el resto. La otra opción importante, aunque reservada a grandes (y caras) máquinas, era UNIX.

Una primera opción que apareció fue MINIX (1984), creado desde cero por Andrew Tanenbaum, con el objetivo de usarlo para la educación, para enseñar diseño e implementación de sistemas operativos [Tan87] [Tan06].

MINIX fue pensado para ejecutarse sobre una plataforma Intel 8086, muy popular en la época porque era la base de los primeros IBM PC. La principal ventaja de este operativo radicaba en su código fuente, accesible a cualquiera (doce mil líneas de código entre ensamblador y C), ya que estaba incluido en el libro docente de sistemas operativos de Tanenbaum [Tan87]. Pero MINIX era más una herramienta de enseñanza que un sistema eficaz pensado para el rendimiento o para actividades profesionales.

En los noventa, la FSF (Free Software Foundation) y su proyecto GNU, motivó a muchos programadores para promover el software de calidad y de distribución libre. Y aparte de software de utilidades, se trabajaba en un núcleo (*kernel*) de operativo denominado HURD, que llevaría varios años de desarrollo.

Mientras, en octubre de 1991, un estudiante finlandés llamado Linus Torvalds presentaría la versión 0.0.1 de su *kernel* de sistema operativo, que denominó Linux, orientado a máquinas Intel con 386, y lo ofreció bajo licencia GPL a foros de programadores y a la comunidad de Internet para que lo probaran y, si les gustaba, le ayudaran a su desarrollo. El entusiasmo fue tal, que en poco tiempo había un gran número de programadores trabajando en el núcleo o en aplicaciones para él.

Algunas de las características que diferenciaron a Linux de los sistemas de su tiempo y que siguen siendo aplicables, y otras heredadas de UNIX podrían ser:

a) Sistema operativo de código abierto, cualquiera puede disponer de sus fuentes, modificarlas y crear nuevas versiones que poder compartir bajo la licencia GPL (que, de hecho, lo convierte en un software libre).

b) Portabilidad: tal como el UNIX original, Linux está pensado para depender muy poco de una arquitectura concreta de máquina; consecuentemente,

Linux es, en su mayor parte, independiente de la máquina de destino y puede portarse a prácticamente cualquier arquitectura que disponga de un compilador C como el GNU *gcc*. Sólo restan algunas pequeñas partes de código ensamblador y de algunos dispositivos dependientes de la máquina, que tienen que ser rescritas en cada puerto a una nueva arquitectura. Gracias a esto, GNU/Linux es uno de los sistemas operativos que corre en mayor número de arquitecturas: Intel x86 y IA64, AMD x86 y x8664, Sparc de Sun, MIPS de Silicon, PowerPC (Apple), IBM S390, Alpha de Compaq, m68k Motorola, Vax, ARM, HPPArisc...

c) *Kernel* de tipo monolítico: el diseño del *kernel* está unido en una sola pieza, pero es conceptualmente modular en las diferentes tareas. Otra escuela de diseño de operativos propone los *microkernel* (un ejemplo es Mach), donde los servicios se implementan como procesos aparte, comunicados por un (micro) *kernel* más básico. Linux se decidió como monolítico, porque es difícil extraer buen rendimiento de los *microkernels* (es un trabajo bastante duro y complejo). Por otra parte, el problema de los monolíticos es el crecimiento, cuando se vuelven muy grandes se vuelven intratables en el desarrollo, esto se intentó solucionar con los módulos de carga dinámica.

Nota

Proyecto original Mach:
http://www.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html

d) Módulos dinámicamente cargables: permiten poner partes del sistema operativo, como *filesystems*, o *controladores de dispositivos*, como porciones externas que se cargan (o enlazan) con el *kernel* en tiempo de ejecución bajo demanda. Esto permite simplificar el *kernel* y ofrecer estas funcionalidades como elementos que se pueden programar por separado. Con este uso de módulos, se podría considerar a Linux como un *kernel* mixto, ya que es monolítico, pero ofrece una serie de módulos que complementan el *kernel* (aproximación parecida al *microkernel*).

e) Desarrollo del sistema por una comunidad vinculada por Internet: los sistemas operativos nunca habían tenido un desarrollo tan amplio y disperso, no suelen salir de la compañía que los elabora (en el caso propietario) o de un pequeño conjunto de instituciones académicas y laboratorios que colaboran para crear uno. El fenómeno de la comunidad Linux permite que cada uno colabore en la medida en que el tiempo y sus propios conocimientos se lo permitan. El resultado es: de cientos a miles de desarrolladores para Linux. Además, por su naturaleza de sistema de código fuente abierto, Linux es un laboratorio ideal para probar ideas de sistemas operativos al mínimo coste; se puede implementar, probar, tomar medidas y, si funciona, añadir la idea al *kernel*.

Los proyectos se sucedieron y –en el inicio de Linus con el *kernel*– a la gente de la FSF, con el software de utilidad GNU y, sobre todo, con su compilador de C (*GCC*), se les unieron otros proyectos importantes como las XFree (una versión PC de las X Window), los proyectos de escritorio como KDE y Gnome. Y el desarrollo de Internet con proyectos como el servidor web Apache, el navegador Mozilla, o las bases de datos MySQL y PostgreSQL, acabaron por dar al

kernel inicial Linux el recubrimiento de aplicaciones suficiente para construir los sistemas GNU/Linux y competir en igualdad de condiciones con los sistemas propietarios. Y convertir a los sistemas GNU/Linux en el paradigma del software de fuente abierta (Open Source).

Los sistemas GNU/Linux se han convertido en la punta de lanza de la comunidad Open Source, por la cantidad de proyectos que se han podido aglutinar y llevar a buen término.

El nacimiento de nuevas empresas, que crearon distribuciones GNU/Linux (empaquetamientos de *kernel* + aplicaciones) y le dieron apoyo, como Red Hat, Mandrake, SuSe, contribuyó a introducir GNU/Linux en las empresas reacias, y a comenzar el imparable crecimiento que vivimos actualmente.

Comentaremos también la discusión sobre la denominación de los sistemas como GNU/Linux. El término *Linux* para identificar el sistema operativo con que se trabaja es de común uso (para simplificar el nombre), aunque en opinión de algunos desmerece el trabajo de la FSF con el proyecto GNU, el cual ha proporcionado las principales herramientas del sistema. Aun así, el término *Linux*, para referirse al sistema operativo completo es ampliamente usado comercialmente.

Nota

GNU y Linux, por Richard Stallman:

http://www.gnu.org/gnu/linux-andgnu.html.

En general, para seguir una denominación más ajustada a la participación de la comunidad, se utiliza el termino *Linux*, cuando nos estamos refiriendo sólo al núcleo (*kernel*) del sistema operativo. Esto crea cierta confusión, ya que hay gente que habla de "sistemas" o del "sistema operativo Linux" para abreviar. Cuando se trabaja con un sistema operativo GNU/Linux, se está trabajando sobre una serie de software de utilidades, en gran parte fruto del proyecto GNU, sobre el núcleo Linux. Por lo tanto, el sistema es básicamente GNU con un núcleo Linux.

El proyecto GNU de la FSF tenía por objetivo crear un sistema operativo de software libre al estilo UNIX denominado GNU [Sta02].

Linus Torvalds consiguió en 1991 juntar su *kernel* Linux con las utilidades GNU cuando la FSF todavía no disponía de *kernel*. El *kernel* de GNU se denomina HURD, y hoy en día se trabaja bastante en él, y ya existen algunas versiones beta de distribuciones de GNU/HURD (ver más en el apartado dedicado a la administración del *kernel*).

Se calcula que en una distribución GNU/Linux hay un 28% de código GNU y un 3% que corresponde al código del *kernel* Linux; el porcentaje restante corresponde a código de terceros, ya sea de aplicaciones o de utilidades.

Para destacar la contribución de GNU [FSF], podemos ver algunas de sus aportaciones incluidas en los sistemas GNU/Linux:

- El compilador de C y C++ (*GCC*)
- El *shell bash*
- El editor Emacs (GNU Emacs)
- El intérprete *postscript* (*ghostscript*)
- La biblioteca C estándar (*GNU C library*, o también *glibc*)
- El depurador (*GNU gdb*)
- *Makefile* (*GNU make*)
- El ensamblador (*GNU assembler* o *gas*)
- El *linker* (*GNU linker* o *gld*)

Los sistemas GNU/Linux no son los únicos en utilizar software GNU; por ejemplo, los sistemas BSD incorporan también utilidades GNU. Y algunos operativos propietarios como MacOS X (de Apple) también usan software GNU. El proyecto GNU ha producido software de alta calidad, que se ha ido incorporando a la mayor parte de las distribuciones de sistemas basadas en UNIX, tanto libres como propietarias.

Es justo para todo el mundo reconocer el trabajo de cada uno denominando GNU/Linux a los sistemas que trataremos.

4. El perfil del administrador de sistemas

Las grandes empresas y organizaciones dependen cada vez más de sus recursos de computación y de cómo éstos son administrados para adecuarlos a las tareas. El gran incremento de las redes distribuidas, con sus equipos servidores y clientes, ha creado una gran demanda de un nuevo perfil laboral: el llamado *administrador de sistemas*.

El administrador de sistemas tiene una amplia variedad de tareas importantes. Los mejores administradores de sistema suelen ser bastante generalistas, tanto teóricamente como prácticamente. Pueden enfrentarse a tareas como: realizar cableados de instalaciones o reparar cables; instalar sistemas operativos o software de aplicaciones; corregir problemas y errores en los sistemas, tanto hardware como software; formar a los usuarios; ofrecer trucos o técnicas para mejorar la productividad en áreas que pueden ir desde aplicaciones de procesamiento de textos hasta áreas complejas de sistemas CAD o simuladores; evaluar económicamente compras de equipamiento de hardware y software; automatizar un gran número de tareas comunes, e incrementar el rendimiento general del trabajo en su organización.

Puede considerarse al administrador como un perfil de empleado que ayuda a los demás empleados de la organización a aprovechar mejor y más óptimamente los recursos disponibles, de forma que mejore toda la organización.

La relación con los usuarios finales de la organización puede establecerse de diferentes maneras: o bien mediante la formación de usuarios o bien por ayuda directa en el caso de presentarse problemas. El administrador es la persona encargada de que las tecnologías utilizadas por los usuarios funcionen adecuadamente, o sea, que los sistemas cumplan las perspectivas de los usuarios, así como las tareas que éstos quieran realizar.

Hace años, y aún actualmente, en muchas empresas u organizaciones no hay una perspectiva clara del papel del administrador. En los inicios de la informática (años ochenta y noventa) en la empresa, el administrador era visto en un principio como la persona "entendida" en ordenadores (el "gurú") que se encargaba de poner máquinas y que vigilaba o las reparaba en caso de problemas. Normalmente, era una especie de informático polivalente que tenía que solucionar los problemas que fueran apareciendo. Su perfil de currículum no era claro, ya que no necesitaba tener amplios conocimientos, sino sólo tener conocimientos básicos de una decena (como mucho) de aplicaciones (el procesador de texto, la hoja de cálculo, la base de datos, etc.), y algunos conocimientos básicos de hardware eran suficientes para las tareas diarias. Así, cualquier simple "entendido" en el tema podía dedicarse a este trabajo, de manera

que no solían ser informáticos tradicionales, y muchas veces incluso se llegaba a una transmisión oral de los conocimientos entre algún "administrador" más antiguo en la empresa y el nuevo aprendiz.

Con lo anterior, nos encontrábamos de alguna manera en la prehistoria de la administración de sistemas (aunque hay personas que siguen pensando que básicamente se trata del mismo trabajo). Hoy en día, en la época de Internet y de los servicios distribuidos, un administrador de sistemas es un profesional (con dedicación propia y exclusiva) que proporciona servicios en la "arena" del software y hardware de sistemas. El administrador tiene que llevar a cabo varias tareas que tendrán como destino múltiples sistemas informáticos, la mayoría heterogéneos, con objeto de hacerlos operativos para una serie de tareas.

Actualmente, los administradores necesitan tener unos conocimientos generales (teóricos y prácticos) de áreas muy diversas, desde tecnologías de redes, sistemas operativos, aplicaciones de ámbitos diversos, programación básica en una amplia variedad de lenguajes de programación, conocimientos amplios de hardware –tanto del ordenador como de los periféricos usados– tecnologías Internet, diseño de páginas web, bases de datos, etc. Y normalmente también es buscado con el perfil de conocimientos básicos sobre el área de trabajo de la empresa, ya sea química, física, matemáticas, etc. No es de extrañar, entonces, que en una empresa de tamaño medio a grande se haya pasado del "chapuzas" de turno a un pequeño grupo de profesionales con amplios conocimientos, la mayoría con nivel académico universitario, con diferentes tareas asignadas dentro de la organización.

El administrador debe dominar un rango amplio de tecnologías para poder adaptarse a una multitud de tareas variadas, que pueden surgir dentro de la organización.

Debido a la gran cantidad de conocimientos, no es extraño que aparezcan a su vez diferentes subperfiles de la tarea del administrador. En una gran organización puede ser habitual encontrar a los administradores de sistemas operativos (UNIX, Mac, o Windows), que suelen ser diferentes: administrador de bases de datos, administrador de copias de seguridad, administradores de seguridad informática, administradores encargados de atención a los usuarios, etc.

En una organización más pequeña, varias o todas las tareas pueden estar asignadas a uno o pocos administradores. Los administradores de sistemas UNIX (o de GNU/Linux) serían una parte de estos administradores (cuando no el administrador que tendrá que hacer todas las tareas). Normalmente, su plataforma de trabajo es UNIX (o GNU/Linux en nuestro caso), y requiere de bastantes elementos específicos que hacen este trabajo único. UNIX (y variantes) es un sistema operativo abierto y muy potente, y, como cualquier sistema software, requiere de cierto

nivel de adecuación, configuración y mantenimiento en las tareas para las que vaya a ser usado. Configurar y mantener un sistema operativo es una tarea seria, y en el caso de UNIX puede llegar a ser bastante frustrante.

Algunas áreas importantes por tratar son:

a) Que el sistema sea muy potente también indica que habrá bastantes posibilidades de adaptarlo (configurarlo) a las tareas que queremos hacer. Habrá que evaluar las posibilidades que se nos ofrecen y cuán adecuadas son para nuestro objetivo final.

b) Un sistema abierto y ejemplo claro de ello es nuestro GNU/Linux, que nos ofrecerá actualizaciones permanentes, ya sea en la corrección de errores del sistema, como en la incorporación de nuevas prestaciones. Y, evidentemente, todo esto tiene unos impactos directos importantes en costes de mantenimiento de las tareas de administración.

c) Los sistemas se pueden utilizar para tareas de coste crítico, o en puntos críticos de la organización, donde no se pueden permitir fallos importantes, o que ralenticen o paren la marcha de la organización.

d) Las redes son actualmente un punto muy importante (si no el que más), pero también es un área de problemas potenciales muy crítica, tanto por su propia naturaleza distribuida como por la complejidad del sistema para encontrar, depurar y solucionar los problemas que se puedan presentar.

e) En el caso particular de los sistemas UNIX, y en nuestros GNU/Linux, la abundancia, tanto de versiones como de distribuciones diferentes del sistema, incorpora problemas adicionales a la administración, ya que es necesario conocer las problemáticas y diferencias de cada versión y distribución.

En particular, las tareas de administración del sistema y de la red suelen presentar particularidades diferentes, y a veces se tratan por separado (o por administradores diferentes). Aunque también pueden verse como dos caras del mismo trabajo, con el sistema propiamente dicho (máquina y software) por un lado, y el ambiente donde el sistema (el entorno de red) convive, por el otro.

Normalmente, por *administración de la red* se entiende la gestión del sistema como parte de la red, y hace referencia a los servicios o dispositivos cercanos necesarios para que la máquina funcione en un entorno de red; no cubre dispositivos de red como *switches*, *bridges* o *hubs* u otros dispositivos de red, pero unos conocimientos básicos son imprescindibles para facilitar las tareas de administración.

En este curso trataremos primero aquellos aspectos locales del propio sistema, y en una segunda parte veremos las tareas de administración de red y sus servicios.

Ya hemos comentado el problema de determinar qué es exactamente un administrador de sistemas, ya que en el mercado laboral informático no está demasiado claro. Era común pedir administradores de sistemas según categorías (establecidas en las empresas) de programador o ingenieros de software, las cuales no se adecuan correctamente.

Un programador es básicamente un productor de código; en este caso, un administrador obtendría poca producción, ya que en algunas tareas puede ser necesario, pero en otras no. Normalmente, será deseable que el administrador tenga más o menos conocimientos dependiendo de la categoría laboral:

a) Alguna carrera o diplomatura universitaria, preferible en informática, o en algún campo directamente relacionado con la empresa u organización.

El perfil del administrador suele incluir estudios informáticos o afines a la organización junto con experiencia demostrada en el campo y conocimientos amplios de sistemas heterogéneos y tecnologías de red.

b) Suele pedirse de 1 a 3 años de experiencia como administrador (a no ser que el puesto sea para ayudante de uno ya existente). La experiencia también puede ampliarse de 3 a 5 años.

c) Familiaridad o conocimientos amplios de entornos de red y servicios. Protocolos TCP/IP, servicios de ftp, telnet, ssh, http, nfs, nis, ldap, etc.

d) Conocimientos de lenguajes de *script* para prototipado de herramientas o automatización rápida de tareas (por ejemplo, shell *scripts*, Perl, tcl, Python, etc.) y experiencia en programación de un amplio rango de lenguajes (C, C++, Java, Asm, etc.).

e) Puede pedirse experiencia en desarrollo de aplicaciones grandes en cualquiera de estos lenguajes.

f) Conocimientos amplios de mercado informático, tanto de hardware como de software, en el caso de que haya que evaluar compras de material o montar nuevos sistemas o instalaciones completas.

g) Experiencia en más de una versión de UNIX (o sistemas GNU/Linux), como Solaris, AIX, AT&T SystemV, BSD, etc.

h) Experiencia en sistemas operativos no UNIX, sistemas complementarios que pueden encontrarse en la organización: Windows 9x/NT/2000/XP/Vista, Mac Os, VMS, sistemas IBM, etc.

i) Sólidos conocimientos del diseño e implementación de UNIX, mecanismos de páginas, intercambio, comunicación interproceso, controladores, etc.,

por ejemplo, si las tareas de administración incluyen optimización de sistemas (*tuning*).

j) Conocimientos y experiencia en seguridad informática: construcción de cortafuegos (*firewalls*), sistemas de autentificación, aplicaciones de criptografía, seguridad del sistema de ficheros, herramientas de seguimiento de seguridad, etc.

k) Experiencia en bases de datos, conocimientos de SQL, etc.

l) Instalación y reparación de hardware y/o cableados de red y dispositivos.

5. Tareas del administrador

Según hemos descrito, podríamos separar las tareas de un administrador GNU/Linux (o UNIX en general) [Lev02] en dos partes principales: administración del sistema y administración de red. En los siguientes puntos mostramos de forma resumida en qué consisten en general estas tareas en los sistemas GNU/LINUX (o UNIX); la mayor parte del contenido se va a tratar con cierto detalle en este manual del curso; otra parte, por cuestiones de espacio o complejidad, se explicará superficialmente o no se tratará.

Las tareas de administración engloban una serie de conocimientos y técnicas de los cuales en este curso sólo podemos ver la "punta del *iceberg*"); en todo caso, en la bibliografía adjunta a cada unidad se aportarán referencias para ampliar dichos temas. Como se verá, hay una amplia bibliografía para casi cualquier punto que se trate.

Las tareas de administración del sistema se podrían resumir, por una parte, en la administración local del sistema, y por otra, en la administración de red.

Tareas de administración local del sistema (sin un orden concreto)

- Arranque y apagado del sistema: cualquier sistema basado en UNIX tiene unos sistemas de arranque y apagado valorables, de manera que podemos configurar qué servicios ofrecemos en el arranque de la máquina y cuándo hay que pararlos, o programar el apagado del sistema para su mantenimiento.

- Gestión de usuarios y grupos: dar cabida a los usuarios es una de las principales tareas de cualquier administrador. Habrá que decidir qué usuarios podrán acceder al sistema, de qué forma y bajo qué permisos; y establecer comunidades mediante los grupos. Un caso particular será el de los usuarios de sistema, pseudousuarios dedicados a tareas del sistema.

- Gestión de recursos del sistema: qué ofrecemos, cómo lo ofrecemos y a quién damos acceso.

- Gestión de los sistemas de ficheros: el ordenador puede disponer de diferentes recursos de almacenamiento de datos y dispositivos (disquetes, discos duros, ópticos, etc.) con diferentes sistemas de acceso a los ficheros. Pueden ser permanentes o extraíbles o temporales, con lo cual habrá que modelar y gestionar los procesos de montaje y desmontaje de los sistemas de ficheros que ofrezcan los discos o dispositivos afines.

- Cuotas del sistema: cualquier recurso que vaya a ser compartido tiene que ser administrado, y según la cantidad de usuarios, habrá que establecer un sistema de cuotas para evitar el abuso de los recursos por parte de los usuarios o establecer clases (o grupos) de usuarios diferenciados por mayor o menor uso de recursos. Suelen ser habituales sistemas de cuotas de espacio de disco, o de impresión, o de uso de CPU (tiempo de computación usado).

- Seguridad del sistema: seguridad local, sobre protecciones a los recursos frente a usos indebidos o accesos no permitidos a datos del sistema o de otros usuarios o grupos.

- *Backup* y restauración del sistema: es necesario establecer políticas periódicas (según importancia de los datos), de copias de seguridad de los sistemas. Hay que establecer periodos de copia que permitan salvaguardar nuestros datos, de fallos del sistema (o factores externos) que puedan provocar pérdidas o corrupción de datos.

- Automatización de tareas rutinarias: muchas de las tareas rutinarias de la administración o del uso habitual de la máquina pueden ser fácilmente automatizables, ya debido a su simplicidad (y por lo tanto, a la facilidad de repetirlas), como a su temporalización, que hace que tengan que ser repetidas en periodos concretos. Estas automatizaciones suelen hacerse, bien mediante programación por lenguajes interpretados de tipo *script* (*shells*, Perl, etc.), como por la inclusión en sistemas de temporización (*crontab*, *at*...).

- Gestión de impresión y colas: los sistemas UNIX pueden utilizarse como sistemas de impresión para controlar una o más impresoras conectadas al sistema, así como gestionar las colas de trabajo que los usuarios o aplicaciones puedan enviar a las mismas.

- Gestión de módems y terminales. Estos dispositivos suelen ser habituales en entornos no conectados a red local ni a banda ancha:

  – Los módems permiten una conexión a la red por medio de un intermediario (el ISP o proveedor de acceso), o bien la posibilidad de conectar a nuestro sistema desde el exterior por acceso telefónico desde cualquier punto de la red telefónica.

  – En el caso de los terminales, antes de la introducción de las redes solía ser habitual que la máquina UNIX fuese el elemento central de cómputo, con una serie de terminales "tontos", que únicamente se dedicaban a visualizar la información o a permitir la entrada de información por medio de teclados externos; solía tratarse de terminales de tipo serie o paralelo. Hoy en día, todavía suelen ser habituales en entornos industriales, y en nuestro sistema GNU/Linux de escritorio tenemos una cosa

particular, que son los terminales de texto "virtuales", a los que se accede mediante las teclas Alt+Fxx.

- *Accounting* (o *log*) de sistema: para poder verificar el funcionamiento correcto de nuestro sistema, es necesario llevar políticas de *log* que nos puedan informar de los posibles fallos del sistema o del rendimiento que se obtiene de una aplicación, servicio o recurso hardware. O bien permitir resumir los recursos gastados, los usos realizados o la productividad del sistema en forma de informe.

- *System performance tunning*: técnicas de optimización del sistema para un fin dado. Suele ser habitual que un sistema esté pensado para una tarea concreta y que podamos verificar su funcionamiento adecuado (por ejemplo, mediante *logs*), para examinar sus parámetros y adecuarlos a las prestaciones que se esperan.

- Personalización del sistema: reconfiguración del *kernel*. Los *kernels*, por ejemplo en GNU/Linux, son altamente personalizables, según las características que queramos incluir y el tipo de dispositivos que tengamos o esperemos tener en nuestra máquina, así como los parámetros que afecten al rendimiento del sistema o que consigan las aplicaciones.

Tareas de administración de red

- Interfaz de red y conectividad: el tipo de interfaz de red que utilizamos, ya sea el acceso a una red local, la conexión a una red mayor, o conexiones del tipo banda ancha con tecnologías DSL o RDSI. Además, el tipo de conectividades que vamos a tener, en forma de servicios o peticiones.

- *Routing* de datos: los datos que circularán, de dónde o hacia dónde se dirigirán, dependiendo de los dispositivos de red disponibles y de las funciones de la máquina en red; posiblemente, será necesario redirigir el trafico desde/hacia uno o más sitios.

- Seguridad de red: una red, sobre todo si es abierta (como Internet) a cualquier punto exterior, es una posible fuente de ataques y, por lo tanto, puede comprometer la seguridad de nuestros sistemas o los datos de nuestros usuarios. Hay que protegerse, detectar e impedir posibles ataques con una política de seguridad clara y eficaz.

- Servicios de nombres: en una red hay infinidad de recursos disponibles. Los servicios de nombres nos permiten nombrar objetos (como máquinas y servicios) para poderlos localizar. Con servicios como el DNS, DHCP, LDAP, etc., se nos permitirá localizar servicios o equipos *a posteriori*...

- NIS (*Network Information Service*): las grandes organizaciones han de tener mecanismos para poder organizar de forma efectiva los recursos y el acceso

a ellos. Las formas UNIX estándar, como los *logins* de usuarios con control por *passwords* locales, son efectivos con pocas máquinas y usuarios, pero cuando tenemos grandes organizaciones, con estructuras jerárquicas, usuarios que pueden acceder a múltiples recursos de forma unificada o separada por diferentes permisos,... los métodos UNIX sencillos se muestran claramente insuficientes o imposibles. Entonces se necesitan sistemas más eficaces para controlar toda esta estructura. Servicios como NIS, NIS+, LDAP nos permiten organizar de forma efectiva toda esta complejidad.

- NFS (*Network Fylesystems*): a menudo, en las estructuras de sistemas en red es necesario compartir informaciones (como los propios ficheros) por parte de todos o algunos de los usuarios. O sencillamente, debido a la distribución física de los usuarios, es necesario un acceso a los ficheros desde cualquier punto de la red. Los sistemas de ficheros por red (como NFS) permiten un acceso transparente a los ficheros, independientemente de nuestra situación en la red.

- *UNIX remote commands*: UNIX dispone de comandos transparentes a la red, en el sentido de que, independientemente de la conexión física, es posible ejecutar comandos que muevan información por la red o permitan acceso a algunos servicios de las máquinas. Los comandos suelen tener una "r" delante, con el sentido de 'remoto', por ejemplo: *rcp*, *rlogin*, *rsh*, *rexec*, etc., que permiten las funcionalidades indicadas de forma remota en la red.

- Aplicaciones de red: aplicaciones de conexión a servicios de red, como telnet (acceso interactivo), ftp (transmisión de ficheros), en forma de aplicación cliente que se conecta a un servicio servido desde otra máquina. O bien que nosotros mismos podemos servir con el servidor adecuado: servidor de telnet, servidor ftp, servidor web, etc.

- Impresión remota: acceso a servidores de impresión remotos, ya sea directamente a impresoras remotas o bien a otras máquinas que ofrecen sus impresoras locales. Impresión en red de forma transparente al usuario o aplicación.

- Correo electrónico: uno de los primeros servicios proporcionados por las máquinas UNIX es el servidor de correo, que permite, ya sea el almacenamiento de correo o un punto de retransmisión de correo hacia otros servidores, si no iba dirigido a usuarios propios de su sistema. Para el caso web, también de forma parecida, un sistema UNIX con el servidor web adecuado ofrece una plataforma excelente para web. UNIX tiene la mayor cuota de mercado en cuanto a servidores de correo y web, y es uno de los principales mercados, donde tiene una posición dominante. Los sistemas GNU/Linux ofrecen soluciones de código abierto para correo y web y conforman uno de sus principales usos.

- X Window: un caso particular de interconexión es el sistema gráfico de los sistemas GNU/Linux (y la mayor parte de UNIX), X Window. Este sistema permite una transparencia total de red y funciona bajo modelos cliente servidor; permite que el procesamiento de una aplicación esté desligado de la visualización y de la interacción por medio de dispositivos de entrada, por lo que éstos se sitúan en cualquier parte de la red. Por ejemplo, podemos estar ejecutando una determinada aplicación en una máquina UNIX cuando desde otra visualizamos en pantalla los resultados gráficos, y entramos datos con el teclado y ratón locales de forma remota. Es más, el cliente, llamado cliente X, es tan sólo un componente software que puede ser portado a otros sistemas operativos, permitiendo ejecutar aplicaciones en una máquina UNIX y visualizarlas en cualquier otro sistema. Un caso particular son los llamados terminales X, que son básicamente una especie de terminales "tontos" gráficos que sólo permiten visualizar o interactuar (por teclado y ratón) con una aplicación en ejecución remota.

6. Distribuciones de GNU/Linux

Al hablar de los orígenes de los sistemas GNU/Linux, hemos comprobado que no había un único sistema operativo claramente definido. Por una parte, hay tres elementos software principales que componen un sistema GNU/Linux:

1. El *kernel* Linux: como vimos, el *kernel* es tan sólo la pieza central del sistema. Pero sin las aplicaciones de utilidad, *shells*, compiladores, editores, etc. no podríamos tener un sistema entero.

2. Las aplicaciones GNU: en el desarrollo de Linux, éste se vio complementado con el software de la FSF existente del proyecto GNU, que le aportó editores (como *emacs*), compilador (*gcc*) y utilidades diversas.

3. Software de terceros: normalmente de tipo de código abierto en su mayor parte. Todo sistema GNU/Linux se integra además con software de terceros que permite añadir una serie de aplicaciones de amplio uso, ya sea el propio sistema gráfico de X Windows, servidores como el de Apache para web, navegadores, etc. Asimismo, puede ser habitual incluir algún software propietario, dependiendo del carácter libre que en mayor o menor grado quieran disponer los creadores de la distribución.

Al ser la mayoría del software de tipo de código abierto o libre, ya sea el *kernel*, software GNU o de terceros, normalmente hay una evolución más o menos rápida de versiones, ya sea por medio de corrección de errores o nuevas prestaciones. Esto obliga a que en el caso de querer crear un sistema GNU/Linux, tengamos que escoger qué software queremos instalar en el sistema, y qué versiones concretas de este software.

El mundo GNU/Linux no se limita a una empresa o comunidad particular, con lo que ofrece a cada uno la posibilidad de crear su propio sistema adaptado a sus necesidades.

Normalmente, entre el conjunto de estas versiones siempre se encuentran algunas que son estables, y otras que están en desarrollo, en fases alfa o beta, que pueden tener errores o ser inestables, por lo que habrá que tener cuidado a la hora de crear un sistema GNU/Linux, con la elección de las versiones. Otro problema añadido es la selección de alternativas, el mundo de GNU/Linux es lo suficientemente rico para que haya más de una alternativa para un mismo producto de software. Hay que elegir entre las alternativas posibles, incorporar algunas o todas, si queremos ofrecer al usuario libertad para escoger su software.

Ejemplo

Un caso práctico lo forman los gestores de escritorio de X Window, en que, por ejemplo, nos ofrecen (principalmente) dos entornos de escritorio diferentes como Gnome y KDE; los dos tienen características parecidas y aplicaciones semejantes o complementarias.

En el caso de un distribuidor de sistemas GNU/Linux, ya sea comercial o bien una organización sin beneficio propio, dicho distribuidor tiene como responsabilidad generar un sistema que funcione, seleccionando las mejores versiones y productos software que puedan conseguirse.

En este caso, una distribución GNU/Linux [Dis] es una colección de software que forma un sistema operativo basado en el *kernel* Linux.

Un dato importante a tener en cuenta, y que causa más de una confusión, es que, como cada uno de los paquetes de software de la distribución tendrá su propia versión (independiente de la distribución en que esté ubicado), el número de distribución asignado no mantiene una relación con las versiones de los paquetes software.

El número de distribución sólo sirve para comparar las distribuciones que genera un mismo distribuidor, y no permite comparar entre otras distribuciones. Si queremos hacer comparaciones entre distribuciones, tendremos que examinar los paquetes software principales y sus versiones para poder determinar qué distribución aporta más novedades.

Ejemplo

Pongamos un ejemplo de algunas versiones actuales (las versiones que aparecen se refieren a finales del año 2003):

a) *Kernel* Linux: actualmente podemos encontrar distribuciones que ofrecen uno o más *kernels*, como los de la serie antigua 2.4.x o generalmente los últimos 2.6.x en revisiones (el número x) de diversa actualidad.

b) La opción en el sistema gráfico X Window, en versión de código abierto, que podemos encontrar prácticamente en todos los sistemas GNU/Linux, ya sean algunas versiones residuales de Xfree86 como las que manejan versiones 4.x.y o bien el nuevo proyecto Xorg (siendo un fork del anterior en 2003), que goza de más popularidad en diversas versiones 6.x o 7.x.

c) Gestor de ventanas o escritorio: podemos disponer de Gnome o KDE, o ambos; Gnome con versiones 2.x o KDE 3.x.y.

Podríamos obtener, por ejemplo, una distribución que incluyese *kernel* 2.4, con XFree 4.4 y Gnome 2.14; o bien otra, por ejemplo, *kernel* 2.6, Xorg 6.8,

KDE 3.1. ¿Cuál es mejor?, es difícil compararlas, ya que suponen una mezcla de elementos, y dependiendo de cómo se haga la mezcla, el producto saldrá mejor o peor, y más o menos adaptado a las necesidades del usuario. Normalmente, el distribuidor mantiene un compromiso entre la estabilidad del sistema y la novedad de las versiones incluidas. Así como proporcionar software de aplicación atrayente para los usuarios de la distribución, ya sea éste generalista, o especializado en algún campo concreto.

En general, podría hacerse un mejor análisis de distribuciones a partir de los siguientes apartados, que habría que comprobar en cada una:

a) Versión del núcleo Linux: la versión viene indicada por unos números *X.Y.Z*, donde normalmente *X* es la versión principal, que representa los cambios importantes del núcleo; *Y* es la versión secundaria, y normalmente implica mejoras en las prestaciones del núcleo: *Y* es par en los núcleos estables e impar en los desarrollos o pruebas. Y *Z* es la versión de construcción, que indica el número de la revisión de *X.Y*, en cuanto a parches o correcciones hechas. Los distribuidores no suelen incluir la ultima versión del núcleo, sino la que ellos hayan probado con más frecuencia y puedan verificar que es estable para el software, y componentes, que ellos incluyen. Este esquema de numeración clásico (que se siguió durante las ramas 2.4.x, hasta los inicios de la 2.6), tuvo algunas modificaciones, para adaptarse al hecho de que el kernel (rama 2.6.x) se vuelve más estable, y cada vez las revisiones son menores (significan un salto de versión de los primeros números), pero el desarrollo es continuo y frenético. En los últimos esquemas, se llegan a introducir cuartos números, para especificar de Z cambios menores, o diferentes posibilidades de la revisión (con diferentes parches añadidos). La versión así definida con cuatro números es la que se considera estable (*stable)*. También son usados otros esquemas para las diversas versiones de test (normalmente no recomendables para entornos de producción), como sufijos -rc (*release candidate*), los -mm, que son kernels experimentales con pruebas de diferentes técnicas, o los -git, que son una especie de "foto" diaria del desarrollo del kernel. Estos esquemas de numeración están en constante cambio para adaptarse a la forma de trabajar de la comunidad del kernel, y a sus necesidades para acelerar el desarrollo del kernel.

b) Formato de empaquetado: es el mecanismo empleado para instalar y administrar el software de la distribución. Se suele conocer por el formato de los paquetes de software soportados. En este caso suelen estar los formatos RPM, DEB, tar.gz, mdk, aunque cada distribución suele tener posibilidad de utilizar varios formatos, suele tener uno por defecto. El software acostumbra a venir con sus archivos en un paquete que incluye información sobre su instalación y posibles dependencias con otros paquetes de software. El empaquetado es importante si se usa software de terceros que no venga con la distribución, ya que el software puede encontrarse sólo en algunos sistemas de paquetes, o incluso en uno sólo.

c) Estructura del sistema de archivos: la estructura del sistema de archivos principal (/) nos indica dónde podemos encontrar nuestros archivos (o los propios del sistema) en el *filesystem*. En GNU/Linux y UNIX hay algunos estándares de colocación de los archivos (como veremos en la unidad de herramientas), como por ejemplo el FHS (*filesystem hierarchy standard*) [Lin03b]. Así, si tenemos una idea del estándar, sabremos dónde encontrar la mayor parte de los archivos; luego depende de que la distribución lo siga más o menos y de que nos avisen de los cambios que hayan hecho.

d) *Scripts* de arranque del sistema: los sistemas UNIX y GNU/Linux incorporan unos guiones de arranque (o *shell scripts*) que indican cómo debe arrancar la máquina y cuál será el proceso (o fases) que se van a seguir, así como lo que deberá hacerse en cada paso. Para este arranque hay dos modelos, los de SysV o BSD (es una diferencia de las dos ramas de UNIX principales); y cada distribución podría escoger uno u otro. Aunque los dos sistemas tienen la misma funcionalidad, son diferentes en los detalles, y esto será importante en los temas de administración (lo veremos en la administración local). En nuestro caso, los sistemas analizados, tanto Fedora como Debian, utilizan el sistema de SysV (será el que veremos en la unidad local), pero hay otras distribuciones como Slackware que utilizan el otro sistema BSD. Y existen determinadas propuestas para nuevas opciones en este aspecto de arranque.

e) Versiones de la biblioteca del sistema: todos los programas (o aplicaciones) que tenemos en el sistema dependen para su ejecución de un número (mayor o menor) de bibliotecas de sistema. Estas bibliotecas, normalmente de dos tipos, ya sean estáticas unidas al programa (archivos *libxxx.a*) o las dinámicas que se cargan en tiempo de ejecución (archivos *libxxx.so*), proporcionan gran cantidad de código de utilidad o de sistema que utilizarán las aplicaciones. La ejecución de una aplicación puede depender de la existencia de unas bibliotecas adecuadas y del número de versión concreto de estas bibliotecas (no es lo recomendable, pero puede suceder). Un caso bastante habitual es la biblioteca GNU C library, la biblioteca estándar de C, también conocida como *glibc*. Puede suceder que una aplicación nos pida que dispongamos de una versión concreta de la *glibc* para poder ejecutarse o compilarse. Es un caso bastante problemático, y por ello, uno de los parámetros que valoran la distribución es conocer qué versión de la *glibc* lleva, y los posibles paquetes adicionales de versiones de compatibilidad con versiones antiguas. El problema aparece al intentar ejecutar o compilar un producto de software muy antiguo en una distribución moderna, o bien un producto de software muy nuevo en una distribución antigua.

El mayor cambio llegó al pasar a una *glibc* 2.0, en que había que recompilar todos los programas para poder ejecutarlos correctamente, y en las diferentes revisiones nuevas de numeración 2.x ha habido algunos cambios menores que podían afectar a alguna aplicación. En muchos casos, los paquetes de software comprueban si se tiene la versión correcta de la *glibc*, o en el mismo nombre mencionan la versión que hay que utilizar (ejemplo: *paquete-xxx-glibc2.rpm*).

f) Escritorio X Window: el sistema X Window es el estándar gráfico para GNU/Linux como visualización de escritorio. Fue desarrollado en el MIT en 1984 y prácticamente todos los UNIX tienen una versión del mismo. Las distribuciones GNU/Linux disponen de diferentes versiones como la Xfree86 o la Xorg. Normalmente, el X Window es una capa gráfica intermedia que confía a otra capa denominada gestor de ventanas la visualización de sus elementos. Además, podemos combinar el gestor de ventanas con utilidades y programas de aplicación variados para formar lo que se denomina un entorno de escritorio.

Linux tiene principalmente dos entornos de escritorio: Gnome y KDE. Cada uno tiene la particularidad de basarse en una biblioteca de componentes propios (los diferentes elementos del entorno como ventanas, botones, listas, etc.): *gtk+* (en Gnome) y *Qt* (en KDE), que son las principales bibliotecas gráficas que se usan para programar aplicaciones en estos entornos. Pero además de estos entornos, hay muchos otros, gestores de ventanas o escritorios: XCFE, Motif, Enlightement, BlackIce, FVWM, etc., de modo que la posibilidad de elección es amplia. Además, cada uno de ellos permite cambiar la apariencia (*look & feel*) de ventanas y componentes al gusto del usuario, o incluso crearse el suyo propio.

g) Software de usuario: software añadido por el distribuidor, en su mayoría de tipo Open Source, para las tareas habituales (o no tanto, para algunos campos muy especializados).

Las distribuciones habituales son tan grandes, que pueden encontrarse de centenares a miles de estas aplicaciones (muchas de las distribuciones tienen de 1 a 4 CD (aproximadamente 1 DVD), de aplicaciones extra. Estas aplicaciones cubren casi todos los campos, desde el hogar hasta administrativos o científicos. Y en algunas distribuciones se añade software propietario de terceros (como, por ejemplo, alguna *suite* ofimática del tipo Office), software de servidor preparado por el distribuidor, como por ejemplo un servidor de correo, un servidor web seguro, etc.

Así es cómo cada distribuidor suele sacar diferentes versiones de su distribución, por ejemplo, a veces hay distinciones entre una versión personal, profesional o de tipo servidor.

El sistema GNU/Linux de fondo es el mismo, sólo hay diferencias (que se pagan en algunos casos) en el software añadido (en general, obra de la misma casa distribuidora). Por ejemplo, en servidores web o en servidores correo, ya sean desarrollos propios, optimizados o mejorados. O bien la inclusión de mejores herramientas, desarrolladas por el fabricante de la distribución.

A menudo, este coste económico extra no tiene mucho sentido, ya que el software estándar es suficiente (con un poco de trabajo extra de administración); pero para las empresas puede ser interesante porque reduce tiempo de instala-

ción y mantenimiento de los servidores, y además optimiza algunas aplicaciones y servidores críticos para la gestión informática de la empresa.

6.1. Debian

El caso de Debian [Debb] es especial, en el sentido de que es una distribución guiada por una comunidad sin fines comerciales, aparte de mantener su distribución y promocionar el uso del software de código abierto y libre.

Debian es una distribución apoyada por una comunidad entusiasta de usuarios y desarrolladores propios, basada en el compromiso de la utilización de software libre.

El proyecto Debian se fundó en 1993 para crear la distribución Debian GNU/Linux. Desde entonces se ha vuelto bastante popular y rivaliza en uso con otras distribuciones comerciales como Red Hat o Mandrake. Por ser un proyecto comunitario, el desarrollo de esta distribución se rige por una serie de normas o políticas; existen unos documentos llamados "Contrato social Debian", que mencionan la filosofía del proyecto en su conjunto, y las políticas Debian, que especifican en detalle cómo se implementa su distribución.

Nota

Los documentos "Contrato social Debian" son consultables en: debian.org.

La distribución Debian está bastante relacionada con los objetivos de la FSF y su proyecto de Software Libre GNU; por esta razón, incluyen siempre en su nombre: "Debian GNU/Linux"; además, su texto del contrato social ha servido como base de las definiciones de código abierto. En cuanto a las políticas, todo aquel que quiera participar en el proyecto de la distribución, tiene que seguirlas. Aunque no se trate de un colaborador, estas políticas pueden ser interesantes porque explican cómo es la distribución Debian.

Figura 2

Mencionamos también un aspecto práctico de cara a los usuarios finales: Debian ha sido siempre una distribución difícil. Suele ser la distribución que usan los *hackers* de Linux, en el buen sentido de los que destripan el *kernel*, aportan modificaciones, programadores de bajo nivel, los que desean estar a la última para probar software nuevo, los que quieren probar los desarrollos del *kernel* que todavía no han sido publicados... o sea, todo tipo de fauna de "locos" por GNU/Linux.

Las versiones anteriores de Debian se habían hecho famosas por su dificultad de instalación. La verdad es que no se habían hecho esfuerzos para que fuese fácil para los no expertos. Pero las cosas con el tiempo han mejorado. Ahora, la instalación, no sin ciertos conocimientos, puede hacerse guiada por menús (eso sí, textuales, a diferencia de otras comerciales que son absolutamente gráficos), y hay programas que facilitan la instalación de los paquetes. Pero aun así, los primeros intentos pueden llegar ser un poco traumáticos.

Normalmente, suelen ser variantes (las llaman "sabores") de la distribución Debian. En este momento hay tres ramas de la distribución: la *stable*, la *testing* y la *unstable*. Y, como sus nombres indican, la *stable* es la que está destinada a entornos de producción (o usuarios que desean estabilidad), la *testing* ofrece software más nuevo que ha sido testado mínimamente (podríamos decir que es una especie de versión beta de Debian) y que pronto van a ser incluidos en la *stable*. Y la *unstable* es la que presenta las últimas novedades de software, cuyos paquetes cambian en plazos muy cortos; en una semana, e incluso cada día, pueden cambiar varios paquetes. Todas las distribuciones son actualizables desde diversas fuentes (CD, ftp, web) por un sistema denominado APT que maneja los paquetes software DEB de Debian. Las tres distribuciones tienen nombres más comunes asignados (una foto actual):

- Etch (*stable*)
- Lenny (*testing*)
- Sid (*unstable*)

La versión previa *stable* se denominaba Sarge (3.1r6), anteriormente Woody (era la 3.0). La más actual (durante 2007), es la Debian GNU/Linux Etch (4.0). Las versiones más extendidas son la Etch y la Sid, que son los dos extremos. La Sid no está recomendada para entornos (de producción) de trabajo diario, porque puede traer características a medias que aún se están probando y pueden fallar (aunque no es habitual); es la distribución que suelen usar los *hackers* de GNU/Linux. Además, esta versión cambia casi a diario; suele ser normal que, si se quiere actualizar a diario, existan de 10 a 20 paquetes de software nuevos por día (o incluso más en algunos momentos puntuales de desarrollo).

La Etch es quizás la mejor elección para el sistema de trabajo diario, se actualiza periódicamente para cubrir nuevo software o actualizaciones (como las de seguridad). Normalmente, no dispone del último software, y éste no se incluye hasta que la comunidad lo haya verificado en un amplio rango de pruebas.

Vamos a comentar brevemente algunas características de esta distribución (las versiones son las que se encuentran por defecto en la Etch y en Sid a día de hoy):

a) La distribución (estable) actual consta de entre 1 a 21 CD (o 3 DVD) de la última versión disponible de Etch. Normalmente hay diferentes posibilidades dependiendo del conjunto de software que nos encontremos en soporte físico (CD o DVD), o bien lo que deseamos posteriormente descargar desde la red, con lo cual sólo necesitamos un CD básico, más el acceso a red, para descargar el resto según demanda. Esta distribución puede comprarse (a precios simbólicos de soporte físico, y de esta manera contribuimos a mantener la distribución) o puede bajarse desde debian.org o sus *mirrors*.

b) La testing y unstable no suelen tener CD oficiales, sino que puede convertirse una debian *stable* a *testing* o *unstable* mediante cambios de configuración del sistema de paquetes APT.

c) Núcleo Linux: utilizaban núcleos de la serie 2.4.x por defecto, incluye 2.6.x de forma opcional, y en las últimas versiones ya por defecto. El enfoque de Debian en *stable* es potenciar la estabilidad y dejar a los usuarios la opción de otro producto más actualizado de software, si lo necesitan (en *unstable* o *testing*).

d) Formato de empaquetado: Debian soporta uno de los que más prestaciones ofrece, el APT. Los paquetes de software tienen un formato denominado DEB. El APT es una herramienta de más alto nivel para gestionarlos y mantener una base de datos de los instalables y los disponibles en el momento. Además, el sistema APT puede obtener software de varias fuentes, ya sea desde CD, ftp, web.

e) El sistema con APT es actualizable en cualquier momento, mediante lista de sitios de fuentes de software Debian (fuentes APT), que pueden ser los sitios Debian por defecto (debian.org) o de terceros. No estamos así ligados a una empresa única ni a ningún sistema de pago por suscripción.

f) Algunas de las versiones utilizadas, por ejemplo, son: Xfree86(4.x), *glibc* (2.3.x)... Debian Sid tiene Xorg (7.1), *glibc* (2.3.x)...

g) En el escritorio acepta tanto Gnome 2.16.x (por defecto) como KDE 3.3.x (K Desktop Environment). Unstable con Gnome 2.18.x, y KDE 3.5.x.

h) En cuanto a aplicaciones destacables, incluye la mayoría de las que solemos encontrar en las distribuciones de GNU/Linux; en Sid: editores como *emacs* (y *xemacs*), compilador *gcc* y herramientas, servidor web Apache, navegador web Mozilla (o firefox), software Samba para compartir archivos con Windows, etc.

i) Incluye también suites ofimáticas como OpenOffice y KOffice.

j) Debian incluye muchos ficheros de configuración personalizados para su distribución en directorios de /etc.

k) Debian usa por defecto el gestor de arranque *lilo,* aunque puede hacer uso también de *Grub*.

l) La configuración de la escucha de los servicios de red TCP/IP, que se realiza como en la mayoría de UNIX, con el servidor inetd (/etc/inetd.conf). Aunque dispone también de xinetd opcional, una opción que toma más peso.

m) Hay muchas distribuciones GNU/Linux más, basadas en Debian, ya que el sistema puede adaptarse fácilmente para hacer distribuciones más pequeñas o más grandes, o con más o menos software adaptado a un segmento. Una de las más famosas es Knoppix, una distribución de un único CD, tipo LiveCD (de ejecución en CD), que es muy usada para demos de GNU/Linux, o para probarlo en una máquina sin hacer una instalación previa, ya que arranca y se ejecuta desde CD, aunque también puede instalarse en disco duro y convertirse en una Debian estándar. Linex es otra distribución que ha conseguido bastante fama por su desarrollo apoyado por una Administración, la de la comunidad autónoma de Extremadura. Por otra parte encontramos a Ubuntu, una de las distribuciones que ha obtenido más

amplia repercursión (superando incluso a Debian en varios aspectos), por sus facilidades para construir una alternativa de escritorio.

Nota

Debian puede usarse como base para otras distribuciones; por ejemplo, Knoppix es una distribución basada en Debian que puede ejecutarse desde el CD sin necesidad de instalarse en disco. Linex es una distribución Debian adaptada por la administración de la Comunidad de Extremadura en su proyecto de adoptar software de código abierto. Y Ubuntu es una distribución optimizada para entornos de escritorio.

Figura 3. Entorno Debian Sid con Gnome 2.14

6.2. Fedora Core

Red Hat inc. [Redh] es una de las principales firmas comerciales del mundo GNU/Linux, con una de las distribuciones con más éxito. Bob Young y Marc Ewing crearon Red Hat Inc en 1994. Estaban interesados en los modelos de software de código abierto y pensaron que sería una buena manera de hacer negocio. Su principal producto es su distribución Red Hat Linux (que abreviaremos como Red Hat), que está abierta a diferentes segmentos de mercado, tanto al usuario individual (versiones personal y profesional), como a las medianas o grandes empresas (con su versión Enterprise y sus diferentes subversiones).

Red Hat Linux es la principal distribución comercial de Linux, orientada tanto a mercado personal de escritorio como a servidores de gama alta. Además, Red Hat Inc es una de las empresas que más colaboran con el desarrollo de Linux, ya que varios miembros importantes de la comunidad trabajan para ella.

Figura 4

Aunque trabajan con un modelo de código abierto, se trata de una empresa, y por lo tanto sus fines son comerciales, por ello suelen añadir a su distribución básica valores por medio de contratos de soporte, suscripciones de actualización y otros métodos. En el caso empresarial, añaden software personalizado (o propio), para hacer que se adecue más el rendimiento a los fines de la empresa, ya sea por servidores optimizados o por software de utilidad propio de Red Hat.

A partir de cierto momento (finales 2003), Red Hat Linux (versión 9.x), su versión de GNU/Linux para escritorio se da por discontinuada, y aconsejan a sus clientes a migrar a las versiones empresariales de la firma, que seguirán siendo las únicas versiones soportadas oficialmente por la firma.

Nota

Ver: http://fedora.redhat.com

En este momento Red Hat decide iniciar el proyecto abierto a la comunidad denominado Fedora [Fed], con el objetivo de realizar una distribución guiada por comunidad (al estilo Debian, aunque con fines diferentes), que se denominará Fedora Core. De hecho se persigue crear un laboratorio de desarrollo abierto a la comunidad que permita probar la distribución, y a su vez guiar los desarrollos comerciales de la empresa en sus distribuciones empresariales.

En cierto modo, algunos críticos señalan que se usa a la comunidad como *betatesters* de las tecnologías que posteriormente se incluirán en productos comerciales. Además, este modelo es utilizado posteriormente por otras compañías para crear a su vez modelos duales de distribuciones de comunidad a la vez que comerciales. Aparecen ejemplos como OpenSuse (a partir de la comercial SuSe), o Freespire (a partir de Linspire).

Normalmente, el duo Red Hat y la comunidad Fedora presentan una cierta visión conservadora (menos acentuada en Fedora) de los elementos software que añade a su distribución, ya que su principal mercado de destino es el empresarial, e intenta hacer su distribución lo más estable posible, a pesar de que no cuentan con las últimas versiones. Lo que sí hace como valor añadido es depurar extensamente el *kernel* de Linux con su distribución, y genera correcciones y parches para mejorar su estabilidad. A veces, puede llegar a deshabilitar alguna funcionalidad (*drivers*) del *kernel*, si considera que éstos no son lo suficientemente estables. También ofrece muchas utilidades en el entorno gráfico y programas gráficos propios, incluidas unas cuantas herramientas de ad-

ministración; en cuanto a los entornos gráficos, utiliza tanto Gnome (por defecto) como KDE, pero mediante un entorno modificado propio denominado BlueCurve, que hace que los dos escritorios sean prácticamente iguales (ventanas, menús, etc.).

La versión que utilizaremos será la última Fedora Core disponible, a la cual denominaremos simplemente como Fedora. En general, los desarrollos y prestaciones que se mantienen suelen ser bastante parecidas en las versiones que salen *a posteriori*, con lo cual, la mayoría de comentarios serían aplicables a las diferentes versiones a lo largo del tiempo. Tenemos que tener en cuenta que la comunidad Fedora [Fed], intenta cumplir un calendario de aproximadamente 6 meses para cada nueva versión. Y se produce un cierto consenso sobre las prestaciones nuevas a introducir.

Red Hat, por contra, deja sus versiones de escritorio en manos de la comunidad, y se centra en sus negocios en las versiones empresariales (Red Hat Linux Enterprise WS, ES, y AS).

Vamos a comentar brevemente algunas características de esta distribución Fedora Core:

a) La distribución actual consiste en 5 CD, de los cuales el primero, *bootable*, sirve para su instalación. También hay CD extras que contienen documentación y código fuente de la mayoría del software instalado con la distribución. Asimismo, se provee la distribución en 1 DVD.

b) Núcleo Linux: utiliza núcleos de la serie 2.6.x, que pueden irse actualizando con el sistema de paquetes (ver unidad del *kernel*) rpm (a través de la utilidad yum por ejemplo). Red Hat, por su parte, somete el *kernel* a muchas pruebas y crea parches para solucionar problemas, que normalmente también son integrados en la versión de la comunidad Linux, ya que bastantes de los colaboradores importantes de Linux trabajan para Red Hat.

c) Formato de empaquetado: Red Hat distribuye su software mediante el sistema de paquetes RPM (*red hat package manager*), los cuales se gestionan mediante el comando *rpm* o las utilidades yum (lo comentaremos en la unidad de administración local). RPM es uno de los mejores sistemas de empaquetado existentes (al estilo del deb Debian), y algunos UNIX propietarios lo están incluyendo. Básicamente, el sistema RPM mantiene una pequeña base de datos con los paquetes instalados, y verifica que el paquete que se va instalar con el comando *rpm*, no esté ya instalado, o entre en conflicto con algún otro paquete de software o por contra falte algún paquete software o versión de éste, necesaria para la instalación. El paquete RPM es básicamente un conjunto de ficheros comprimidos junto con información de sus dependencias o del software que necesita.

d) En cuanto al arranque, utiliza *scripts* de tipo *SystemV* (que veremos en la unidad de administración local).

e) Algunas de las versiones utilizadas son: Xorg (7.x), glibc (2.5.x), etc.

f) En el escritorio acepta tanto Gnome (escritorio por defecto) como KDE de forma opcional.

g) En cuanto a aplicaciones destacables, incluye la mayoría de las que solemos encontrar en casi todas las distribuciones de GNU/Linux: editores como *emacs* (y *xemacs*), compilador *gcc* y herramientas, servidor web Apache, navegador web Firefox/Mozilla, software Samba para compartir archivos con Windows, etc.

h) Incluye también suites ofimáticas como OpenOffice y KOffice.

i) El software adicional puede obtenerse por los servicios de actualización yum (entre otros) de forma parecida al sistema APT en Debian o con diferentes herramientas de *update* incluidas, o bien por Internet mediante paquetes RPM pensados para la distribución.

j) Fedora usa el cargador de arranque Grub para arrancar la máquina por defecto.

k) La configuración de escucha de los servicios de red TCP/IP, que se lleva a cabo en la mayoría de UNIX, con el servidor inetd (/etc/inetd.conf), en Red Hat ha sido substituido por xinetd, que tiene una configuración más modular (directorio /etc/xinetd.d).

l) Dispone en arranque de un programa denominado Kudzu que se encarga de verificar cambios de hardware y detectar el hardware nuevo instalado. Se espera que en siguientes versiones quede fuera, debido a la existencia de una nueva API denominada HAL, que permite realizar esta función.

m) Hay varias distribuciones más basadas en el Red Hat original, que siguen muchas de sus características, a destacar Mandriva (antes Mandrake): una distribución francesa, que en su origen se basó en Red Hat y que sigue en los primeros puestos junto con Red Hat en las preferencias de los usuarios (sobre todo en trabajo de escritorio). Mandriva desarrolla software propio y multitud de asistentes para ayudar a la instalación y administración de las tareas más comunes, separándose de su origen con base en Red Hat. Por otra parte, las versiones empresariales de Red Hat también han originado una serie de distribuciones libres muy populares en entornos de servidor, como CentOS [Cen] (que intenta mantener una compatibilidad 100% con el Red Hat empresarial), y Scientific Linux [Sci] (especializada en el cómputo científico en proyectos de investigación científica). En cuanto al sistema de empaquetado, cabe destacar que el sistema rpm se usa en un amplio número de distribuciones, entre ellas podemos nombrar SuSe.

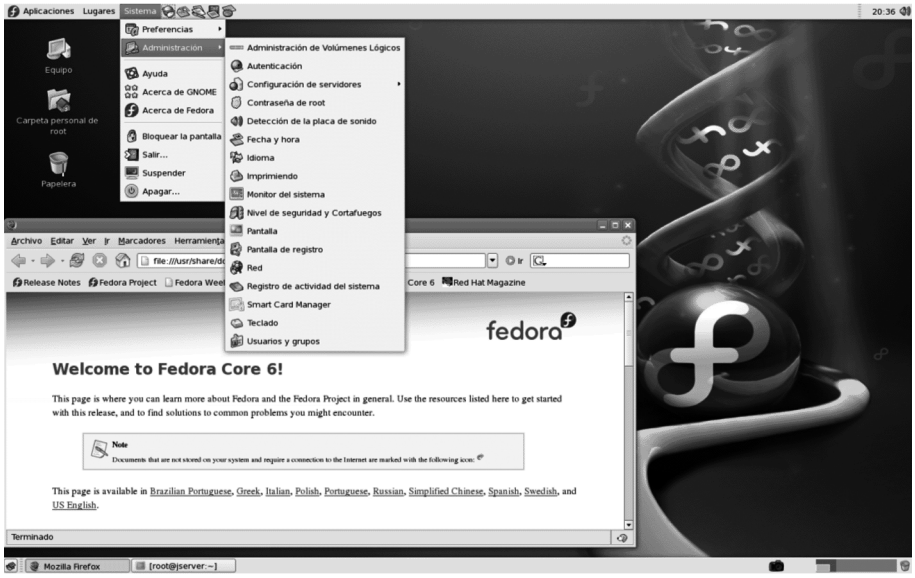

Figura 5. Escritorio Fedora Core con Gnomo

Con respecto a la distribución comunitaria Fedora Core, respecto a sus orígenes comerciales en Red Hat:

a) Es una distribución creada por la comunidad de programadores y usuarios basada en desarrollo, que no cuenta con soporte ni de actualizaciones ni de mantenimiento por parte del fabricante. Este aspecto pasa a depender de la comunidad, de forma semejante al caso de la distribución Debian GNU/Linux.

b) Las versiones se van a producir con bastante rapidez, se esperan nuevas versiones de la distribución aproximadamente cada seis meses.

c) Para la gestión de paquetes, también utiliza el sistema de paquetes RPM. Respecto al proceso de la actualización de los paquetes de la distribución o a la instalación de otros nuevos, pueden obtenerse por diferentes herramientas, ya sea vía *update,* con los canales de actualización de Fedora, o bien por los nuevos sistemas de actualización Yum y en algunos casos Apt (heredado de Debian, pero que trabaja con ficheros RPM).

d) Otras cuestiones más técnicas (algunas de las cuales veremos en los siguientes capítulos) pueden encontrarse en las notas de la versión de Fedora Core.

Nota

Ver:
http://fedora.redhat.com/docs/release-notes.

7. Qué veremos...

Una vez que hemos examinado esta introducción "filosófica" al mundo del código abierto y la historia de los sistemas UNIX y GNU/Linux, así como definido cuáles serán las tareas de la figura del administrador del sistema, pasaremos a tratar las diferentes tareas típicas que nos encontraremos durante la administración de sistemas GNU/Linux.

A continuación, examinaremos las diferentes áreas que implica la administración de sistemas GNU/Linux. En cada área, intentaremos examinar un mínimo de fundamentos teóricos que nos permitan explicar las tareas que haya que realizar y que nos permitan entender el funcionamiento de las herramientas que utilizaremos. Cada tema vendrá acompañado de algún "taller", donde veremos una pequeña sesión de trabajo de una tarea o el uso de algunas herramientas. Sólo recordaremos que, como dijimos en la presentación, el tema de la administración es amplio y cualquier intento de abarcarlo completamente (como éste) tiene que fallar por las dimensiones limitadas; por ello, en cada tema encontraréis abundante bibliografía (en forma de libros, sitios web, howto's, etc.), donde ampliar la "pequeña" introducción que habremos hecho del tema.

Los temas que veremos son los siguientes:

- En el apartado de migración, obtendremos una perspectiva del tipo de sistemas informáticos que se están utilizando y en qué ambientes de trabajo se usan; veremos, asimismo, cómo los sistemas GNU/Linux se adaptan mejor o peor a cada uno de ellos, y plantearemos una primera disyuntiva a la hora de introducir un sistema GNU/Linux: ¿cambiamos el sistema que teníamos o lo hacemos por etapas coexistiendo ambos?

- En el apartado de herramientas estudiaremos (básicamente) aquel conjunto de útiles con el que el administrador tendrá que "vivir" (y/o sufrir) a diario, y que podrían formar la "caja de herramientas" del administrador. Hablaremos de los estándares GNU/Linux, que nos permitirán conocer aspectos comunes a todas las distribuciones GNU/Linux, es decir, lo que esperamos poder encontrar en cualquier sistema. Otra herramienta básica serán: los editores simples (o no tan simples); algunos comandos básicos para conocer el estado del sistema u obtener información filtrada según nos interese; la programación de guiones de comandos (o *shell scripts*) que nos permitirán automatizar tareas; características de los lenguajes que podemos encontrarnos en las aplicaciones o en herramientas de administración; procesos básicos de compilación de programas a partir de los códigos fuente; herramientas de gestión del software instalado, al mismo tiempo

que comentaremos la disyuntiva de uso de herramientas gráficas o las de línea de comandos.

- En el apartado dedicado al *kernel*, observamos el *kernel* Linux, y cómo, mediante el proceso de personalización, podemos adaptarlo mejor al hardware o a los servicios que queramos proporcionar desde nuestro sistema.

- En el apartado dedicado a administración local, trataremos aquellos aspectos de administración que podríamos considerar "locales" a nuestro sistema. Estos aspectos pueden conformar la mayor parte de las tareas típicas del administrador a la hora de manejar elementos tales como usuarios, impresoras, discos, software, procesos, etc.

- En el apartado dedicado a red, examinaremos todas aquellas tareas de administración que engloben nuestro sistema con su "vecindario" en la red, sea cual sea su tipo, así como ver los diferentes tipos de conectividad que podemos tener con los sistemas vecinos, o los servicios que les podemos ofrecer o recibir de ellos.

- En el apartado dedicado a servidores, veremos algunas configuraciones típicas de los servidores habituales, que podemos encontrar en un sistema GNU/Linux.

- En el apartado dedicado a datos, nos dedicaremos a uno de los aspectos más importantes hoy en día, a los mecanismos de almacenamiento y consulta de datos que nos pueden ofrecer los sistemas GNU/Linux, en concreto, a los sistemas de base de datos, y los mecanismos de control de versiones.

- En el apartado dedicado a seguridad, trataremos uno de los puntos más importantes de todo sistema GNU/Linux de hoy en día. La existencia de un "mundo" interconectado por Internet nos trae una serie de "peligros" importantes para el correcto funcionamiento de nuestros sistemas y plantea el tema de la fiabilidad, tanto de estos sistemas, como de los datos que podamos recibir o proporcionar por la red. Con ello, tenemos que asegurar unos niveles mínimos de seguridad en nuestros sistemas, así como controlar y evitar los accesos indebidos o las manipulaciones de nuestros datos. Veremos qué tipos de ataques son los más frecuentes, qué políticas de seguridad cabría seguir y qué herramientas nos pueden ayudar para controlar nuestro nivel de seguridad.

- En el apartado dedicado a optimización, veremos cómo ,en los sistemas GNU/Linux, debido a la gran cantidad de servidores y servicios ofrecidos, así como los diferentes ambientes para los que está pensado el sistema, suelen existir muchos parámetros de funcionamiento que influyen en el rendimiento de las aplicaciones o de los servicios ofrecidos. Podemos (o

debemos) intentar extraer el máximo rendimiento, analizando las configuraciones del propio sistema y de los servidores, para adecuarlos a la calidad de servicio que queramos proporcionar a los clientes.

- En el apartado dedicado a *clustering*, veremos algunas de las técnicas para proporcionar computación de altas prestaciones en los sistemas GNU/Linux, muy utilizadas en áreas de computación científica, y cada vez más utilizadas por gran cantidad de industrias (farmacéutica, química, materiales, etc.), para el desarrollo y la investigación de nuevos productos. Así como la organización de varios sistema GNU/Linux en forma de *clusters*, para ampliar las prestaciones de los sistemas individuales, mediante la formación de grupos de sistemas que permitan escalar los servicios ofrecidos ante el aumento de demanda de los clientes.

Actividades

1. Leer el manifiesto Debian en:

http://www.debian.org/social_contract

2. Documentarse sobre las diferentes distribuciones basadas en Debian: Knoppix, Linex, variedades Ubuntu. Aparte de los sitios de cada distribución, en la dirección www.distrowatch.com hay una buena guía de las distribuciones y su estado, así como el software que incluyen. En esta web, o bien accediendo a las diversas comunidades o fabricantes pueden obtenerse las imágenes ISO de las diferentes distribuciones.

Otras fuentes de referencia e información

[LPD] The Linux Documentation Project (LDP), colección de Howto's, manuales y guías cubriendo cualquiera de los aspectos de GNU/Linux.

[OSDb] Comunidad de varios sitios web, noticias, desarrollos, proyectos, etc.

[Sla] Sitio de noticias de la comunidad Open Source, y generales de informática e Internet.

[New] [Bar] Noticias Open Source.

[Fre] [Sou] Listado de proyectos Open Source.

[Dis] Seguimiento de las distribuciones GNU/Linux y novedades de los paquetes software. Y enlaces a los sitios de descarga de las imágenes ISO de los CD/DVD de las distribuciones GNU/Linux.

[His] [Bul] [LPD] Documentación general y comunidades de usuarios.

[Mag03] [Jou03] Revistas GNU/Linux.

Migración y coexistencia con sistemas no Linux

Josep Jorba Esteve

P07/M2103/02281

UOC
www.uoc.edu

Índice

Introducción

Una vez realizada una primera aproximación a los sistemas GNU/Linux, el siguiente paso es integrarlos en el entorno de trabajo como sistemas de producción. Según el sistema actual que esté en uso, podemos plantear, o bien una migración total a sistemas GNU/Linux, o bien una coexistencia mediante servicios compatibles.

La migración al entorno GNU/Linux puede hacerse de manera progresiva, sustituyendo servicios parcialmente, o bien sustituyendo todo por los equivalentes GNU/Linux del antiguo sistema.

En los entornos distribuidos actuales, el paradigma más presente es el de cliente/servidor. Cualquier tarea en el sistema global está gestionada por uno o más servidores dedicados, accediendo las aplicaciones o directamente los usuarios a los servicios prestados.

Respecto al ambiente de trabajo, ya sea desde el caso más simple como el usuario individual, o bien uno complejo como un entorno empresarial, cada entorno necesitará un conjunto de servicios que tendremos que seleccionar, adaptando luego las máquinas clientes y servidores, para que puedan acceder a éstos o proporcionar su uso.

Los servicios pueden englobar muchos aspectos diferentes, suelen estar presentes varios tipos, ya sea para compartir recursos o información. Suelen ser habituales servidores de archivos, de impresión, de web, de nombres, correo, etc.

El administrador normalmente seleccionará un conjunto de servicios que deberán estar presentes en el ambiente de trabajo, dependiendo de las necesidades de los usuarios finales, y/o de la organización; y deberá configurar el soporte adecuado a la infraestructura, en forma de servidores que soporten la carga de trabajo esperada.

1. Sistemas informáticos: ambientes

En el proceso de algunas instalaciones de distribuciones de GNU/Linux, podemos encontrarnos a menudo que nos preguntan por el tipo de ambiente, o tareas a las que va a estar dedicado nuestro sistema, esto permite muchas veces escoger un subconjunto de software que se nos instalará por defecto, por ser el más adecuado a la función prevista. Suele ser habitual que nos pregunten si el sistema se destinará a:

a) Estación de trabajo (*workstation*): este tipo de sistema normalmente incorpora algunas aplicaciones particulares que serán las más usadas. El sistema básicamente se dedica a la ejecución de estas aplicaciones y a un pequeño conjunto de servicios de red.

b) Servidor: básicamente se integran la mayoría de servicios de red o, en todo caso, alguno particular, que va a ser el servicio principal del sistema.

c) Estación dedicada a cálculo: aplicaciones intensivas en cálculo, *renders*, aplicaciones científicas, gráficos CAD, etc.

d) Estación gráfica: escritorio con aplicaciones que necesitan de interactividad con el usuario en forma gráfica.

Normalmente podemos componer nuestro sistema GNU/Linux a partir de una o más de estas posibilidades.

Más en general, si tuviésemos que separar los ambientes de trabajo [Mor03] en que se puede utilizar un sistema GNU/Linux, podríamos identificar tres tipos principales de ambiente: estación de trabajo (*workstation*), servidor y escritorio (*desktop*).

También se podría incluir otro tipo de sistemas, los que llamaríamos genéricamente como dispositivos empotrados (*embebbed*), o bien sistemas móviles de pequeñas dimensiones, por ejemplo, un PDA, un teléfono móvil, una videoconsola portátil, etc. GNU/Linux ofrece, asimismo, soporte para estos dispositivos, con *kernels* reducidos y personalizados para ello.

Nota

Los sistemas GNU/Linux pueden dedicarse a funciones de servidor, estación de trabajo o escritorio.

Ejemplo

Destacamos, por ejemplo, el trabajo realizado por la firma Sharp en sus modelos Zaurus, un PDA con Linux de altas prestaciones (existen cuatro o cinco modelos en el mercado). O también otras iniciativas Linux de tipo empotrado como los TPV (terminales punto de venta). O videoconsolas como la GP2X.

Respecto a los tres principales ambientes, veamos cómo se desarrolla cada uno de estos sistemas informáticos en un entorno GNU/Linux:

1) Un sistema de tipo *workstation* suele ser una máquina de alto rendimiento, utilizada para una tarea específica en lugar de un conjunto general de tareas. La *workstation*, clásicamente, estaba compuesta de una máquina de altas prestaciones con hardware específico adecuado a la tarea que había que desarrollar; solía tratarse de una máquina Sun Sparc, IBM Risc o Silicon Graphics (entre otras) con sus variantes de UNIX propietarios. Estas máquinas de alto coste se orientaban a un segmento claro de aplicaciones, ya fuese el diseño gráfico 3D (caso Silicon o Sun) o bases de datos (IBM o Sun). Hoy en día, muchos de los actuales PC tienen un rendimiento comparable a estos sistemas (aunque no igual), y la frontera entre uno de estos sistemas y un PC no está ya tan clara, gracias a la existencia de GNU/Linux como alternativa a los versiones de UNIX propietarios.

2) Un sistema de tipo servidor tiene un objetivo concreto, que es ofrecer servicios a otras máquinas de la red: ofrece características o una funcionalidad clara al resto de máquinas. En sistemas informáticos pequeños (por ejemplo, menores de 10 máquinas), no suele haber un sistema exclusivo de servidor, y suele estar compartido con otras funcionalidades, por ejemplo también como máquina de tipo escritorio. En sistemas medianos (unas pocas decenas de máquinas), suele haber una o más máquinas dedicadas a actuar de servidor, ya sea la máquina exclusiva que centra todos los servicios (correo, web, etc.) o un par de máquinas dedicadas a repartirse los servicios principales.

En sistemas grandes (un centenar o más de máquinas, incluso miles), por la capacidad de carga puede ser necesario que exista un buen grupo de servidores, dedicados normalmente cada uno de ellos a algún servicio en exclusiva, o incluso dedicar un conjunto de máquinas a exclusivamente a un servicio. Es más, si estos servicios se proporcionan –hacia dentro o hacia fuera de la organización–, mediante acceso por clientes directos o abierto a Internet, dependiendo de la capacidad de carga que tengamos que soportar, tendremos que recurrir a soluciones de tipo SMP (máquinas con varios procesadores) o de tipo *cluster* (agrupación de máquinas que se distribuyen la carga de un determinado servicio).

Los servicios que podemos necesitar de forma interna (o externa), podrían englobarse (entre otras) dentro de estas categorías de servicios:

a) Aplicaciones: el servidor dispone de ejecución de aplicaciones y como clientes sólo observamos la ejecución de éstas e interactuamos con ellas. Puede, por ejemplo, englobar servicios de terminales y ejecución de aplicaciones en web.

b) Ficheros: se nos proporciona un espacio común y accesible desde cualquier punto de la red de donde almacenar/recuperar nuestros ficheros.

c) Base de datos: se centralizan datos que se van a consultar o producir por parte de las aplicaciones del sistema en red (o bien de otros servicios).

d) Impresión: se dispone de conjuntos de impresoras, donde se gestionan sus colas y los trabajos que se les envíen desde cualquier punto de la red.

e) Correo electrónico: se ofrecen servicios para recibir, enviar o reenviar correos procedentes o destinados tanto al interior como al exterior.

f) Web: servidor (o servidores) propios de la organización, de utilización interna o externa para los clientes.

g) Información de red: en organizaciones grandes es imprescindible poder localizar los servicios ofrecidos o recursos compartidos; o los mismos usuarios, si necesitan servicios que permitan esta localización y consulta de propiedades de cada tipo de objeto.

h) Servicios de nombres: se necesitan servicios que permitan nombrar y traducir los diversos nombres por los que se conoce a un mismo recurso.

i) Servicios de acceso remoto: en caso de no disponer de acceso directo, debemos disponer de métodos alternativos que nos permitan interaccionar desde el exterior, que nos permitan acceder al sistema que queramos.

j) Servicios de generación de nombres: en el nombramiento de máquinas, por ejemplo, puede darse una situación muy variable de número o que aquéllas no sean siempre las mismas. Debemos proporcionar métodos para identificarlas claramente.

k) Servicios de acceso a Internet: en muchas organizaciones no tiene por qué haber accesos directos, sino accesos por medio de pasarelas (*gateways*) o por intermediario (*proxys*).

l) Servicios de filtrado: medidas de seguridad para filtrar información incorrecta o que afecte a nuestra seguridad.

3) Un sistema de tipo *desktop* sería simplemente una máquina que se utiliza para las tareas informáticas rutinarias, de todos los días (por ejemplo, el PC que tenemos en casa o en la oficina).

Ejemplo

Por ejemplo, podríamos poner las siguientes tareas como comunes (se incluyen algunos de los programas GNU/Linux más utilizados):

- Tareas ofimáticas: disponer de software clásico de una *suite* ofimática: procesador de texto, hoja de cálculo, presentaciones, alguna pequeña base de datos, etc. Podemos encontrar *suites* como OpenOffice (gratuita), StarOffice (de pago, producida por Sun), KOffice (de KDE), o varios programas como Gnumeric, AbiWord que formarían una *suite* para Gnome (denominada GnomeOffice).

Nota

http://www.gnome.org/gnomeoffice

- Navegación web: navegadores como Mozilla, Konqueror, Galeon, etc.

- Soporte hardware (dispositivos USB, de almacenamiento...). En GNU/Linux soportados por los controladores adecuados, normalmente proporcionados en el *kernel*, o bien por fabricantes. También hay herramientas de análisis de hardware nuevo, como kudzu (Fedora/Red Hat) o discover (Debian). Media y entretenimiento (gráficos, procesamiento imágenes, fotografía digital, juegos y más). En GNU/Linux hay una cantidad enorme de estas aplicaciones, de calidad muy profesional: Gimp (retoque fotográfico), Sodipodi, Xine, Mplayer, gphoto, etc.

- Conectividad (acceso al escritorio de forma remota, acceso a otros sistemas). A este respecto, en GNU/Linux hay una cantidad enorme de herramientas, ya sea las propias TCP/IP como ftp, telnet, web, etc., como X Window, que tiene capacidades de escritorio remoto hacia cualquier máquina UNIX, rdesktop (para conectarse a escritorios Windows 2000/XP), o VNC (que permite conectarse a UNIX, Windows, Mac, etc.).

2. Servicios en GNU/Linux

GNU/Linux dispone de servidores adaptados para cualquier ambiente de trabajo.

Las categorías de los servicios que hemos comentado tienen equivalentes en servicios que podemos proporcionar desde nuestros sistemas GNU/Linux al resto de máquinas de la red (y de los que también podremos actuar como cliente):

a) Aplicaciones: GNU/Linux puede proporcionar servicios de terminales remotos, ya sea por conexión directa mediante interfaces serie de terminales "tontos", que sirvan para visualizar o interactuar con las aplicaciones. Otra posibilidad es la conexión remota de modo textual, desde otra máquina, por medio de servicios TCP/IP como los rlogin, telnet, o de forma segura con ssh. GNU/Linux proporciona servidores para todos estos protocolos. En el caso de ejecutar aplicaciones gráficas, disponemos de soluciones mediante X Window de forma remota, cualquier cliente UNIX, Linux o Windows (u otros) que dispongan de un cliente X Window puede visualizar la ejecución del entorno y sus aplicaciones. Asimismo, hay otras soluciones como VNC para el mismo problema. En cuanto al tema de aplicaciones vía web, GNU/Linux dispone del servidor Apache, y cualquiera de los múltiples sistemas de ejecución web están disponibles, ya sean Servlets (con Tomcat), JSP, Perl, PHP, xml, *webservices*, etc., así como servidores de aplicaciones web como BEA Weblogic, IBM Websphere, JBoss (gratuito) que también se ejecutan sobre plataformas GNU/Linux.

b) Ficheros: pueden servirse ficheros de múltiples maneras, desde el acceso por ftp a los ficheros, como servirlos de forma transparente a otras máquinas UNIX y Linux con NFS, o bien actuar de cliente o servidor hacia máquinas Windows mediante Samba.

c) Base de datos: soporta una gran cantidad de bases de datos cliente/servidor de tipo relacional como MySQL, PostgreSQL y varias comerciales como Oracle o IBM DB2 entre otras.

d) Impresión: puede servir impresoras locales o remotas, tanto a sistemas UNIX con protocolos TCP/IP, como a Windows mediante Samba/CIFS.

e) Correo electrónico: ofrece tanto servicios para que los clientes obtengan correo en sus máquinas (servidores POP3 o IMAP), como agentes MTA (*mail transfer agent*) para recuperar y retransmitir correo, como el servidor Sendmail (el estándar UNIX) u otros como Exim, y en el caso de envíos externos, el servicio de SMTP para el envío de correo externo.

f) Web: disponemos del servidor http Apache, ya sea en sus versiones 1.3.x o las nuevas 2.0.x. Además, podemos integrar servidores de aplicaciones web, como Tomcat para servir servlets, JSP...

g) Información de red: servicios como NIS, NIS+ o LDAP nos permiten centralizar la información de las máquinas, usuarios y recursos varios de nuestra red, facilitando la administración y los servicios a los usuarios, de manera que éstos no dependan de su situación en la red. O si nuestra organización tiene cierta estructura interna, estos servicios nos permiten modelarla dejando acceso a los recursos a quien los necesita.

h) Servicios de nombres: servicios como DNS para los nombres de las máquinas y su traducción desde IP o a IP, por medio de, por ejemplo, el servidor Bind (el DNS estándar UNIX).

i) Servicios de acceso remoto: ya sea para ejecutar aplicaciones o para obtener información remota de las máquinas. Los servidores podrían ser los que hemos comentado para aplicaciones: X Window, VNC, etc., y también los que permiten ejecutar algunos comandos remotos sin interactividad como rexec, rsh, ssh, etc.

j) Servicios de generación de nombres: servicios como DHCP permiten redes TCP/IP, una generación dinámica (o estática) de las direcciones IP que se disponen en función de las máquinas que las necesiten.

k) Servicios de acceso a Internet: en determinadas situaciones puede tenerse un único punto de salida a Internet (o varios). Estos puntos suelen actuar como *proxy*, ya que tienen el acceso y lo redirigen a los posibles accesos a Internet por parte de los clientes. También suelen actuar de caché de contenidos. En GNU/Linux podemos disponer, por ejemplo, del Squid. Dentro de esta categoría, también podría entrar la actuación de un sistema GNU/Linux de pasarela (*gateway*) o de *router*, ya sea para dirigir paquetes hacia otras redes o para buscar rutas de reenvío alternativas. También, en el caso de pequeñas instalaciones como las domesticas, podríamos incluir el acceso a Internet mediante módem por los servicios PPP.

l) Servicios de filtrado: una de las medidas de seguridad más utilizadas actualmente es la implantación de cortafuegos (o *firewalls*). Esto supone básicamente técnicas de filtrado de los paquetes entrantes o salientes, de los diferentes protocolos que estemos usando, para poner barreras a los no deseados. En GNU/Linux disponemos de mecanismos como ipchains e iptables (más moderno) para implementar los cortafuegos.

3. Tipologías de uso

GNU/Linux ofrece, como sistema, características válidas para el uso desde el usuario personal hasta el usuario de una infraestructura de media o gran escala.

Desde la perspectiva de los usuarios de los sistemas GNU/Linux, podríamos diferenciar:

a) El usuario individual o usuario doméstico: normalmente, este tipo de usuario dispone de una o varias máquinas en su hogar, que serán compartidas o no. En general, en este ambiente, GNU/Linux se usaría para desarrollar un sistema de escritorio, con lo cual, será importante la parte gráfica: el escritorio de GNU/Linux.

Para este escritorio tenemos dos opciones principales, en forma de los entornos Gnome y KDE, los dos entornos constituyen opciones perfectamente válidas. Cualquiera de los dos entornos dispone de servicios de visualización y ejecución de las aplicaciones, así como de un amplio conjunto de aplicaciones propias básicas que nos permiten desarrollar todo tipo de tareas rutinarias. Los dos entornos ofrecen un escritorio visual con diferentes menús, barras de utilidad e iconos, así como navegadores de ficheros propios y aplicaciones de utilidad variadas. Cada entorno puede ejecutar sus aplicaciones y las del otro, aunque, del mismo modo que las aplicaciones, tienen mejor ejecución en su entorno propio por tener un aspecto visual más acorde al entorno para el que se diseñaron.

En cuanto a las aplicaciones para el usuario personal, incluiríamos las típicas del sistema de escritorio. En el caso de que el usuario disponga de una red en su casa, por ejemplo, un pequeño conjunto de ordenadores mediante una red de tipo ethernet, también podrían ser interesantes los servicios para compartir ficheros e impresoras entre las máquinas. Podrían ser necesarios servicios como NFS si hay otras máquinas Linux; o bien Samba, si hay máquinas con Windows.

En el caso de tener una conexión a Internet por algún proveedor de acceso (ISP), según la forma de conexión utilizada, necesitaríamos controlar los dispositivos y los protocolos correspondientes:

- Conexión por módem: los módems telefónicos suelen utilizar el protocolo PPP de conexión con el proveedor. Tendríamos que habilitar este protocolo y configurar las cuentas que tengamos habilitadas en el proveedor. Un problema importante con Linux es el tema de los winModems, que han traído bastantes problemas. Estos módem (con excepciones) no están so-

portados, ya que no es un módem real, sino una simplificación hardware más un software de *driver*, y la mayoría funcionan únicamente con Windows, por lo que hay que evitarlos (si no están soportados) y comprar módem "reales" (completos).

- Conexión mediante un módem ADSL: el funcionamiento sería parecido, se podría utilizar el protocolo PPP u otro denominado EoPPP. Esto puede depender del fabricante del módem y del tipo de módem: Ethernet o USB.

- Conexión por ADSL con *router*: la configuración es muy simple, debido a que en esta situación sólo hay que configurar la tarjeta de red Ethernet que acompaña al *router* ADSL.

Una vez la interfaz a Internet está conectada y configurada, el último punto es incluir el tipo de servicios que necesitaremos. Si sólo queremos actuar como clientes en Internet, bastará con utilizar las herramientas cliente de los diferentes protocolos, ya sea ftp, telnet, el navegador web, el lector de correo o *news*, etc. Si además queremos ofrecer servicios hacia fuera –por ejemplo, publicar una web (servidor web) o permitir nuestro acceso externo a la máquina (servicios de ssh, telnet, ftp, X Window, VNC, etc.), en este caso, servidor– entonces, cabe recordar que esto será posible solamente si nuestro proveedor nos ofrece direcciones IP fijas para nuestra máquina. De otro modo, nuestra dirección IP cambiaría a cada conexión y la posibilidad de proporcionar un servicio se volvería muy difícil o imposible.

Otro servicio interesante sería compartir el acceso a Internet entre las máquinas de que dispongamos.

b) Usuario de media escala: es un usuario de una organización de media escala, ya sea una pequeña empresa o un grupo de usuarios. Normalmente, este tipo de usuarios dispondrán de conectividad en red local (por ejemplo, una LAN) con algunas máquinas e impresoras conectadas. Y tendrá acceso directo a Internet, bien a través de algún *proxy* (punto o máquina destinada a la conexión externa), o bien habrá unas pocas máquinas conectadas físicamente a Internet. En general, en este ambiente, el trabajo suele ser en parte local y en parte compartido (ya sea recursos como las impresoras o aplicaciones comunes). Normalmente, necesitaremos sistemas de escritorio, por ejemplo, en una oficina podemos utilizar las aplicaciones ofimáticas junto con clientes Internet; y quizás también sistemas de tipo *workstation*; por ejemplo, en trabajos de ingeniería o científicos pueden utilizarse aplicaciones de CAD, de procesamiento de imágenes, aplicaciones de cálculo matemático intensivo, etc., y seguramente habrá algunas máquinas más potentes destinadas a estas tareas.

En este ambiente de uso, por lo común necesitaremos servicios de compartición de recursos como ficheros, impresoras, posiblemente aplicaciones, etc. Por lo tanto, en un sistema GNU/Linux serán adecuados los servicios de NFS,

servicios de impresión, Samba (si hay máquinas Windows con las que compartir ficheros o impresoras), así como también es posible que tengamos necesidad de entornos de bases de datos, algún servidor interno de web con aplicaciones compartidas, etc.

c) Usuario de organización amplia: este tipo de usuario es bastante parecido al anterior, y sólo se diferencia en el tamaño de la organización y en los recursos de los que puede disponer, que podrían llegar a ser muy altos, de modo que se necesitarían algunos recursos de sistemas de directorio de red de tipo NIS, NIS+ o LDAP para poder manejar la información de la organización y reflejar su estructura, así como, seguramente, disponer de grandes infraestructuras de servicios hacia los clientes externos, por lo general en forma de sitios web con aplicaciones diversas.

En este tipo de organizaciones se presentan niveles de heterogeneidad elevados, tanto en el hardware como en el software de los sistemas, y podríamos encontrar muchas arquitecturas y diferentes sistemas operativos, por lo que la tarea principal va a consistir en facilitar la compatibilidad de los datos vía bases de datos y formatos de documentos estándar y facilitar la interconectividad mediante protocolos, clientes y servidores estándar (normalmente con elementos TCP/IP).

4. Migrar o coexistir

A continuación vamos a plantear otro aspecto importante en el proceso de adopción de los sistemas GNU/Linux. Supongamos que somos principiantes en el manejo de este sistema; o, por el contrario, que somos experimentados y queremos adoptar uno o varios sistemas GNU/Linux como usuarios individuales, para el trabajo en nuestra pequeña organización, o nos estamos planteando sustituir la infraestructura completa (o parcial) de nuestra gran empresa u organización.

La migración a un nuevo sistema no es trivial, hay que evaluarla mediante un estudio, donde se analicen tanto los costes, como las prestaciones que esperamos obtener. Además, puede realizarse total o parcialmente, con cierto grado de coexistencia con los antiguos sistemas.

Estaremos ante un proyecto de migración, total o parcial, de nuestros sistemas informáticos hacia GNU/Linux, y como administradores, seremos responsables de este proceso.

Como en todo proyecto, habrá que estudiar el modo de responder a cuestiones como: ¿es rentable el cambio, en prestaciones, en coste?, ¿con qué objetivo lo hacemos?, ¿qué requisitos queremos o debemos cumplir?, ¿podemos hacer, o es necesario hacer, una migración completa?, ¿tiene que haber coexistencia con otros sistemas?, ¿habrá que formar de nuevo a los usuarios?, ¿podremos utilizar el mismo hardware, o necesitaremos uno nuevo?, ¿habrá costes añadidos importantes?, etc., o simplemente, ¿saldrá bien?. Ésta y muchas preguntas más son las que tendremos que intentar responder. En el caso empresarial, la respuesta normalmente pasaría por la definición de un proyecto de migración, con sus objetivos, análisis de requisitos, proceso de implantación, estudios económicos, planes de formación de usuarios, etc. No entraremos en esto, pero nos plantearemos algunas de las cuestiones de forma sencilla. Y en el taller final examinaremos unos pequeños casos prácticos de cómo haríamos la migración.

Además, en el momento en que empecemos la migración a los sistemas GNU/Linux, es cuando comenzaremos a notar las ventajas que éstos aportarán a nuestra organización:

a) Costes: reducción de los costes, en licencias software del sistema y de las aplicaciones. GNU/Linux tiene un coste 0 en cuanto a las licencias, si se obtiene desde la red (por ejemplo, en forma de imágenes de los CD de la distribución), o un coste despreciable teniendo en cuenta que la mejor comparación para sistemas equivalentes en prestaciones serían sistemas

Windows Server, con costes que van de 1.500 a 3.000 euros por licencias, sin incluir gran parte del software extra que proporciona una distribución GNU/Linux típica.

Pero cuidado, no hay que desestimar los costes de mantenimiento y formación. Si nuestra organización sólo está formada por usuarios y administradores Windows, podemos tener costes altos en nueva formación, personal, y quizás en mantenimiento. Por eso, muchas grandes empresas desean depender de algún distribuidor comercial de GNU/Linux para que les implante y mantenga el sistema, como por ejemplo las versiones empresariales que ofrecen Red Hat, SuSe y otros. Estas versiones GNU/Linux también tienen altos costes de licencia (comparables a Windows), pero por el contrario, están ya adaptadas a estructuras empresariales y traen software propio para gestionar la infraestructura informática de las empresas. Otro aspecto importante, que resumiría esta estimación de costes, es el concepto de TCO (*total cost of ownership*), como evaluación global de los costes asociados que nos encontraremos al emprender un desarrollo tecnológico; no sólo hay que evaluar los costes de licencias y maquinaria, sino también los costes de soporte y formación de las personas y productos implicados, los cuales pueden ser tan importantes o más que los de la solución implementada.

b) Soporte: GNU/Linux tiene el soporte de mantenimiento mayor que haya tenido un sistema operativo y gratis en su mayor parte. A pesar de ello, algunas empresas no adoptan GNU/Linux por ciertos temores, diciendo que no hay soporte del producto, y se dedican a comprar distribuciones comerciales que les ofrecen contratos de soporte y mantenimiento. GNU/Linux tiene una comunidad de soporte mundial bien establecida, por medio de diferentes organizaciones que proporcionan documentación libre (los famosos HOWTO's), foros de usuarios especializados, comunidades de usuarios de prácticamente cualquier región o país del mundo, etc. Cualquier duda o problema con el que nos encontremos puede buscarse (por ejemplo, por alguno de los buscadores en Internet), y podemos tener respuestas en minutos. Cuando no, si hemos encontrado un bug, error, o situación no probada, podemos informar de ella en varios lugares (foros, sitios de desarrollo, sitios de bugs de distribuciones, etc.), y obtener soluciones en horas o a lo sumo algunos días. Siempre que aparezca una duda o algún problema, intentar primero algunos procedimientos (así se aprende), y si no obtenemos solución en un tiempo prudencial, consultar a la comunidad GNU/Linux por si a algún otro usuario (o grupo de ellos) le ha ocurrido el mismo problema y ha obtenido solución, y si no, siempre podemos informar del problema y que nos ofrezcan algunas soluciones.

Nota

Linux Howto's: http://www.tldp.org/

4.1. Identificar requisitos de servicios

Normalmente, si tenemos unos sistemas ya funcionando, tendremos que tener implantados algunos servicios, de los cuales serán clientes los usuarios, o

servicios que ayuden a la infraestructura del soporte informático. Los servicios entrarán dentro de alguna de las categorías vistas anteriormente, con las opciones GNU/Linux que comentamos.

Los sistemas GNU/Linux no son "nuevos" en absoluto, y derivan (como vimos en la introducción) de una historia de más de treinta años de uso y desarrollo de los sistemas UNIX. Por lo tanto, una de las primeras cosas que encontraremos es que no nos falta soporte para ningún tipo de servicio que queramos. Si acaso, habrá diferencias en la forma de hacer las cosas. Además, muchos de los servicios que se utilizan en los sistemas informáticos fueron pensados, investigados, desarrollados e implementados en su día para UNIX, y posteriormente adaptados a otros sistemas (como Windows, con más o menos acierto).

Muchas de las empresas que disponen de UNIX propietarios participan en GNU/Linux y ofrecen algunos de sus desarrollos a la comunidad.

Cualquier servicio disponible en el momento podrá ser adaptado en los sistemas GNU/Linux con servicios equivalentes (cuando no iguales).

Ejemplo

Un caso famoso es el de los servidores Samba [Woo00] [Sam]. Windows ofrece lo que él denomina "compartir archivos e impresoras en red" mediante unos protocolos propios denominados genéricamente SMB (*server message block*) [Smb] (con apoyo de red en los protocolos NetBios y NetBEUI). También es de común uso el nombre CIFS (*common Internet file system*), que es como se denominó al protocolo en una segunda revisión (que seguía incluyendo a SMB como protocolo base). Estos protocolos permiten compartir carpetas de archivos (o discos) y de impresoras en una red de máquinas Windows (en una configuración de *workgroup* o trabajo en grupo, o en dominios Windows). En UNIX esta idea ya era antigua, cuando apareció en Windows, y se disponía de servicios como NFS de compartición de archivos o la gestión remota de impresoras, bajo protocolos TCP/IP.

Uno de los problemas de sustituir los servicios Windows de compartición basados en NetBios/NetBeui (y últimamente con NetBios sobre TCP/IP), era cómo dar soporte a estos protocolos, ya que, si queríamos conservar las máquinas clientes con Windows, no podíamos utilizar los servicios UNIX. Para esto, Samba se desarrolló como un servidor para UNIX que soportaba los protocolos Windows, y podía sustituir a una máquina cliente/server Windows de forma transparente, los usuarios clientes con Windows no tenían por qué notar absolutamente nada. Es más, el resultado fue que en la mayor parte de los casos el rendimiento era comparable, cuando no era mejor que en la máquina original con los servicios Windows.

Nota

Un ejemplo de la evaluación de rendimiento Samba en: http://www.vnunet.com/News/1144289

Actualmente, Samba [Sam] evoluciona constantemente para mantener la compatibilidad de los servicios Windows de compartición de impresoras y archivos. Debido a los cambios generales a los que Microsoft somete a los protocolos SMB/CIFS [Smb] (base que Samba implementa) en cada nueva versión de Windows, en particular la evolución desde los esquemas de trabajo en grupo en sus versiones cliente del operativo, a los esquemas centralizados en servidor (o en grupos de ellos), con servicios particulares de autentificación de usuarios (NTLM, NTLMv2, Kerberos), y almacenamiento centralizado de la gestión del sistema como Active Directory. Además de esto, la configuración de servidores de dominios existentes (ya sean con servidores controladores primarios, *backup* o Active Directory).

Actualmente, en los procesos de migración con Samba, tendremos que observar qué configuraciones de clientes/servidores Windows (y las versiones de éste) existen en el

sistema informático, así como qué mecanismos de autentificación de los usuarios y/o gestión de la información se utilizan. Asimismo, necesitaremos conocer la estructuración del sistema informático en dominios (y sus servidores controladores, miembros o servidores aislados), para poder realizar un mapeado completo y correcto hacia soluciones basadas sobre Samba, y en servicios complementarios de autentificación de usuarios (winbind, kerberos, nss_ldap) y la gestión (como por ejemplo openLDAP) [Sama] [Samb] .

4.2. Proceso de migración

En el proceso de migración, hay que tener en cuenta qué se quiere migrar, y si quiere efectuarse de manera completa o parcial, coexistiendo con otros servicios o con equipos que disponen de un sistema operativo diferente.

En ambientes como el de las grandes organizaciones, donde encontramos un gran número de sistemas heterogéneos, habrá que tener en cuenta que seguramente no se migrarán todos, en especial los sistemas de tipo *workstation* dedicados a la ejecución de alguna aplicación básica para una tarea; puede que no exista la aplicación equivalente o simplemente podemos desear quedarnos con esos sistemas por razones de coste o de rentabilizar la inversión realizada.

Podemos migrar varios elementos, ya sean los servicios que ofrecemos, las máquinas que los sirven o los clientes que acceden a ellos.

Los elementos que se migren pueden ser:

a) Servicios o máquinas dedicadas a uno o más servicios. En la migración, pasaremos por la sustitución del servicio por otro equivalente, normalmente con el menor impacto posible si no queremos sustituir también a los clientes. En caso de clientes Windows, podemos usar el servidor Samba para sustituir los servicios de archivos e impresoras que proporcionaban las máquinas Windows. Si se trata de otros servicios, podremos sustituirlos por los equivalentes GNU/Linux. En el caso de sustituir sólo algún servicio, normalmente se inhabilitará el servicio en la máquina que lo ofrecía y se habilitará en el sistema nuevo. Pueden ser necesarios cambios en los clientes (por ejemplo, direcciones de la nueva máquina o parámetros relacionados con el servicio).

Si la función la cumplía por entero una máquina servidora, hay que analizar si la máquina estaba dedicada a uno o más servicios y si todos podrán ser sustituidos. En tal caso, sólo hay que reemplazar la máquina antigua por la nueva (o mantener la antigua) con los servicios bajo GNU/Linux, y en todo caso, modificar algún parámetro en los clientes si fuese necesario. Normalmente, antes de efectuar el cambio, es conveniente testear la máquina por separado con algunos clientes para asegurarse de que cumple su función correctamente y sustituir las máquinas en algún periodo de inactividad del sistema.

En cualquier caso, seguramente habrá que hacer *backups* de los datos anteriores al nuevo sistema, por ejemplo, el sistema de ficheros o las aplicaciones disponibles en el servidor original. Otro de los puntos previos a tener en cuenta es la portabilidad de los datos; un problema que a menudo presenta difícil solución si en la organización se utilizaban formatos de datos o aplicaciones dependientes de una plataforma.

Ejemplo

Por poner algunos casos prácticos de problemas con que se encuentran algunas empresas hoy en día:

- Aplicaciones en web con ASP: estas aplicaciones son sólo realizables en plataformas web con Windows y el servidor web IIS de Microsoft. Habría que evitarlas, si en algún momento pensamos hacer una migración de plataformas, y no queremos reescribirlas o pagar a una empresa para que lo haga. En plataformas GNU/ Linux, está disponible el servidor web Apache (el más utilizado en Internet), que también se puede utilizar con Windows, este servidor soporta ASP en Perl (en Windows se suele utilizar visual basic, C# y Javascript generalmente), hay soluciones de terceros para migrar los ASP o más o menos convertirlos. Pero si nuestra empresa dependiese de esto, sería muy costoso en tiempo y dinero. Una solución práctica habría sido realizar los desarrollos web en Java (que sí que es portable entre plataformas) u otras soluciones como PHP. En este punto cabe destacar el proyecto Mono [Mon] (patrocinado por Novell) para la portabilidad de parte del entorno .NET de Microsoft a GNU/Linux, en particular gran parte de las API de .NET, el lenguaje C#, y la especificación ASP.NET. Permitiendo una migración flexible de aplicaciones .NET basadas en APIs .NET que esten soportadas por la plataforma Mono. Por otra parte cabe señalar el proyecto DotGnu [Dgn] de la FSF, como alternativa GPL a Mono.

- Bases de datos: utilizar, por ejemplo, un SQL *Server* de Microsoft, nos hace totalmente dependientes de su plataforma Windows, además, si utilizamos soluciones propietarias en un entorno concreto para aplicaciones de la base de datos, serán de difícil transferencia. Otras bases de datos como Oracle y DB2 (de IBM) son más portables por disponer de versión en las diferentes plataformas, o por utilizar lenguajes de programación más portables. También se podría trabajar con sistemas de bases de datos PostgreSQL o MySQL (también tiene versión para Windows) disponibles en GNU/Linux, y que permiten una transición más fácil. Asimismo, si se combina con el desarrollo web tenemos muchas facilidades; en este sentido, hoy en día se utilizan sistemas como: aplicaciones web con Java, ya sea *servlets*, *applets*, o EJB; o bien soluciones como las famosas LAMP, combinación de GNU/Linux, Apache, Mysql y Php.

b) Workstation: en estas migraciones, el mayor problema parte de las aplicaciones, ya que son las que dan su razón de ser a la estación de trabajo, ya sean programas de CAD, de animación, de ingeniería o científicos. Aquí será importante que podamos sustituirlas por aplicaciones iguales o, como mínimo, compatibles con las mismas características o funcionalidad esperada. Normalmente, la mayor parte de estas aplicaciones ya provienen de un mundo UNIX, puesto que la mayoría de estas *workstations* estaban pensadas como máquinas UNIX. Con lo cual, quizás baste una recopilación o una adaptación mínima al nuevo sistema GNU/Linux, si disponemos del código fuente (como suele pasar en muchas aplicaciones científicas). Si se trata de aplicaciones comerciales, los fabricantes (de software de ingeniería y científico) comienzan a adaptarlas a GNU/Linux, aunque en estos casos las aplicaciones suelen ser muy caras (pueden ir perfectamente de miles a centenares de miles de euros).

c) Máquinas clientes de escritorio. Las máquinas de escritorio continúan siendo un quebradero de cabeza en el mundo GNU/Linux, ya que ofrecen bastantes problemas adicionales. En los servidores, las máquinas se destinan a

funcionalidades claras, en general no requieren interfaces gráficas complejas (muchas veces con comunicación textual es suficiente), el hardware, normalmente *expreso* y de altas prestaciones, se compra para unas funcionalidades concretas y las aplicaciones suelen ser los propios servidores incluidos en el sistema operativo o algunos de terceros. Además, estas máquinas suelen estar gestionadas por personal de tipo administrador que tiene amplios conocimientos de lo que maneja. Por contra, en el caso del escritorio, nos encontramos con un factor problemático (en sí mismo, y aún más para los administradores): los usuarios finales del sistema. Los usuarios de escritorio esperan disponer de potentes interfaces gráficas, más o menos intuitivas, y aplicaciones que permitan desarrollar sus tareas rutinarias, normalmente ofimáticas. Este tipo de usuario (con excepciones) no tiene por qué tener unos conocimientos informáticos elevados; en general, sus conocimientos son de tipo ofimático y suelen usar un reducido número de aplicaciones con mayor o menor dominio de las mismas. Aquí GNU/Linux tiene un problema claro, ya que UNIX como tal nunca fue pensado como un sistema puramente de escritorio, y sólo fue adaptado *a posteriori* por sistemas gráficos como X Window y los diferentes escritorios, como los actuales de GNU/Linux: Gnome y KDE. Además, el usuario final suele estar acostumbrado a sistemas Windows (que copan casi un 95% del mercado de escritorio).

En el caso del escritorio, GNU/Linux tiene que superar unos cuantos obstáculos. Uno de los más críticos es que no viene preinstalado en las máquinas, lo que obliga al usuario a tener conocimientos para poder instalarlo. Otros motivos podrían ser:

Nota

El ambiente de escritorio es una batalla todavía por librar para los sistemas GNU/Linux; tienen que vencer la desconfianza de los usuarios a cambiar de sistema, y saber dar a conocer que ofrecen alternativas, de sencillez y aplicaciones, que solucionan las tareas de los usuarios.

- Desconfianza del usuario: una pregunta que se puede plantear un usuario es, ¿por qué debo cambiar de sistema? o ¿me ofrecerá lo mismo el nuevo entorno? Una de las razones básicas para hacer el cambio sería el software de calidad y su precio, del que una buena parte es libre. En este punto, cabe tener en cuenta el tema de las copias de software ilegales. Parece ser que los usuarios consideran que su software es gratis, cuando en realidad están en una situación ilegal. El software GNU/Linux ofrece gran calidad a bajo coste (o gratis en muchos casos), y existen múltiples alternativas para una misma tarea.

- Sencillez: el usuario se muestra normalmente perdido si el sistema no le ofrece algunas referencias que lo hagan parecido a lo que ya conoce, como el comportamiento de la interfaz, o que las herramientas sean parecidas en funcionalidad. Espera que, en general, no necesite mucho tiempo extra para aprender y manejar el nuevo sistema. GNU/Linux aún presenta algunos problemas en las instalaciones más o menos automáticas que, aunque

mejoran día a día, todavía es necesario un cierto grado de conocimiento para hacer una instalación correcta del mismo. En este punto, cabe destacar la facilidad de instalación en diferentes ambientes ofrecida por distribuciones recientes como Ubuntu [Ubu]. Otro problema habitual radica en el soporte del hardware del PC, que a pesar de que cada día mejora, los fabricantes actualmente no le prestan la atención adecuada (en parte por la cuota de mercado). Hasta que no haya una clara intención en este aspecto, no podremos tener el mismo soporte que en otros sistemas propietarios (como en Windows). Sin embargo, cabe destacar el trabajo de la comunidad del *kernel* de Linux para dar el soporte adecuado a nuevas tecnologías, en algunos casos dando soporte al fabricante, o preparando soporte primario (si no esta soportado por el fabricante) o alternativo al ofrecido por el fabricante.

- Transparencia: los entornos GNU/Linux tienen muchos mecanismos complejos, como los *daemons*, servicios, ficheros ASCII difíciles de configurar, etc. De cara a un usuario final, sería necesario poder ocultar todas estas complejidades, mediante programas gráficos, asistentes de configuración, etc. Es uno de los caminos que han tomado algunas distribuciones como Red Hat, Mandriva, Ubuntu o SuSe.

- Soporte de aplicaciones conocidas: un usuario ofimático típico tendrá el problema de la portabilidad de sus datos, o el tratamiento de los formatos de éstos. ¿Qué hace con los datos que tenía hasta el momento? Este problema está siendo mejorado día a día, gracias a las *suites* ofimáticas que comienzan a tener las funcionalidades necesarias para un usuario de escritorio. Por ejemplo, si nos planteamos una migración desde un uso de una *suite* Office de Windows, podemos encontrar *suites* como OpenOffice (software libre y gratuito) que puede leer (y crear) los formatos (con algunas restricciones) de ficheros Office. La compatibilidad de formatos no es que sea difícil, cuando éstos son abiertos, pero en el caso de Windows Microsoft continúa manteniendo una política de formatos cerrados; y hay que hacer un trabajo importante para poder utilizar estos formatos, mediante reingeniería inversa (proceso bastante costoso). Además, en la era de Internet, donde la información se supone que se mueve libremente, los formatos cerrados sin documentar son más un obstáculo que otra cosa. Lo mejor es utilizar formatos abiertos como RTF (aunque este caso también tiene algún problema, por las múltiples versiones que existen de él), o bien formatos basados en XML (OpenOffice genera sus documentos propios en XML), o PDF para la documentación de lectura. También hay que destacar los esfuerzos realizados recientemente por la comunidad OpenOffice para la creación del *standard open document* (usado por la *suite* a partir de las versiones 2.x). Que han permitido disponer de un formato libre como estándar ISO para la creación de documentos. Este hecho ha obligado a Microsoft, a abrir (parcialmente) su formato en las versiones a partir de Office 2007, incorporando los formatos OpenXML.

- Proporcionar alternativas válidas: el software que se deja de usar tiene que tener alternativas que cumplan el trabajo anterior en el otro sistema. En la mayoría de aplicaciones existe una o varias alternativas con funcionalidades parecidas, cuando no superiores. Pueden encontrarse por Internet diferentes listas de equivalencias (más o menos completas) de aplicaciones Windows con sus correspondientes GNU/Linux.

Nota

Por ejemplo: http://www.linuxrsp.ru/win-linsoft/table-eng.html

- Soporte de ejecución de otras aplicaciones de diferentes sistemas: en algunas condiciones es posible ejecutar aplicaciones de otros sistemas UNIX (de la misma arquitectura, por ejemplo, Intel x86), o bien de MS-DOS o Windows, mediante paquetes de compatibilidad o algún tipo de emuladores.

La mayor parte de estos problemas, que aparecen en las migraciones de escritorio, están superándose poco a poco y nos permitirán en el futuro disfrutar de una mayor cuota de usuarios GNU/Linux en el escritorio, y a medida que aumenten, disponer de mejores aplicaciones y que las empresas de software se dediquen a implementar versiones para GNU/Linux.

En el caso empresarial, esto puede superarse con una migración suave, primero de las etapas de servidores, y *workstations*, y después pasar por un proceso de formación amplia de los usuarios en los nuevos sistemas y aplicaciones, para, finalmente, integrarlos en su escritorio.

Un proceso que va a ayudar en gran medida es la introducción del software de código abierto en las fases educativas y en las administraciones publicas, como es el caso reciente de la Comunidad de Extremadura con su distribución GNU/Linux llamada Linex; o recientes medidas para llevar este software a la educación primaria, o las medidas de universidades mediante la ejecución de cursos y materias con estos sistemas.

5. Taller de migración: análisis de casos de estudio

En este taller vamos a intentar aplicar lo estudiado en esta unidad para analizar unos procesos de migración sencillos, y algún detalle de las técnicas necesarias (en el caso de técnicas de red, las veremos en las unidades dedicadas a administración de redes).

Nos plantearemos los siguientes casos de estudio:

- Migración individual de un usuario de escritorio Windows a un sistema GNU/Linux.
- Migración de una pequeña organización que dispone de sistemas Windows y algunos UNIX.
- Migración de un servidor Windows aislado (*standalone*) a un servidor Samba en GNU/ Linux.

a) Migración individual de un usuario de escritorio Windows a un sistema GNU/Linux

Un usuario se plantea la migración a GNU/Linux [Ray02b]. Normalmente, primero se pasará por un periodo de convivencia, de modo que el usuario dispondrá de los dos sistemas, y dedicará cada uno de ellos a una serie de tareas: continuará desarrollando tareas en Windows mientras aprende el nuevo sistema y encuentra software equivalente, o software nuevo que le permita hacer otras tareas para las que antes no disponía de software.

La migración es para un usuario personal quizás uno de los procesos más complejos; hay que ofrecerle alternativas a lo que ya utiliza comúnmente, de manera que la adaptación no entrañe muchas complejidades extra y pueda adaptarse progresivamente con facilidad al nuevo sistema.

Una primera posibilidad será hacer una instalación dual [Ban01] [Sko03b] del sistema original (un Windows 9x o un NT/XP por ejemplo), junto con el sistema GNU/Linux.

Nota

Linux Hardware Howto: http://www.tldp.org/HOWTO/HardwareHOWTO/index.html

Un primer paso en una determinada configuración de máquina consistirá en verificar que nuestro hardware sea compatible con Linux [Pri02], ya sea por medio de alguna lista de compatibilidad de hardware o verificándolo en el fabricante, por si fuera necesario adquirir nuevos componentes o configurar de alguna forma particular el existente. Si desconocemos nuestro hardware, podemos verificarlo o bien en Windows con el "administrador de dispositivos"

(en el panel de control), o algún software de reconocimiento de hardware. Por otra parte, un método recomendable es la utilización de distribuciones GNU/Linux de tipo LiveCD, que nos permitirán sin instalación física comprobar el funcionamiento de GNU/Linux en nuestro hardware, ya que el único requisito es la posibilidad de arranque del sistema desde CD/DVD (en algunos casos se necesitará modificar la configuración BIOS para permitir este arranque). Existen LiveCD como Knoppix [Knp] con gran soporte hardware para las comprobaciones, así como los de las mayoría de las distribuciones GNU/Linux que suelen ofrecer LiveCD para la comprobación inicial de funcionamiento (en algunos casos, por ejemplo Ubuntu [Ubn], la instalación completa puede realizarse desde el mismo LiveCD). En todo caso cabe señalar que la prueba con un LiveCD concreto no nos impide que puedan surgir dificultades en la instalación final, debido o bien a que el LiveCD no es de la misma distribución GNU/Linux que la que finalmente instalaremos, o las versiones del sistema y/o aplicaciones no serán las mismas.

En cuanto a la instalación física en disco, necesitaremos disponer o bien de espacio libre en disco no particionado, o bien, si estamos en particiones de tipo FAT/32, podemos liberar espacio con programas que permitan el reajuste del tamaño de las particiones, que permiten recortar la partición existente (un *backup* previo de los datos es evidentemente recomendable). En la actualidad la mayor parte de las distribuciones soportan diversos esquemas de particionado del disco y el recorte de particiones, aunque dependiendo de la distribución pueden surgir problemas. En caso de no disponer del espacio suficiente, o de tener particiones con sistemas de ficheros que presenten problemas (por ejemplo, NTFS en algunas distribuciones), habrá que plantearse comprar un nuevo disco duro complementario, que dedicaremos totalmente o en parte a GNU/Linux.

Una vez completada la revisión del hardware, tendremos que decidir la distribución del sistema GNU/Linux que usaremos (una posibilidad apuntada antes, es escoger un LiveCD que nos haya satisfecho, y pasar a la instalación de la distribución). Si el usuario es poco experimentado en GNU/Linux, o tiene conocimientos básicos de informática, mejor decidirse por alguna de las distribuciones más “amigables” de cara al usuario, como Fedora, Mandriva, SuSe, o similares (cabe destacar las facilidades de Ubuntu en este punto). Si tenemos más conocimientos o estamos tentados a experimentar, podemos probar una distribución Debian. En el caso de las comerciales, la distribución, la mayoría de veces, con un hardware compatible (en algunos casos como Red Hat y SuSe, versiones empresariales, las distribuidoras certifican el hardware que soportan), se instala perfectamente sin problemas, y se realizan configuraciones básicas que permiten utilizar ya el operativo. En el proceso tendremos que instalar el software, que normalmente vendrá definido por unos conjuntos de software orientados: a servidores, a aplicaciones concretas, o a aplicaciones de escritorio como las ofimáticas, aplicaciones de desarrollo (si nos interesa la programación), etc.

Una vez instalado el sistema, se plantea el tema de la compartición de datos [Gon00] [Kat01], ¿cómo compartimos datos entre los dos sistemas? o ¿hay posibilidad de compartir algunas aplicaciones? Para esto hay varias soluciones:

a) Método por "intermediario": consiste en compartir los datos, por ejemplo, mediante disquete. Para ello, lo mejor son las utilidades denominadas *mtools*, son utilidades que permiten acceder a disquetes con formato MS-DOS de forma transparente, y existen múltiples comandos de línea que funcionan de forma muy parecida a MS-DOS o Windows. Estos comandos se llaman exactamente como los comandos MS-DOS originales, sólo que con una "m" delante, por ejemplo: mcd, mcopy, mdir, mdel, mformat, mtype, etc.

b) Método directo: consiste en usar directamente los sistemas de ficheros de Windows. Como veremos en la unidad de administración local, GNU/Linux puede leer y escribir una gran cantidad de sistemas de ficheros, entre ellos el FAT, FAT32, y NTFS (sólo lectura en algunos casos, aunque la mayoría de distribuciones ya incorporan el driver ntfs-3g [Nt3] que permite la escritura). Se tiene que pasar por un proceso denominado "de montaje", que permite incorporar el sistema de ficheros de Windows a un punto del árbol de archivos de Linux; por ejemplo, podríamos montar nuestro disco Windows en /mnt/Windows y acceder desde este punto a sus carpetas y archivos, permitiendo escrituras y lecturas. Con los ficheros de texto ASCII, hay que tener en cuenta las conversiones, ya que UNIX y Windows los tratan de modo diferente: en UNIX, el final de línea tiene un sólo carácter, el avance de línea, ASCII 10, mientras que en Windows hay dos, un retorno y un avance de línea, caracteres ASCII 13 y 10 (como detalle curioso en MAC es el ASCII 13). Con lo cual, suele ser habitual que, al leer un fichero ASCII dos/windows, éste contenga caracteres "raros" al final de línea. Hay editores como emacs que los tratan de forma transparente y, en todo caso, hay utilidades GNU/Linux que permiten convertirlos de uno a otro formato (con utilidades como duconv, recode, dos2UNIX, UNIX2dos).

c) Uso de aplicaciones: existen algunas alternativas para poder ejecutar las aplicaciones (no todas) de MS-DOS y Windows. Para GNU/Linux hay emuladores de MS-DOS como Dosemu [Sun02] o DOsBox, y para Windows existe el software de ejecución Wine [Win]. Éste puede ejecutar diversas aplicaciones de Windows (por ejemplo, permite ejecutar algunas versiones de Office e Internet Explorer), pero se continúa mejorando constantemente. Si la ejecución de aplicaciones Windows es imprescindible, nos puede ayudar algún software comercial: los cuales dan soporte extra a Wine, existen por ejemplo, Win4Lin, CrossOver y en algún caso con soporte especial para juegos como Cedega. Otra posible solución es el uso de las máquinas virtuales; un ejemplo de software de amplio uso es VMware, que crea como máquina virtual un PC completo, simulado por software, en el cual se le puede instalar un gran numero diferente de sistemas operativos. VMware está disponible en versiones para Windows y para GNU/Linux, lo que permite tener un GNU/Linux instalado con un

Windows corriendo virtualmente sobre él, o un Windows con GNU/Linux en virtual. Existen también otras soluciones de máquina virtual libres como QEmu, Bochs. En otro segmento, las máquinas virtuales, o genéricamente la virtualización se usa orientada a la creación de servidores virtuales, con soluciones como como Vmware server, o los proyectos abiertos Xen, OpenVZ, Vserver; donde es posible hacer coexistir varias máquinas virtuales corriendo sobre un operativo (normalmente a través de modificaciones en el *kernel* que soporten esta virtualización), o incluso sobre el hardware directamente, con pequeñas capas de software.

Aparte de compartir la información (aplicaciones y/o datos), pueden buscarse aplicaciones GNU/Linux que sustituyan a las originales Windows a medida que el usuario vaya aprendiendo a utilizarlas, y observe que cumplen las funcionalidades esperadas.

Ejemplo

Un caso típico seria la *suite* ofimática que puede migrarse a OpenOffice, que tiene un alto grado de compatibilidad con los ficheros de Office y un funcionamiento bastante semejante, o bien KOffice (para el escritorio KDE), o GNumeric y AbiWord (para Gnome). O en el caso de procesamiento de imágenes, tomamos Gimp, con funcionalidades semejantes a Photoshop. Y multitud de reproductores multimedia: Xine, Mplayer (o también una versión del RealPlayer). En Internet se pueden encontrar varias listas de equivalencias de programas entre Windows y GNU/Linux.

b) Migración de una pequeña organización que dispone de sistemas Windows y algunos UNIX

Nota

Listas de equivalencias:
http://www.linuxrsp.ru/win-lin-soft/table-eng.html
http://www.linuxeq.com/

La migración en una organización (aunque sea pequeña) plantea muchas dificultades: tendremos diferentes ambientes de trabajo, hardware y software heterogéneo, y más de una vez, reticencias de los usuarios al cambio.

Consideremos ahora una organización que tenga máquinas Windows y algunas máquinas UNIX dedicadas a servicios, o a *workstations* y unos usuarios un poco "anárquicos". Por ejemplo, estudiemos la siguiente situación: la organización tiene una pequeña red local de máquinas Windows repartidas por los usuarios, como máquinas de igual a igual en un grupo de trabajo Windows (no hay dominios NT).

El grupo es variopinto: tenemos máquinas con Windows 98, ME, NT, XP, pero personalizadas por cada usuario con el software que necesita para su trabajo diario: ya sea Office, navegador, lector de correo, o entornos de desarrollo para los programadores de diferentes lenguajes (por ejemplo, C, C++, Java).

Se dispone de algunos recursos hardware extras, como varias impresoras conectadas a la red local (aceptan trabajos TCP/IP), que permiten utilizarse desde cualquier punto de la organización. Asimismo, existe una máquina compartida, con algunos recursos especiales, como *scanner*, grabadora de CD y directorios compartidos por red, donde los usuarios pueden dejar sus directorios con

sus ficheros para procesos de *backup* o para recuperar, por ejemplo, imágenes escaneadas.

También disponemos de varias *workstations*, en este caso Sun Microsystem Sparc, que ejecutan Solaris (UNIX, comercial de Sun). Estas estaciones están dedicadas a desarrollo y a algunas aplicaciones científicas y gráficas. Estas máquinas disponen de servicios de NFS para compartir archivos y NIS+ para manejar la información de los usuarios que se conectan a ellas y que puedan hacerlo desde cualquiera de éstas de manera transparente. Algunas de las máquinas incluyen servicios específicos; hay una destinada a servidor web de la organización y otra destinada a servidor de correo.

Se plantea la posibilidad de realizar una migración a GNU/Linux por intereses de desarrollo de software y por el interés particular de algunos usuarios de disponer de este sistema.

Además, se aprovechará la migración para intentar solucionar algunos problemas: de seguridad, algunos sistemas antiguos Windows no son la mejor forma de compartir archivos; se quiere restringir el uso de la impresora (el gasto en papel y el coste asociado es alto) a unas cuotas más razonables. Por otra parte, se quiere ofrecer cierta libertad a los usuarios, no se les obligará a cambiar de sistema, aunque se les hará la sugerencia. Y aprovecharemos para comprar hardware nuevo que complemente al existente, por ejemplo, si las estaciones de trabajo están faltas de espacio de disco, lo cual supone limitaciones de espacio para correo y cuentas de usuario.

Después de toda esta pequeña descripción de nuestra organización (en otros casos más complejos, podría llenar varias páginas o ser un documento entero de análisis de la situación presente, y propuestas futuras), nos comenzamos a plantear posibilidades para solucionar todo esto:

1) ¿Qué hacemos con las *workstations* actuales? El coste en mantenimiento y licencias de software es elevado. Tenemos que cubrir el mantenimiento de fallos en las estaciones, hardware caro (en este caso, discos SCSI) y ampliaciones de memoria también caras. El coste del sistema operativo y sus actualizaciones también es caro. En este caso, se nos ofrecen dos posibilidades (dependiendo del presupuesto de que dispongamos para el cambio):

a) Podemos reducir costes convirtiendo las máquinas a sistemas GNU/Linux. Estos sistemas son de arquitectura Sparc y existen distribuciones que soportan esta arquitectura. Podríamos sustituir los servicios por sus equivalentes GNU/Linux; la sustitución sería prácticamente directa, puesto que ya usamos un sistema UNIX.

b) Otra posibilidad sería la eliminación del hardware propietario de Sun y convertir las estaciones en PC potentes con GNU/Linux; esto simplifica su mantenimiento posterior, aunque con un alto coste inicial.

2) ¿Y con el software de las *workstations*? Si las aplicaciones son de desarrollo propio, puede ser suficiente volver a compilarlas o la adaptación simple al nuevo entorno. Si son comerciales, tendremos que ver si la empresa puede proporcionarlas en entornos GNU/Linux, o si podemos encontrar reemplazos con funcionalidad parecida. En el caso de los desarrolladores, sus entornos de lenguajes C, C++ y Java pueden portarse fácilmente; en caso de C y C++, se puede utilizar el compilador GNU *gcc*, y existen multitud de IDE para el desarrollo (KDevelop, Anjuta,...); en el caso de Java, se puede utilizar el *kit* de Sun en GNU/Linux y entornos varios de código abierto (Eclipse de IBM o Netbeans).

3) ¿Y con los usuarios? A aquellos que estén interesados en GNU/Linux les podemos instalar equipos duales con Windows y GNU/Linux para que comiencen a probar el sistema y, si les interesa, pasar finalmente a un único sistema GNU/Linux. Podemos encontrar dos tipos de usuarios: los puramente ofimáticos, que necesitarán básicamente la *suite*, navegador y correo; todo lo cual se les puede ofrecer con un escritorio GNU/Linux como Gnome o KDE y software como OpenOffice, navegador Mozilla/Firefox, y correo Mozilla Mail u Thunderbird (o cualquier otro Kmail, Evolution...). La equivalencia es más o menos directa, depende de las ganas que los usuarios tengan de probar y usar el nuevo software. Para los desarrolladores, el cambio puede ser más directo, ya que se les ofrecen muchos más entornos y herramientas flexibles; podrían pasarse completamente a sistemas GNU/Linux o trabajar directamente con las *workstations*.

4) ¿Y las impresoras? Puede establecerse alguna estación de trabajo como servidor de impresión (ya sea por colas TCP/IP o por servidor Samba), y controlar las impresiones mediante cuotas.

5) ¿La máquina compartida? El hardware compartido puede dejarse en la misma máquina o se puede controlar desde un sistema GNU/Linux. En cuanto al espacio de disco compartido, puede moverse a un servidor Samba que sustituya al actual.

6) ¿Ampliamos el espacio del disco? Dependerá del presupuesto. Podemos mejorar el control mediante un sistema de cuotas que reparta el espacio de una forma equitativa y ponga límites a la saturación.

c) Migración de un servidor Windows a un servidor Samba en GNU/Linux

El proceso básico necesario suele ser bastante más extenso, consultar la bibliografía para ver los pasos completos del mismo.

En este caso, el proceso básico necesario para efectuar una posible migración de un servidor Windows que comparte carpetas e impresora a un servidor Samba en un sistema GNU/Linux.

Gracias al software como Samba, la migración desde entornos Windows es muy flexible y rápida e incluso con mejoras de prestaciones o en rendimiento.

Supongamos una máquina que pertenece a un grupo de trabajo GRUPO, que comparte una impresora llamada PRINTER y que tiene una carpeta compartida DATOS que no es más que el disco D de la máquina. Varios clientes Windows acceden a la carpeta para lectura/escritura, dentro de una red local con direcciones IP 192.168.1.x, donde x será 1 para nuestro servidor Windows, y los clientes tienen otros valores (las redes 192.168.x.x se utilizan a menudo como direcciones para montar redes privadas internas).

En nuestro proceso vamos a construir un servidor Samba, que es el que, como vimos, nos permitirá la ejecución en GNU/Linux del protocolo SMB/CIFS (*server message block / common Internet file system*). Este protocolo permite la interacción del sistema de archivos y de las impresoras por medio de redes en múltiples sistemas operativos. Podemos montar carpetas pertenecientes a Windows en las máquinas GNU/Linux, o bien parte de los archivos de GNU/Linux en Windows, y lo mismo con las impresoras de uno u otro. El servidor está compuesto de dos *daemons* (procesos de sistema) llamados smbd y nmbd.

El proceso smbd gestiona las peticiones de los clientes hacia los archivos o impresoras compartidos. El nmbd gestiona el sistema de nombres de las máquinas y los recursos bajo el protocolo NetBIOS (creado por IBM). Este protocolo es independiente de la red que se usa (actualmente, Microsoft utiliza generalmente en NT/2000/XP Netbios sobre TCP/IP). El nmbd también proporciona servicios WINS, que es el servicio de asignación de nombres, que normalmente se ejecuta sobre Windows NT/Server si tenemos una colección de máquinas; es una especie de combinación de DNS y DHCP para entornos Windows. El proceso es un poco complejo, pero en resumen: cuando una máquina Windows arranca, o bien tiene una dirección IP estática, o bien dinámica a través de un servidor DHCP, y además puede que tenga un nombre NetBIOS (el nombre que el usuario asigna a la máquina: en identificación de red), entonces, el cliente WINS contacta con el servidor para informar de cuál es su IP; si una máquina de red pregunta posteriormente por el nombre NetBios, se contacta con el servidor WINS para obtener su dirección IP y se establecen las comunicaciones. El nmbd ejecuta este proceso sobre GNU/Linux.

Como cualquier otro servicio de red, no se debería ejecutar sin considerar qué riesgos puede suponer su activación y cómo podemos minimizarlos. Respecto a Samba, hay que tener presentes los temas de seguridad, puesto que estamos abriendo parte de nuestros archivos y las impresoras locales o de la red. Tendremos que verificar bien las restricciones de comunicación que ponemos para no dar acceso a usuarios o máquinas no deseadas. En este ejemplo básico, no vamos a comentar estos temas; en un caso real, ten-

dríamos que examinar las opciones de seguridad y restringir el acceso sólo a quien realmente deseemos.

En el proceso de migración, primero tendremos que configurar el sistema GNU/Linux para el soporte de Samba [Woo00], se necesita el soporte en el *kernel* de los *filesystems* Samba (*smbfs*), que normalmente ya viene activado. Hay que añadir que actualmente hay un soporte adicional en el *kernel* a través del modulo (cifs) [Ste07], el cual al partir de la versión del *kernel* 2.6.20, se considera el método por defecto, quedando smbfs en segundo término. El modulo cifs aporta soporte para nuevas prestaciones relacionadas con el protocolo CIFS (como extensión de SMB). Estos módulos nos permiten a través de los nombres de sistemas de ficheros "smbfs" y "cifs" realizar operaciones de montaje de sistemas de ficheros Windows en el árbol de directorios de Windows (mount -t smbfs o mount -t cifs). Aparte de que el soporte *kernel* se decanta hacia el modulo cifs, hay algunas características que pueden necesitar soporte smbfs, con lo cual suele ser habitual disponer de los dos módulos activados en el *kernel*. También hay que destacar la cuestión de la configuración, mientras smbfs basa su operación en la configuración Samba (como veremos en el fichero smb.conf), al modulo cifs se le proporciona la configuración en las operaciones (por ejemplo, en el proceso de montaje mediante mount).

En el caso del uso del servidor Samba, además del soporte *kernel*, necesitaremos instalar los paquetes software asociados: habrá que examinar qué paquetes relacionados con Samba hay en la distribución e instalar los que tengan que ver con el funcionamiento de servidor. Y también, si se quiere, los relacionados con Samba como cliente, en el caso de que deseemos ser clientes de máquinas Windows o testear desde nuestro GNU/Linux los recursos compartidos de las máquinas Windows. En una distribución Debian, estos paquetes son: samba, samba-common, smbclient, smbfs. También puede ser interesante instalar *swat*, que es una herramienta gráfica basada en web para la administración de los servicios Samba. Para nuestro servidor GNU/Linux de Samba [Woo00] [War03], del ejemplo propuesto, tendremos que transferir los contenidos del anterior disco D (donde teníamos nuestro sistema de ficheros compartido) de la máquina original a la nueva máquina y colocar su contenido en algún path, por ejemplo, /home/DATOS, ya sea por copia de *backup*, transferencia ftp, o usando Samba como cliente para transferir los archivos.

En cuanto a la utilización de GNU/Linux como cliente Samba, es bastante sencilla. Mediante el uso de comandos cliente para un uso ocasional de un sistema de ficheros:

a) Montamos un directorio compartido Windows (sea host el nombre del servidor Windows), en un punto de montaje predefinido (ya existente):

```
smbmount //host/carpeta /mnt/windows
```

b) Colocamos el acceso a la "carpeta" Windows de la máquina host en nuestro directorio local, accediendo en el árbol de directorios a:

```
/mnt/windows
```

c) A continuación, cuando ya no esté en uso, podemos desmontar el recurso con:

```
smbumount /mnt/windows
```

Si no conocemos los recursos compartidos, podemos obtener una lista con:

```
smbclient -L host
```

Y también podemos utilizar smbclient //host/carpeta, que es un programa parecido a un cliente ftp.

En caso de querer hacer los sistemas de ficheros disponibles permanentemente, o proporcionar determinadas configuraciones particulares, podemos estudiar el uso de *mount* directamente (las utilidades smbxxxx lo utilizan), ya sea con los sistemas de ficheros (soportados en el *kernel*) smbfs o cifs, teniendo en cuenta los parámetros (autentificación de usuarios/grupos Windows, u otros parámetros de servicio) que deberemos aportar dependiendo del caso, y de la configuración Samba preexistente [Ste07].

Nota

Consultar siempre las páginas man, o 'manuales', que acompañen al software.

En el caso del servidor Samba, una vez tengamos instalado todo el software Samba, tendremos que configurar el servidor a través de su fichero de configuración. Según la versión (o la distribución), este fichero puede estar en /etc/smb.conf o bien en /etc/samba/smb.conf. Las opciones aquí mostradas pertenecen a un Samba 3.x.x instalado sobre una Debian. Otras versiones pueden tener algunas modificaciones menores.

Durante la instalación de los paquetes de software es habitual que se nos pregunten algunos datos sobre su configuración. En el caso de Samba, se nos pregunta por el grupo de trabajo al que se va a servir; habrá que colocar el mismo nombre del grupo que en Windows. También se nos pregunta si deseamos contraseñas encriptadas (recomendable por seguridad, antes en los Windows 9x se enviaban en texto en bruto, en lo que constituye un claro ejemplo de escasa seguridad y de alta vulnerabilidad del sistema).

A continuación pasamos a ver el proceso de configuración del fichero smb.conf. Este fichero tiene tres secciones principales:

1) *Global* (características básicas de funcionamiento).

2) *Browser* (controla lo que otras máquinas ven de nuestros recursos).

3) *Share* (controla qué compartimos).

En el manual (extenso) de este fichero pueden verse las opciones disponibles (man smb.conf). Editaremos el fichero con algún editor e iremos viendo algunas de las líneas del fichero (los caracteres '#' o ';' a principio de línea son comentarios, si la línea contiene ';' es un comentario; para habilitar una línea, si es alguna línea opcional de configuración, deberemos editarla y quitar el ';'):

`workgroup` = *GRUPO*

Aquí está indicado el grupo de trabajo Windows del cual las máquinas Windows clientes serán miembros.

```
server string = %h server (Samba %v)
```

Podemos colocar una descripción textual de nuestro servidor, la *h* y la *v* que aparecen son variables de Samba, que hacen referencia al nombre del host y a la versión de Samba. Por seguridad, es mejor quitar la *v*, ya que con ello se informa al exterior de qué versión de Samba tenemos; si hay bugs de seguridad conocidos, esto puede aprovecharse.

```
hosts allow = 192.168.1
```

Esta línea puede estar presente o no, y la podemos incluir para habilitar qué hosts serán servidos; en este caso, todos los del rango 192.168.1.x.

```
printcap name = /etc/printcap
```

El fichero printcap es donde GNU/Linux guarda la definición de las impresoras, y es aquí donde Samba buscará la información acerca de éstas.

```
guest account = nobody
```

Ésta es la cuenta de "invitado". Podemos crear una cuenta diferente, o solamente habilitar el acceso a Samba a los usuarios dados de alta en el sistema GNU/Linux.

```
log file = /var/log/samba/log.%m
```

Esta línea nos dice dónde se van a guardar los ficheros del registro de sesión (logs) de Samba. Se guarda uno por cada cliente (variable *m* es el nombre del cliente conectado).

```
encrypt passwords = true
```

Es conveniente, por seguridad, usar encriptación de contraseñas (*passwords*) si tenemos máquinas clientes con Windows 98, NT o superiores. Estas contraseñas

se guardan en un fichero /etc/samba/smbpasswd, que normalmente se genera para los usuarios de la instalación de Samba. Las contraseñas se pueden cambiar con el comando *smbpasswd*. También hay una opción llamada *UNIX* passwdord sync, que permite que el cambio sea simultáneo a las dos contraseñas (usuario Samba y usuario Linux).

A continuación, saltaremos ahora a la sección "Share Definitions":

```
[homes]
```

Estas líneas permiten dar acceso a las cuentas de los usuarios desde las máquinas Windows. Si no lo queremos, añadimos unos ';' al inicio de estas líneas, y las máquinas al conectarse verán el nombre comment. En principio, la escritura está deshabilitada, para habilitarla, sólo hay que poner "yes" en la opción writable.

Cualquier compartición de un directorio concreto (en Samba se suele denominar *partición* a un grupo de datos compartidos), se hará como los ejemplos que aparecen (ver, por ejemplo, la definición de compartir el CD-ROM en las líneas que empiezan por [cdrom]). En path se coloca la ruta de acceso.

Ejemplo

En nuestro caso, por ejemplo, pondríamos un nombre DATOS a la partición en la ruta /home/DATOS, donde habíamos copiado el disco D de la máquina original Windows y el path donde se puede encontrar, además de un alto grupo de opciones que se pueden modificar, el usuario que podrá acceder a ellas y la manera de hacerlo.

Nota

Ver: man smb.conf

También hay una definición [*profiles*], que permite controlar los perfiles (*profiles*) de los usuarios Windows, o sea, el directorio donde se guarda su configuración de escritorio Windows, el menú de inicio, etc.

El método es parecido para las impresoras: se hace una partición con el nombre de la impresora (el mismo que se haya dado en GNU/Linux), y en el path se coloca la dirección de la cola asociada a la impresora (en GNU/Linux la encontramos en: /var/spool/samba/PRINTER). Y la opción printable = yes, si queremos que se envíen trabajos con Samba. Y también se puede restringir qué usuarios acceden (*valid users*).

Una vez hechos estos cambios, sólo tenemos que guardarlos y reiniciar Samba para que lea la nueva configuración. En Debian:

```
/etc/init.d/samba restart
```

Ahora, nuestro directorio compartido y la impresora por Samba estarán disponibles, de manera que sirvan a los usuarios sin que éstos noten diferencia alguna respecto a las conexiones anteriores con el servidor Windows.

Actividades

1. En la descripción de servicios GNU/Linux, ¿se encuentra a faltar alguna funcionalidad?, ¿qué otro tipo de servicios añadiríais?

2. En el segundo caso de estudio del taller (el de la organización), ¿cómo cambiaríais la infraestructura informática si dispusierais de un presupuesto de coste cero, un presupuesto medio, o un presupuesto alto? Presentad algunas soluciones diferentes a las expuestas.

3. El software VMware Workstation es una máquina virtual por software, que permite instalar operativos sobre un PC virtual. Se puede conseguir una demo en www.vmware.com. Probar (en el caso de disponer de una licencia Windows) a instalarla sobre Windows, y entonces un GNU/Linux sobre el PC virtual (o al contrario). ¿Qué ventajas nos aporta este sistema de compartir los operativos? ¿Qué problemas ocasiona?

4. Si se dispone de dos máquinas para instalar un servidor Samba, podemos probar la instalación o configuración del servidor en configuraciones de cliente Samba UNIX-servidor Windows, o cliente Windows-servidor Samba en GNU/Linux. En una sola máquina puede probarse, utilizando la misma máquina como servidor y cliente Samba.

Otras fuentes de referencia e información

[LPD] Linux Documentation Project proporciona los Howto's de los diferentes aspectos de un sistema GNU/Linux y un conjunto de manuales más elaborados.

[Mor03] Buena referencia de configuración de sistemas Linux, con algunos casos de estudio en diferentes entornos; comenta diferentes distribuciones Debian y Red Hat.

Herramientas básicas para el administrador

Josep Jorba Esteve

P07/M2103/02282

UOC
www.uoc.edu

Índice

Introducción

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.

Nota

GNU/Linux posee un conjunto muy amplio de herramientas con funcionalidades básicas, cuya potencia está en su combinación.

En esta unidad veremos diferentes grupos de herramientas, identificaremos algunas de sus funciones básicas, y veremos algunos 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*) [Linb], nos hablan de herramientas que esperamos encontrar disponibles, de una estructura común para el sistema de ficheros, así como de diversas normas que tienen que cumplirse para que una distribución sea considerada un sistema GNU/Linux y mantenga reglas comunes para la compatibilidad entre ellos.

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 exportables 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 diversos 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.

1. Herramientas gráficas y líneas de comandos

Existe un gran número de herramientas, de las que examinamos una pequeña porción en éste y 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.

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 trabajar con variantes comerciales de otros UNIX. Esto conlleva que tengamos que establecer una cierta manera de trabajar que nos permita hacer de forma uniforme las tareas en los diferentes sistemas.

Por esta razón, a lo largo de los diferentes módulos 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 [Sm02]. Pero hay que tener en cuenta varias cosas:

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 u optimización de servicios.

e) Estas herramientas son útiles en la mejora de la producción, una vez que el administrador tiene los conocimientos adecuados, ya que puede manejar 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 adecuadas: pueden no haber estado pensadas para los pasos que hay que realizar, o bien hacerlo de una forma no eficaz. Por ejemplo, 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 unos cuantos usuarios queremos introducir una lista de decenas o centenares de éstos?, la herramienta, si no está preparada, se vuelve totalmente ineficiente.

h) Por ultimo, 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* (veremos los fundamentos en esta unidad) para combinarlas de modo que formen una tarea.

Tenemos que saber valorar estas herramientas extra según la valía 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 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.

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*) [Linb], 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 que solemos encontrar en los sistemas [Linc].

Nota

Ver FHS en: www.pathname.com/fhs

En el estándar **FHS** (*filesystem hierchachy standard*) se describen la estructura de 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.

Nota

El estándar FHS 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.

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 necesarios durante el arranque del sistema, por ejemplo la imagen del *kernel* Linux, en /boot/vmlinuz.

- /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: /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 /dev/sdx... con x variando según el

número de dispositivo. Aquí cabe comentar que en su inicio este directorio era estático, con los ficheros predefinidos, y/o configurados en determinados momentos, hiy en dia se utilizan técnicas tecnologías dinámicas (como *hotplug* u *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.

- /home: contiene las cuentas de los usuarios, es decir, los directorios personales de cada usuario.

- /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; por ejemplo: /mnt/cdrom, para montar un disco en el lector de CD-ROM momentáneamente.

- /media: para punto de montaje habitual de dispositivos extraíbles.

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

- /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.

- /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.

- /usr: diferentes elementos instalados en el sistema. 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 software.

- /var: ficheros de registro de sesión o de estado (ficheros de tipo log) y/o errores del propio sistema y de diversos servicios, tanto locales como de

red. Por ejemplo, ficheros de sesión en /var/log, contenido de los correos en /var/spool/mail, o trabajos de impresión en /var/spool/lpd.

Éstos son algunos de los directorios definidos en el FHS para el sistema raíz, luego se especifican por ejemplo 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 (ver las referencias a los documentos FHS).

Respecto a las distribuciones, Fedora/Red Hat sigue el estándar FHS muy de cerca. Sólo presenta algunos cambios en los archivos presentes en /usr, /var. En /etc suele existir un directorio por componente configurable, y 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.

Nota

http://www.linuxbase.org/spec/refsp ecs/LSB_1.3.0/gLSB/gLSB/rstandard s.html

Otro estándar en proceso es el LSB (*Linux standard base*) [Linc]. La idea de éste 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

http://www.linuxbase.org/spec/

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, pero 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, normalmente, algunos ejemplos de utilización. A veces, puede haber más de una entrada en el manual. Por ejemplo, puede 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.

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. 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. Es básicamente una herramienta textual en la que los capítulos y 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. Normalmente, estos componentes de documentación se instalan en el directorio /usr/share/doc (O /usr/doc dependiendo de la distribución), donde normalmente se crea un directorio por paquete de aplicación (normalmente 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 su web. Debian trae los manuales como un paquete de software más y suelen instalarse en /usr/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, las cuales presentan interfaces web para examinar la documentación del sistema.

e) Por último, los escritorios X, como Gnome y KDE, normalmente 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).

4. *Shells* y *Scripts*

El término genérico *shell* se utiliza para denominar un programa que sirve de interfaz entre el usuario y el núcleo (*kernel*) del sistema GNU/Linux. En este apartado nos centraremos en los *shell*s interactivos de texto, que serán los que nos encontraremos como usuarios una vez estemos validados y dentro del sistema.

El *shell*, como programa, es una utilidad de sistema, que permite a los usuarios interactuar con el *kernel* por interpretación de comandos que el mismo usuario introduce en la línea de comandos o en los ficheros de tipo *shell script*.

El *shell* es lo que los usuarios ven del sistema. El resto del sistema operativo permanece esencialmente oculto a sus ojos. El *shell* está escrito de la misma forma que un proceso (programa) de usuario; no está integrado en el *kernel*, sino que se ejecuta como un programa más del usuario.

Cuando nuestro sistema GNU/Linux arranca, suele presentar a los usuarios una interfaz de cara determinada; esta interfaz puede ser tanto de texto, como gráfica. Dependiendo de los modos (o niveles) de arranque del sistema, ya sea con los diferentes modos de consola de texto o con modos donde directamente tengamos arranque gráfico en X Window.

En los modos de arranque gráfico, la interfaz está compuesta por algún administrador de acceso que gestiona el proceso de *login* del usuario desde una "carátula" gráfica, en la que se le pide la información de entrada correspondiente: su identificador como usuario y su palabra de paso (o *passsword*). En GNU/Linux suelen ser habituales los gestores de acceso: xdm (propio de X Window), gdm (Gnome) y kdm (KDE), así como algún otro asociado a diferentes gestores de ventanas *(window managers*). Una vez validado nuestro acceso, nos encontraremos dentro de la interfaz gráfica de X Window con algún gestor de ventanas, como Gnome o KDE. Para interactuar desde un *shell* interactivo, sólo tendremos que abrir alguno de los programas de emulación de terminal disponibles.

Si nuestro acceso es por modo consola (en texto), una vez validados obtendremos el acceso directo al *shell* interactivo.

Otro caso de obtención de un *shell* interactivo es el acceso remoto a la máquina, ya sea vía cualquiera de las posibilidades de texto como telnet, r*login*, ssh, o gráficas como los emuladores X Window.

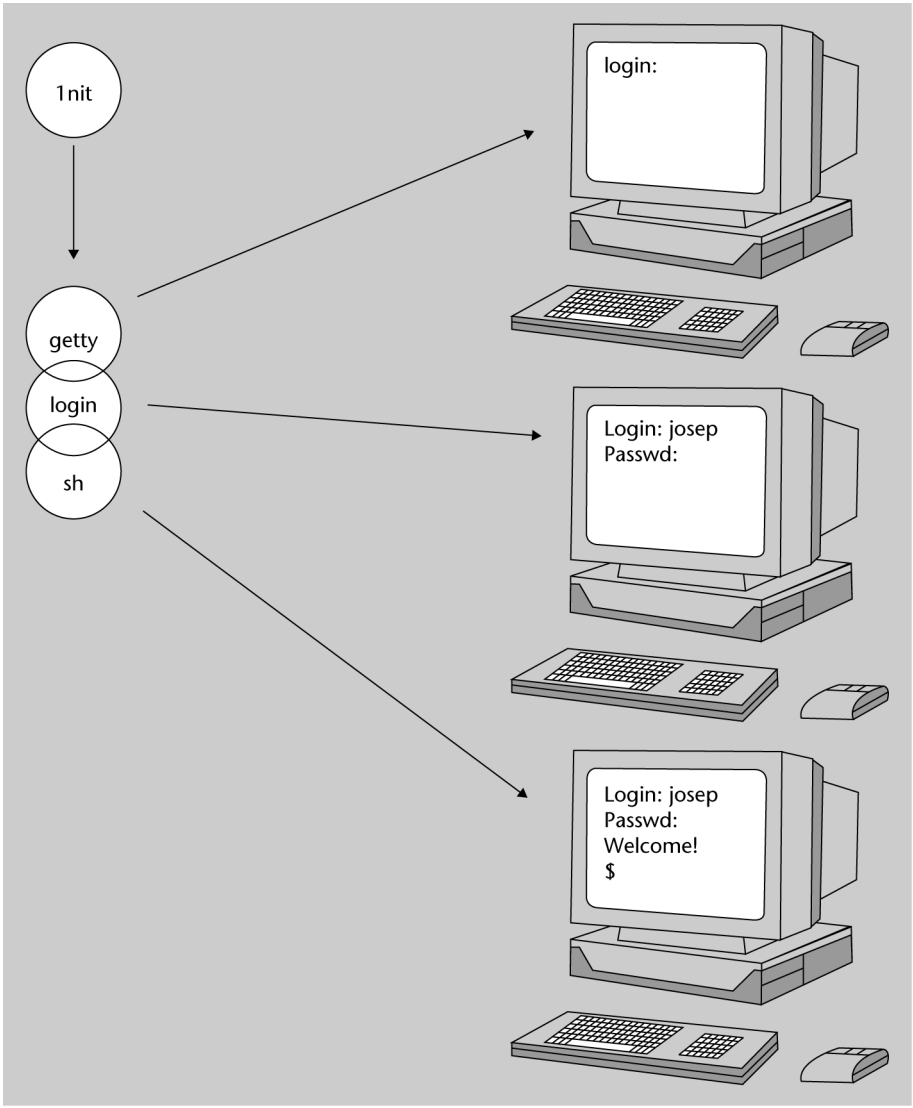

Figura 1. Ejemplo de arranque *shell* textual y procesos de sistema involucrados [Oke]

4.1. *Shells* interactivos

Una vez iniciado el *shell* interactivo [Qui01], se muestra un prompt de cara al usuario, indicándole que puede introducir una línea de comando. Tras la introducción, el *shell* asume la responsabilidad de validarla y poner los procesos necesarios en ejecución, mediante una serie de fases:

- Leer e interpretar la línea de comandos.
- Evaluar los caracteres "comodín" como $ * ? u otros.
- Gestionar las redirecciones de E/S necesarias, los pipes y los procesos en segundo plano (*background*) necesarios (&).
- Manejar señales.
- Preparar la ejecución de los programas.

Normalmente, las líneas de comandos podrán ser ejecuciones de comandos del sistema, comandos propios del *shell* interactivo, puesta en marcha de aplicaciones o *shell script*s.

Los *shell script*s son ficheros de texto que contienen secuencias de comandos de sistema, más una serie de comandos propios internos del *shell* interactivo, más las estructuras de control necesarias para procesar el flujo del programa (tipo *while, for*, etc.).

Los ficheros *script* son directamente ejecutables por el sistema bajo el nombre que se haya dado al fichero. Para ejecutarlos, se invoca el *shell* junto con el nombre del fichero, o bien se dan permisos de ejecución al *shell script*.

En cierta manera, podemos ver el *shell script* como código de un lenguaje interpretado que se ejecuta sobre el *shell* interactivo correspondiente. Para el administrador, los *shell script*s son muy importantes básicamente por dos razones:

1) La configuración del sistema y de la mayoría de los servicios proporcionados se hacen mediante herramientas proporcionadas en forma de *shell script*s.

2) La principal forma de automatizar procesos de administración es mediante la creación de *shell script*s por parte del administrador.

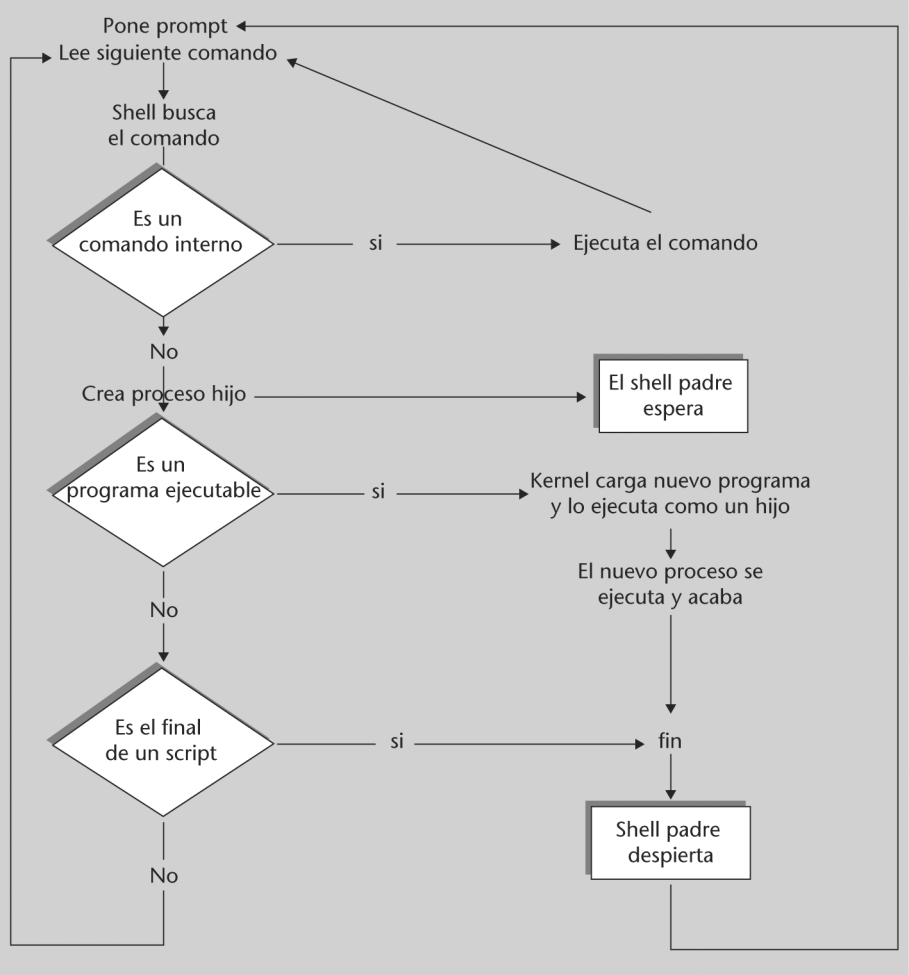

Figura 2. Flujo de control básico de un *shell*

Todos los programas invocados mediante un *shell* poseen tres ficheros predefinidos, especificados por los correspondientes des*cript*ores de ficheros (*file handles*). Por defecto, estos ficheros son:

1) *standard input* (entrada estándar): normalmente asignada al teclado del terminal (consola); usa el des*cript*or número 0 (en UNIX los ficheros utilizan des*cript*ores enteros).

2) *standard output* (salida estándar): normalmente asignada a la pantalla del terminal; usa el des*cript*or 1.

3) *standard error* (salida estándar de errores): normalmente asignada a la pantalla del terminal; utiliza el des*cript*or 2.

Esto nos indica que cualquier programa ejecutado desde el *shell* tendrá por defecto la entrada asociada al teclado del terminal, su salida hacia la pantalla y, en el caso de producirse errores, también los envía a la pantalla.

Además, los *shell*s suelen proporcionar los tres mecanismos siguientes:

1) Redirección: dado que los dispositivos de E/S y los ficheros se tratan de la misma manera en UNIX, el *shell* los trata a todos simplemente como ficheros. Desde el punto de vista del usuario, se pueden reasignar los des*cript*ores de los ficheros para que los flujos de datos de un des*cript*or vayan a cualquier otro des*cript*or; a esto se le llama redirección. Por ejemplo, nos referiremos a la redirección de los des*cript*ores 0 o 1 como a la redirección de la E/S estándar.

2) Tuberías (*pipes*): la salida estándar de un programa puede usarse como entrada estándar de otro por medio de *pipes*. Varios programas pueden ser conectados entre sí mediante *pipes* para formar lo que se denomina un *pipeline*.

3) Concurrencia de programas de usuario: los usuarios pueden ejecutar varios programas simultáneamente, indicando que su ejecución se va a producir en segundo plano (*background*), en términos opuestos a primer plano (o *foreground*), donde se tiene un control exclusivo de pantalla. Otra utilización consiste en permitir trabajos largos en segundo plano cuando interactuamos con el *shell* con otros programas en primer plano.

En los *shell*s UNIX/Linux estos tres aspectos suponen en la práctica:

- **Redirección:** un comando va a poder recibir su entrada o salida desde otros ficheros o dispositivos.

  Ejemplo

  veamos

  ```
  comando op fichero
  ```

donde op puede ser:

- < : recibir entrada del fichero.
- > : enviar salida al fichero.
- >> : indica que se añada la salida (por defecto, con > se crea de nuevo el fichero).

- *Pipes*: encadenamiento de varios comandos, con transmisión de sus datos:

```
comando1 | comando2 | comando3
```

  Esta instrucción nos indica que comando1 recibirá entrada posiblemente de teclado, enviará su salida a comando2, que la recibirá como entrada, y éste producirá salida hacia comando3, que la recibe y produce su salida hacia salida estándar (la pantalla, por defecto).

- Concurrencia en segundo plano: cualquier comando ejecutado con el '&' al final de línea se ejecuta en segundo plano y el *prompt* del *shell* se devuelve inmediatamente mientras continúa su ejecución. Podemos seguir la ejecución de los comandos con el comando *ps* y sus opciones, que nos permite ver el estado de los procesos en el sistema. Y también disponemos de la orden *kill*, que nos permite eliminar procesos que todavía se estén ejecutando o que hayan entrado en alguna condición de error: *kill 9 pid* permite "matar" el proceso número pid. pid es el identificador asociado al proceso, un número entero que el sistema le asigna y que puede obtenerse con el comando *ps*.

4.2. *Shells* disponibles

La independencia del *shell* respecto al *kernel* del operativo (el *shell* es sólo una capa de interfaz), nos permite disponer de varios de ellos en el sistema [Qui01]. Algunos de los más comunes son:

a) El *shell* Bash (*bash*). El *shell* GNU/Linux por defecto.

b) El *shell* Bourne (sh). Éste ha sido desde siempre el *shell* estándar UNIX, y el que todos los UNIX poseen en alguna versión. Normalmente, es el *shell* por defecto del administrador (*root*). En GNU/Linux suele ser el anterior Bash, una versión mejorada del Bourne. El sh fue creado por Stephen Bourne en AT&T a finales de los setenta. El indicador (o *prompt*) por defecto suele ser un '$' (en *root* un '#').

c) El *shell* Korn (ksh). Es un superconjunto del Bourne (se mantiene cierta compatibilidad), escrito en AT&T por David Korn (a mediados de los ochenta), en el cual se hizo cierta mezcla de funcionalidades del Bourne y del C, más algún añadido. El *prompt* por defecto es el $.

d) El *shell* C (csh). Fue desarrollado en la Universidad de Berkeley por Bill Joy a finales de los setenta y tiene unos cuantos añadidos interesantes al Bourne, como un histórico de comandos, alias, aritmética desde la línea de comandos, completa nombres de ficheros y hace control de trabajos en segundo plano. El *prompt* por defecto para los usuarios es '%'. Los usuarios UNIX suelen preferir este *shell* como interactivo, pero los administradores UNIX prefieren utilizar el Bourne, ya que los *script*s suelen quedar más compactos, y la ejecución suele ser más rápida. Por otro lado, una ventaja de los *script*s en C *shell* es que, como su nombre indica, su sintaxis está basada en el lenguaje C (aunque no igual).

e) Otros, como versiones restringidas o especializadas de los anteriores.

El *shell* Bash (*Bourne again shell*) [Bas] [Coo] ha adquirido importancia desde su inclusión en los sistemas GNU/Linux como *shell* por defecto. Este *shell* forma parte del software del proyecto GNU. Es un intento de combinar los tres *shell* anteriores (Bourne, C y Korn), manteniendo la sintaxis del *shell* Bourne original. Es en el que nos vamos a fijar para desarrollar ejemplos posteriores.

Una forma rápida de conocer bajo qué *shell* nos encontramos como usuarios es mediante la variable $*SHELL*, desde una línea de comandos con la instrucción:

```
echo $SHELL
```

Algunas cuestiones que encontraremos comunes a todos los *shell*s:

- Todos permiten la escritura de *shell script*s, que son luego interpretados ejecutándolos bien por el nombre (si el fichero tiene permiso de ejecución) o bien pasándolo como parámetro al comando del *shell*.

- Los usuarios del sistema tienen un *shell* por defecto asociado a ellos. Esta información se proporciona al crear las cuentas de los usuarios. El administrador asigna un *shell* a cada usuario, o bien si no se asigna el *shell* por defecto (*bash* en GNU/Linux). Esta información se guarda en el fichero de *passwords* en */etc/passwd,* y puede cambiarse con la orden *chsh,* esta misma orden con opción -l nos lista los *shell*s disponibles en el sistema (o ver también */etc/shells*).

- Cada *shell* es en realidad un comando ejecutable, normalmente presente en los directorios /bin en GNU/Linux (o /usr/bin).

- Se pueden escribir *shell script*s en cualquiera de ellos, pero ajustándose a la sintaxis de cada uno, que es normalmente diferente (a veces hay sólo pequeñas diferencias). La sintaxis de las construcciones, así como los comandos internos, están documentados en la página man de cada *shell* (man bash por ejemplo).

- Cada *shell* tiene algunos ficheros de arranque asociados (ficheros de inicialización), cada usuario puede adaptarlos a sus necesidades, incluyendo código, variables, caminos (path)...

- La potencia en la programación está en combinar la sintaxis de cada *shell* (de sus construcciones), con los comandos internos de cada *shell*, y una serie de comandos UNIX muy utilizados en los *script*s, como por ejemplo los cut, sort, cat, more, echo, grep, wc, awk, sed, mv, ls, cp...

Nota

Para la programación shell es recomendable tener un buen conocimiento de estos comandos UNIX, y sus diferentes opciones.

- Si como usuarios estamos utilizando un *shell* determinado, nada impide arrancar una copia nueva de *shell* (lo llamamos sub*shell*), ya sea el mismo u otro diferente. Sencillamente, lo invocamos por el nombre del ejecutable, ya sea el sh, bash, csh o ksh. También cuando ejecutamos un *shell script* se lanza un sub*shell* con el *shell* que corresponda para ejecutar el *script* pedido.

Algunas diferencias básicas entre ellos [Qui01]:

a) Bash es el *shell* por defecto en GNU/Linux (si no se especifica lo contrario al crear la cuenta del usuario). En otros sistemas UNIX suele ser el *shell* Bourne (sh). Bash es compatible con sh, y además incorpora algunas características de los otros *shell*s, csh y ksh.

b) Ficheros de arranque: sh, ksh tienen *.profile* (en la cuenta del usuario, y se ejecuta en el *login* del usuario) y también ksh suele tener un *.kshrc* que se ejecuta a continuación, csh utiliza *.login* (se ejecuta al iniciarse el *login* del usuario una sola vez), *.logout* (antes de la salida de la sesión del usuario) y *.cshrc* (parecido al *.profile*, en cada sub*shell* C que se inicia). Y Bash utiliza el *.bashrc* y el *.bash_profile*. Además, el administrador puede colocar variables y caminos comunes en el fichero */etc/profile* que se ejecutará antes que los ficheros que tenga cada usuario. Los ficheros de inicialización de los *shell* se ponen en la cuenta del usuario al crearla (normalmente se copian del directorio /etc/skel), donde el administrador puede dejar unos esqueletos de los ficheros preparados.

c) Los *script*s de configuración del sistema o de servicios suelen estar escritos en *shell* Bourne (sh), ya que la mayoría de los UNIX así los utilizaban. En GNU/Linux también nos podemos encontrar con algunos en Bash, y también en otros lenguajes de *script* no asociados a los *shell* como perl o python.

d) Podemos identificar en qué *shell* se ejecuta el *script* mediante el comando *file*, por ejemplo *file <nombrescript>*. O bien examinando la primera línea del *script*, que suele ser: *#!/bin/nombre*, donde nombre es *bash, sh, csh, ksh*... Esta línea le dice, en el momento de ejecutar el *script*, qué *shell* hay que utilizar a la hora de interpretarlo (o sea, qué sub*shell* hay que lanzar para ejecutarlo). Es importante que todos los *script*s la contengan, ya que si no, se intentarán ejecutar en el *shell* por defecto (Bash en nuestro caso), y la sintaxis puede no corresponder, causando muchos errores sintácticos en la ejecución.

4.3. Variables del sistema

Algunas variables de sistema útiles (podemos verlas por ejemplo, mediante el comando *echo*), que pueden consultarse ya sea en la línea de comandos o dentro en la programación de *shell*s *script* son:

Variable	Valor Ejemplo	Descripción
HOME	/home/juan	Directorio raíz del usuario
LOGNAME	Juan	Id del usuario en el *login*
PATH	/bin:/usr/local/bin:/usr/X11/bin	Caminos
SHELL	/bin/bash	*Shell* del usuario
PS1	$	Prompt del *shell*, el usuario puede cambiarlo
MAIL	/var/mail/juan	Directorio del buzón de correo
TERM	xterm	Tipo de terminal que el usuario utiliza
PWD	/home/juan	Directorio actual del usuario

Las diferentes variables del entorno pueden verse con el comando *env*. Por ejemplo:

```
$ env
SSH_AGENT_PID = 598
MM_CHARSET = ISO-8859-15
TERM = xterm
DESKTOP_STARTUP_ID =
SHELL = /bin/bash
WINDOWID = 20975847
LC_ALL = es_ES@euro
USER = juan
LS_COLORS = no = 00:fi = 00:di = 01;34:ln = 01;
SSH_AUTH_SOCK = /tmp/ssh-wJzVY570/agent.570
SESSION_MANAGER = local/aopcjj:/tmp/.ICE-unix/570
USERNAME = juan
PATH=/soft/jdk/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/
X11:/usr/games
MAIL = /var/mail/juan
PWD = /etc/skel
JAVA_HOME = /soft/jdk
LANG = es_ES@euro
GDMSESSION = Gnome
JDK_HOME = /soft/jdk
SHLVL = 1
HOME = /home/juan
GNOME_DESKTOP_SESSION_ID = Default
LOGNAME = juan
DISPLAY = :0.0
```

```
COLORTERM = gnome-terminal
XAUTHORITY = /home/juan/.Xauthority
_ = /usr/bin/env
OLDPWD = /etc
```

4.4. Programación *scripts* en Bash

Aquí veremos algunos conceptos básicos de los *shell script*s en Bash, se recomienda ampliar en [Bas] [Coo].

Todos los *script*s Bash tienen que comenzar con la línea:

```
#!/bin/bash
```

Esta línea le indica al *shell* usado por el usuario, el que tengamos activo en el momento, con qué *shell* hay que ejecutar el *script* que aparezca a continuación.

El *script* puede ejecutarse de dos modos diferentes:

1) Por ejecución directa desde la línea de comandos, siempre que tenga permiso de ejecución. Si no se da el caso, ponemos el permiso con: *chmod +x script*.

2) Por ejecución mediante el *shell*, llamamos al *shell* explícitamente: */bin/bash script*.

Hay que tener en cuenta que, sea cuál sea el método de ejecución, siempre estamos creando un sub*shell* donde se va a ejecutar nuestro *script*.

4.4.1. Variables en Bash

La asignación de variables se realiza por:

```
variable = valor
```

El valor de la variable se puede ver con:

```
echo $variable
```

donde '$' nos hace referencia al valor de la variable.

La variable por defecto sólo es visible en el *script* (o en el *shell*). Si la variable tiene que ser visible fuera del *script*, a nivel de *shell* o de cualquier *shell* hijo (o

subshell) que se genere a posteriori, será necesario "exportarla" además de asignarla. Podemos hacer dos cosas:

- Asignar y exportar después:

```
var = valor
export var
```

- Exportar en la asignación:

```
export var = valor
```

En los *script*s Bash tenemos algunas variables predeterminadas accesibles:

- $1-$N: Guarda los argumentos pasados como parámetros al *script* desde la línea de comandos.
- $0: Guarda el nombre del *script*, sería el parámetro 0 de la línea de comandos.
- $*: Guarda todos los parámetros del 1 al N en esta variable.
- $: Guarda todos los parámetros, pero con comillas dobles (" ") en cada uno de ellos.
- $?: "Status": guarda el valor devuelto por el último comando ejecutado. Útil para verificar condiciones de error, ya que UNIX suele devolver 0 si la ejecución ha sido correcta, y un valor diferente como código de error.

Otra cuestión importante en las asignaciones es el uso de las comillas:

- Las "dobles" permiten que sea considerado todo como una unidad.
- Las 'simples' son parecidas, pero se ignoran los caracteres especiales que se encuentren dentro.
- Las inclinadas hacia la izquierda `comando`, son utilizadas para evaluar el interior, si hay alguna ejecución o sustitución que hacer. Primero se ejecuta el contenido, y se sustituye lo que había por el resultado de la ejecución. Por ejemplo: var = `ls` guarda el listado del directorio en $var.

4.4.2. Comparaciones

Para las condiciones se suele utilizar la orden *test expresión* o directamente *[expresión]*. Podemos agrupar las condiciones disponibles en:

- Comparación numérica: -eq, -ge, -gt, -le, -lt, -ne, correspondiendo a: igual que, más grande o igual que (ge), más grande que, menor o igual que (le), menor que, distinto que.

- Comparación de cadenas: :=, !=, -n, -z, correspondiendo a cadenas de caracteres: iguales, diferentes, con longitud mayor que 0, longitud igual a cero o vacío.
- Comparación de ficheros: -d, -f -r, -s, -w, -x. El fichero es: un directorio, un fichero ordinario, es leíble, es no vacío, es escribible, es ejecutable.
- Booleanos entre expresiones: !, -a, -o, condiciones de not, and y or.

4.4.3. Estructuras de control

Respecto a la programación interna del *script*, hay que pensar que nos vamos a encontrar básicamente con:

- Comandos propios del operativo.
- Comandos propios internos del *shell* Bash (ver: man bash).
- Las estructuras de control propias de programación (for, while...), con la sintaxis propia de Bash.

La sintaxis básica de las estructuras de control es la siguiente:

a) Estructura *if...then*, se evalúa la expresión y si se obtiene un valor cierto, entonces se ejecutan los commands.

```
if [ expresion ]
  then
    commands
fi
```

b) Estructura if..then...else, se evalúa la expresión, y si se obtiene un valor de cierto, entonces se ejecutan los commands1, en caso contrario se ejecutan los comands2:

```
if [ expresion ]
  then
    commands1
  else
    commands2
fi
```

c) Estructura if..then...else if...else, misma utilización que la anterior, con anidamientos de estructuras if.

```
if [ expresion ]
  then
    commands
  elif [ expresion2 ]
  then
    commands
```

```
  else
    commands
fi
```

d) Estructura *case select,* estructura de selección múltiple según valor de selección (en *case*)

Nota

Los shells como Bash, ofrecen un conjunto amplio de estructuras de control que los hace comparables a cualquier otro lenguaje.

```
case string1 in
  str1)
    commands;;
  str2)
    commands;;
  *)
    commands;;
esac
```

e) Bucle *for*, sustitución de variable por cada elemento de la lista:

```
for var1 in list
do
   commands
done
```

f) Bucle *while*, mientras se cumpla la expresión:

```
while [ expresion ]
do
   commands
done
```

g) Bucle *until*, hasta que se cumpla la expresión:

```
until [ expresion ]
do
   commands
done
```

h) Declaración de funciones:

```
fname() {
   commands
}
```

o bien con llamada acompañada de parámetros:

```
fname2(arg1,arg2...argN) {
   commands
}
```

y la llamadas de la función con fname o fname2 p1 p2 p3 ... pN.

5. 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, normalmente formando un único fichero (denominado paquete), con un formato propio y normalmente comprimido, que es el que se distribuye, ya sea vía CD, disquete o mediante acceso a servicios de ftp o web.

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

En el contenido de la distribución (sus CD) 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: editores, compiladores, depuradores... d) gráficos: controladores e interfaces gráficas, escritorios, gestores de ventanas... e) otras categorías.

Normalmente, 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) Descomprimir el contenido del paquete, copiando los ficheros a sus localizaciones definitivas, ya sean absolutas (tendrán una posición fija) o si se permite reubicarlas 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]), o pueden necesitar hacerlos todos a mano (caso .tgz) dependiendo de las herramientas que proporcione la distribución.

Veremos a continuación quizás los tres paquetes más clásicos de la mayoría de las distribuciones. Cada distribución tiene uno por estándar y soporta alguno de los demás.

5.1. Paquete TGZ

Los paquetes TGZ son quizás los de utilización más antigua. Las primeras distribuciones de GNU/Linux los utilizaban para instalar el software, y aún varias distribuciones los usan (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.

Nota

Los paquetes TGZ son una herramienta básica a la hora de instalar software no organizado. Además, son una herramienta útil para realizar procesos de backup y restauración de archivos.

Este tipo de paquete no contiene ninguna 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.

En contra de lo que pudiera parecer, 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.

Un ejemplo de este caso 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 del navegador web Mozilla):

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

donde tenemos el nombre del paquete, como Mozilla, 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 con em64t), o ia64 (intel Itaniums) otras para arquitecturas de otras máquinas como sparc, powerpc, mips, hppa, alpha... después nos indica qué es para Linux, en una máquina PC, la versión del software 1.4.

Si fuese en formato fuente, suele aparecer como:

```
mozilla-source-1.4.tar.gz
```

donde se nos indica la palabra *source*; en este caso no menciona la versión de arquitectura de máquina, esto nos indica que está preparado para compilarse en diferentes arquitecturas.

De otro modo, habría diferentes códigos para cada sistema operativo o fuente: GNU/Linux, Solaris, Irix, 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 parte):

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

Con el comando *tar* ponemos opciones de z: descomprimir, x: extraer ficheros, v: ver proceso, f: dar nombre del fichero a tratar.

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 por si disponemos del software adecuado, y si no, 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. Entonces, antes de instalar el software tendremos que pasar por una compilación. Para eso habrá que leerse con cierto detalle las instrucciones que lleve el programa. Pero la mayoría de desa-

rrolladores usan un sistema de GNU llamado autoconf (de autoconfiguración), en el que habitualmente se usan los siguientes pasos (si no aparecen errores):

- ./*configure*: se trata de un *script* que configura el código para poder ser compilado en nuestra máquina, verifica que existan las herramientas adecuadas. La opción --prefix = directorio permite especificar dónde se instalará el software.

- *make*: compilación propiamente dicha.

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

Éste 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 ficheros que se instalaron, teniendo cuidado de posibles dependencias.

Los paquetes tgz 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 utilizarse 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 -zcvf 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).

5.2. Fedora/Red Hat: paquetes RPM

El sistema de paquetes RPM [Bai03] 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, em64t, ia64) u otras como alpha, aparc, ppc... La "arquitectura" Noarch suele usarse cuando es independiente, por ejemplo, un conjunto de *script*s, y src en el caso de que se trate de paquetes de código fuente. La ejecución típica incluye la ejecución de rpm, las opciones de la operación a realizar, junto con uno o más nombres de paquetes por procesar juntos.

Nota

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:

- **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. Por ejemplo, preguntas a un paquete instalado:

Consulta	Opciones RPM	Resultados
Archivos	`rpm -ql`	Lista de los archivos que contiene
Información	`rpm -qi`	Descripción del paquete
Requisitos	`rpm -qR`	Requisitos previos, bibliotecas o software

- **Instalación:** simplemente rpm -i paquete.rpm, o bien con URL donde encontrar el paquete, para descargarlo desde servidores FTP o web, sólo 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 se ignora la información de las dependencias.

  La tarea de instalación (realizada por rpm) de un paquete conlleva diferentes subtareas: a) verificar las posibles dependencias; b) examinar por conflictos con otros paquetes previamente instalados; c) efectuar tareas previas a la instalación; c) decidir qué hacer con los ficheros de configuración asociados al paquete si previamente existían; d) desempaquetar los ficheros y colocarlos en el sitio correcto; e) llevar a cabo tareas de postinstalación; finalmente, f) almacenar registro de las tareas efectuadas 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 verificará todos.

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

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.

Hay que vigilar con la procedencia de los paquetes, y sólo utilizar fuentes de paquetes conocidas y fiables, ya sea del propio fabricante de la distribución, o sitios en los que confiemos. Normalmente se nos ofrece, junto con los paquetes, alguna "firma" digital de éstos 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. Asimismo, podemos encontrar en Internet diferentes almacenes de paquetes RPM, donde están disponibles para diferentes distribuciones que usen o permitan el formato RPM.

Nota

Ver el sitio: www.rpmfind.net.

Para un uso seguro de paquetes, actualmente 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. Normalmente, cada proveedor (el repositorio) incluye unos ficheros de firma PGP con la clave para su sitio. De los repositorios oficiales normalmente ya vienen instaladas, si proceden 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, normalmente este fichero también tendrá suma md5 para comprobar su integridad. Y podemos conocer las claves existentes en el sistema con:

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

podemos observar más detalles a partir de la clave 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 sólo aquellas claves de los sitios en los que confiemos. Cuando RPM encuentre paquetes con firma que no tengamos 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), hay soporte para RPM (existe el comando rpm), pero sólo 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 soportar 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 de automática de dependencias entre los paquetes. Permite acceder a múltiples repositorios diferentes, centraliza su configuración en un fichero (/etc/yum.conf habitualmente), y tiene una interfaz de comandos simple.

Nota

YUM en: http://linux.duke.edu/projects/yum

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 gpg).

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
yum remove <nombre>	Eliminar paquete
yum list <nombre>	Buscar paquete por nombre (sólo 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

Para finalizar, Fedora también ofrece un par de utilidades gráficas para YUM, pup para controlar las actualizaciones recientes disponibles, y pirut como paquete de gestión de software. También existen algunas otras como yumex, con mayor control de la configuración interna de yum.

5.3. 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 nos permite navegar por toda la lista de paquetes disponible (hay miles), y escoger aquellos que queramos instalar o desinstalar. De hecho estas son sólo un front-end del gestor de software nivel intermedio APT.

En el nivel de línea de comandos dispone de dpkg, que es el comando de más bajo nivel (*base*, sería el equivalente a rpm), para gestionar directamente los paquetes DEB de software [Deb02], 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 a través 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; un ejemplo podría ser:

```
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

#Sources Oficiales de Debian STABLE "etch"
deb http://ftp.debian.org/debian/ etch main non-free contrib
debsrc http://ftp.debian.org/debian/ etch main non-free contrib
```

Donde hay recopiladas varias de las fuentes "oficiales" para una Debian (*etch* en este caso, suponiendo ésta como *stable*), 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 o etch en este ejemplo), 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.

Nota

Los paquetes DEB de Debian es quizás el sistema de instalación más potente existente en GNU/Linux. Una prestación destacable 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 permite una interfaz interactiva. De hecho se espera que aptitude sea la interfaz por defecto en línea de comandos para la gestión de paquetes.

Algunas funciones básicas de apt-get:

- Instalación de un paquete particular:

```
apt-get install paquete
```

- Borrado de un paquete:

```
apt-get remove paquete
```

- Actualización de la lista de paquetes disponibles:

```
apt-get update
```

- 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. Unas herramientas útiles 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 (podemos configurarlas con apt-setup), o bien copiar algún fichero de fuentes disponible. Software adicional (de terceros) puede necesitar añadir otras fuentes más (a sources.list); se pueden obtener listas de sitios de fuentes disponibles (por ejemplo: http://www.apt-get.org).

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 caché, 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 si vamos a volver a necesitar estos paquetes por razones de reinstalación, ya que, si es así, tendremos 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 de este modo:

```
# 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 por defecto al iniciar el 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). Utilizándose 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 (ver instrucciones del repositorio para comprobar la forma recomendada de importar la clave y los datos necesarios).

Otra funcionalidad importante del sistema AP 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 comandos que nos permiten buscar información sobre los paquetes, como por ejemplo:

- Buscar paquetes sobre la base de un nombre incompleto:

  ```
  apt-cache search nombre
  ```

- Mostrar la descripción del paquete:

  ```
  apt-cache show paquete
  ```

- De qué paquetes depende:

  ```
  apt-cache depends paquete
  ```

Otras herramientas o funcionalidades de apt interesantes:

- apt-show-versions: nos especifica qué paquetes pueden ser actualizados (y por qué versiones, ver opción -u).

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

```
dpkg -L paquete
```

La lista de paquetes entera con

```
dpkg -l
```

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

```
dpkg -S fichero
```

Éste 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, gnome-apt para gnome, y kpackage o adept para KDE. O las textuales ya mencionadas como aptitude o dselect.

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 que se usa en Debian y sus distribuciones derivadas como Ubuntu, Kubuntu, Knoppix, Linex, etc.

6. 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 destacar que para estas herramientas es difícil mantenerse al día, debido a los actuales planes de versiones de las distribuciones, que tienen una evolución muy rápida. Dejamos algunos ejemplos (pero en un momento determinado pueden no ser funcionales al completo):

a) Linuxconf: es una herramienta genérica de administración que agrupa diferentes aspectos en una interfaz de menús textual, que en las ultimas versiones evolucionó a soporte web; puede utilizarse en prácticamente cualquier distribución GNU/Linux, y soporta diversos detalles propios de cada una (por desgracia lleva tiempo sin una actualización).

Nota

Podemos encontralas en: Linuxconf http://www.solucorp.qc.c a/linuxconf

b) Webmin: es otra herramienta de administración pensada desde una interfaz web; funciona con una serie de *plugins* que se pueden añadir para cada servicio que hay que administrar; normalmente 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.

c) 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, a la distribución Fedora (Red Hat y derivados), que intenta disponer de diversas 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, im-

presora, red, seguridad, usuarios, paquetes, etc. Podemos ver en la figura algunas de ellas:

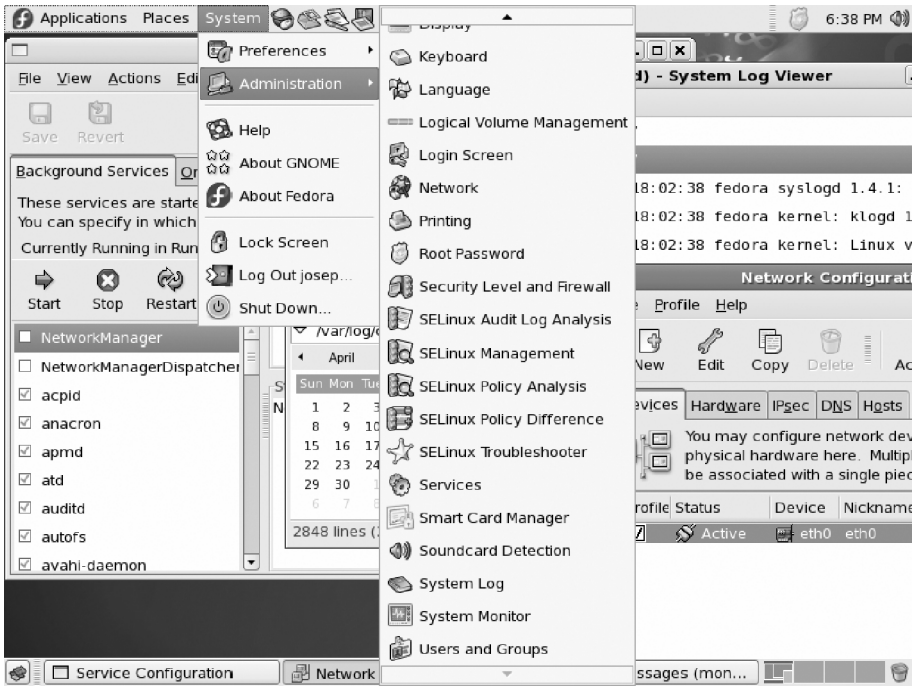

Figura 3. Varias de las utilidades gráficas de administración en Fedora

7. 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:

- Los múltiples comandos 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 traer 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, editor compatible con Wordstar. Y muchos otros...

Nota

Ver material asociado a curso de introducción a GNU/ Linux, o las páginas man de los comandos, o una referencia de herramientas como [Stu01].

- Lenguajes de tipo *script*, útiles para administración, como: Perl, muy útil 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++), `gdb` (depurador), `xxgdb` (interfaz X para gdb), `ddd` (depurador para varios lenguajes).

Actividades

1) Hacer una lectura rápida del estándar FHS, que nos servirá para tener una buena guía a la hora de buscar archivos por nuestra distribución.

2) Para repasar y ampliar conceptos y programación de *shell script*s en bash, ver: [Bas] [Coo].

3) Para los paquetes RPM, ¿cómo haríamos algunas de las siguientes tareas?:
- Conocer qué paquete instaló un determinado comando.
- Obtener la descripción del paquete que instaló un comando.
- Borrar un paquete cuyo nombre completo no conocemos.
- Mostrar todos los archivos que estaban en el mismo paquete que un determinado archivo.

4) Efectuar las mismas tareas que en la actividad anterior, pero para paquetes Debian, usando herramientas APT.

5) Actualizar una distribución Debian (o Fedora).

6) Instalar en nuestra distribución alguna herramienta genérica de administración, ya sea por ejemplo Linuxconf o Webadmin. ¿Qué nos ofrecen? ¿Entendemos las tareas ejecutadas y los efectos que provocan?

Otras fuentes de referencia e información

[Bas][Coo] ofrecen una amplia introducción (y conceptos avanzados) a la programación de *shell script*s en bash, así como numerosos ejemplos. [Qui01] comenta los diferentes *shell*s de programación en GNU/Linux, así como sus semejanzas y diferencias.

[Deb02][Bai03] ofrecen una amplia visión de los sistemas de paquetes de software de las distribuciones Debian y Fedora/Red Hat.

[Stu] es una amplia introducción a las herramientas disponibles en GNU/Linux.

El *kernel*

Josep Jorba Esteve

P07/M2103/02283

UOC
www.uoc.edu

Índice

Introducción

El *kernel* del sistema GNU/Linux (al que habitualmente se le denomina Linux) [Vasb] es el corazón del sistema: se encarga de arrancar el sistema y, una vez éste ya es utilizable por las aplicaciones y los usuarios, gestiona los recursos de la máquina en forma de gestión de la memoria, sistema de ficheros, entrada/salida, procesos e intercomunicación de procesos.

Su origen se remonta al año 1991, cuando en agosto, un estudiante finlandés llamado Linus Torvalds anunció en una lista de *news* que había creado su propio núcleo de sistema operativo que funcionaba conjuntamente con software del proyecto GNU, y lo ofrecía a la comunidad de desarrolladores para que ésta lo probara y sugiriera mejoras para una mejor utilización. Así, se constituyó el origen del *kernel* del operativo, que más tarde se llamaría Linux.

Una de las particularidades de Linux es que, siguiendo la filosofía del Software Libre, se nos ofrece el código fuente del propio sistema operativo (del *kernel*), de manera que es una herramienta perfecta para la educación en temas de sistemas operativos.

Nota

El *kernel* Linux se remonta al año 1991, cuando Linus Torvalds lo puso a disposición de la comunidad. Es de los pocos operativos que, siendo ampliamente usado, se puede disponer de su código fuente.

La otra ventaja principal es que, al disponer de las fuentes, podemos recompilarlas para adaptarlas mejor a nuestro sistema, y podemos, asimismo, configurar sus parámetros para dar un mejor rendimiento al sistema.

En esta unidad veremos cómo manejar este proceso de preparación de un *kernel* para nuestro sistema: cómo, partiendo de las fuentes, podemos obtener una nueva versión del *kernel* adaptada a nuestro sistema. Del mismo modo, trataremos cómo se desarrolla la configuración y la posterior compilación y cómo realizar pruebas con el nuevo *kernel* obtenido.

1. El *kernel* del sistema GNU/Linux

El núcleo o *kernel* es la parte básica de cualquier sistema operativo [Tan87], y en él descansa el código de los servicios fundamentales para controlar el sistema entero. Básicamente, su estructura se puede separar en una serie de componentes de gestión orientados a:

- Gestión de procesos: qué tareas se van a ejecutar y en qué orden y con qué prioridad. Un aspecto importante es la planificación de CPU: ¿cómo se optimiza el tiempo de la CPU para ejecutar las tareas con el mayor rendimiento o interactividad posible con los usuarios?

- Intercomunicación de procesos y sincronización: ¿cómo se comunican tareas entre sí, con qué diferentes mecanismos y cómo pueden sincronizarse grupos de tareas?

- Gestión entrada/salida (E/S): control de periféricos y gestión de recursos asociados.

- Gestión de memoria: optimización del uso de la memoria, sistema de paginación y memoria virtual.

- Gestión de ficheros: cómo el sistema controla y organiza los ficheros presentes en el sistema y el acceso a los mismos.

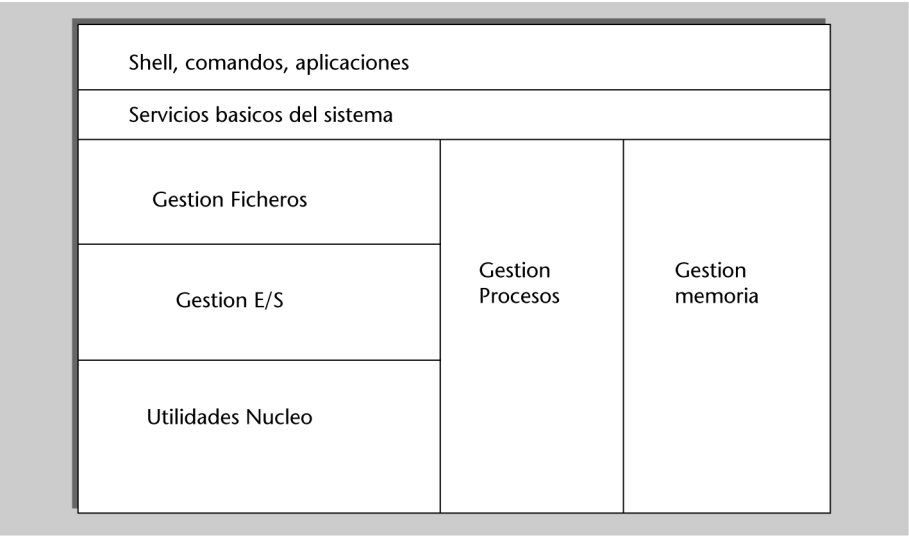

Figura 1. Funciones básicas de un *kernel* respecto de las aplicaciones y comandos ejecutados

En los sistemas propietarios, el *kernel* está perfectamente "oculto" bajo las capas del software del sistema operativo, y el usuario final no tiene una perspectiva clara de qué es el *kernel*, y no tiene ninguna posibilidad de cambiarlo u optimizarlo, si no es por el uso de esotéricos editores de "registros" internos,

o programas especializados de terceros (normalmente de alto coste). Además, el *kernel* suele ser único, es el que el fabricante proporciona, y éste se reserva el derecho de introducir las modificaciones que quiera y cuando quiera, así como tratar los errores que aparezcan en plazos no estipulados, mediante actualizaciones que nos ofrecen como "parches" de errores.

Uno de los principales problemas de esta aproximación es precisamente la disponibilidad de estos parches, disponer de las actualizaciones de los errores a su debido tiempo, y si son de seguridad, todavía más, ya que hasta que no estén corregidos no podemos garantizar la seguridad del sistema, para problemas ya conocidos. Muchas organizaciones, grandes empresas, gobiernos, instituciones científicas y militares no pueden depender de los caprichos de un fabricante para solucionar los problemas de sus aplicaciones críticas.

En este caso, el *kernel* Linux ofrece una solución de código abierto, con los consecuentes permisos de modificación, corrección, posibilidad de generación de nuevas versiones y actualizaciones de forma rápida, por parte de cualquiera que quiera y tenga los conocimientos adecuados para realizarlo.

Esto permite a los usuarios críticos controlar mejor sus aplicaciones y el propio sistema, y poder montar sistemas con el propio operativo "a la carta", personalizado al gusto de cada uno. Y disponer a su vez de un operativo con código abierto, desarrollado por una comunidad de programadores coordinados desde Internet, accesible ya sea para educación por disponer del código fuente y abundante documentación, o para la producción final de los sistemas GNU/Linux adaptados a necesidades individuales o de un determinado colectivo.

Al disponer de código fuente, se pueden aplicar mejoras y soluciones de forma inmediata, a diferencia del software propietario, donde debemos esperar las actualizaciones del fabricante. Podemos, además, personalizar el *kernel* tanto como necesitemos, requisito esencial, por ejemplo, en aplicaciones de alto rendimiento, críticas en el tiempo o soluciones con sistemas empotrados (como dispositivos móviles).

Siguiendo un poco de historia (rápida) del *kernel* [Kera] [Kerb]: lo comenzó a desarrollar un estudiante finlandés llamado Linus Torvalds, en 1991, con la intención de realizar una versión parecida a Minix [Tan87] (versión para PC de UNIX [Bac86]) para el procesador 386 de Intel. La primera versión publicada oficialmente fue la de Linux 1.0 en marzo de 1994, en la cual se incluía sólo la ejecución para la arquitectura i386 y soportaba máquinas de un sólo procesador. Linux 1.2 fue publicado en marzo de 1995, y fue la primera versión en dar cobertura a diferentes arquitecturas como Alpha, Sparc y Mips. Linux 2.0, en junio de 1996, añadió más arquitecturas y fue la primera versión en incorporar soporte multiprocesador (SMP) [Tum]. En Linux 2.2, enero de 1999, se incrementaron las prestaciones de SMP de manera significativa, y se añadieron controladores para gran cantidad de hardware. En la 2.4, en enero del

2001, se mejoró el soporte SMP, se incorporan nuevas arquitecturas soportadas y se integraron controladores para dispositivos USB, PC card (PCMCIA de los portátiles), parte de PnP (*plug and play*), soporte de RAID y volúmenes, etc. En la rama 2.6 del *kernel* (diciembre 2003), se mejora sensiblemente el soporte SMP, mejor respuesta del sistema de planificación de CPU, el uso de *threads* en el *kernel*, mejor soporte de arquitecturas de 64bits, soporte de virtualización, y mejor adaptación a dispositivos móviles.

Respecto al desarrollo, el *kernel*, desde su creación por Linus Torvalds en 1991 (versión 0.01), lo ha seguido manteniendo él mismo, pero a medida que su trabajo se lo permitía, y a medida que el *kernel* maduraba (y crecía), se ha visto ayudado a mantener las diferentes versiones estables del *kernel* por diferentes colaboradores, mientras Linus continuaba (en la medida de lo posible) desarrollando y recopilando aportaciones para la última versión de desarrollo del *kernel*. Los colaboradores principales en estas versiones han sido [lkm]:

Nota

El *kernel* tiene sus orígenes en el sistema MINIX, desarrollado por Andrew Tanenbaum, como un clon de UNIX para PC.

- 2.0 David Weinehall.
- 2.2 Alan Cox (también desarrolla y publica parches para la mayoría de versiones).
- 2.4 Marcelo Tosatti.
- 2.6 Andrew Morton / Linus Torvalds.

Para ver un poco la complejidad del *kernel* de Linux, veamos una tabla con un poco de su historia resumida en las diferentes versiones y su tamaño respectivo del código fuente. En la tabla se indican las versiones de producción únicamente; el tamaño (aproximado) está especificado en miles de líneas (K) de código fuente:

Versión	Fecha de la publicación	Líneas código (miles)
0.01	09-1991	10
1.0	03-1994	176
1.20	03-1995	311
2.0	06-1996	649
2.2	01-1999	1800
2.4	01-2001	3378
2.6	12-2003	5930

Como podemos comprobar, hemos pasado de unas diez mil líneas a seis millones.

En estos momentos el desarrollo continúa en la rama 2.6.x del *kernel*, la última versión estable, que incluye la mayoría de distribuciones como versión por defecto (algunas todavía incluyen algunas 2.4.x, pero 2.6.x suele ser una opción en la instalación); aunque cierto conocimiento de las anteriores versiones es imprescindible, ya que con facilidad podemos encontrar máquinas con distribuciones antiguas que no se hayan actualizado, a las que es posible que deba-

Nota

El kernel hoy en día ha alcanzado un grado de madurez y complejidad significativo.

mos mantener, o bien realizar un proceso de migración a versiones más actuales.

En la rama del 2.6, durante su desarrollo se aceleraron de forma significativa los trabajos del *kernel*, ya que tanto Linus Torvalds, como Andrew Morton (que mantienen Linux 2.6) se incorporaron (durante 2003) a OSDL (Open Source Developement Laboratory) [OSDa], un consorcio de empresas con el fin de promocionar el uso Open Source y GNU/Linux en la empresa (en el consorcio se encuentran entre otras muchas empresas con intereses en GNU/Linux: HP, IBM, Sun, Intel, Fujitsu, Hitachi, Toshiba, Red Hat, Suse, Transmeta...). En esos momentos se dio una situación interesante, ya que el consorcio OSDL hizo de patrocinador de los trabajos, tanto al mantenedor de la versión estable del *kernel* (Andrew) como la de desarrollo (Linus), trabajando a tiempo completo en las versiones, y en los temas relacionados. Linus se mantiene independiente, trabajando en el *kernel*, mientras Andrew se fue a trabajar a Google, donde continuaba a tiempo completo sus desarrollos, realizando parches con diferentes aportaciones al *kernel*. Recientemente, OSDL se reconvirtió en la fundación The Linux Foundation.

Nota

The Linux Foundation: www.linux-foundation.org

Hay que tener en cuenta que con las versiones actuales del *kernel*, se ha alcanzado ya un alto grado de desarrollo y madurez, lo que hará que cada vez se amplíe más el tiempo entre la publicación de las versiones (no así de las revisiones parciales).

Además, otro factor a considerar es el tamaño y el número de personas que están trabajando en el desarrollo actual. En un principio había unas pocas personas que tenían un conocimiento global del *kernel* entero, mientras hoy en día tenemos muchas personas desarrollándolo, se cuenta que cerca de dos mil, con diferentes contribuciones, aunque el núcleo duro de desarrolladores se estima en unas pocas docenas.

Además, cabe tener en cuenta que la mayoría sólo tienen unos conocimientos parciales del *kernel*, y ni todos trabajan simultáneamente, ni su aportación es igual de relevante (algunas de ellas sólo corrigen errores sencillos); mientras que son unas pocas (como los mantenedores) las que disponen de un conocimiento total del *kernel*. Esto supone que se puedan alargar los desarrollos, y que las aportaciones se tengan que depurar, para ver que no entren en conflicto entre ellas, o decidir entre posibles alternativas de prestaciones para escoger.

Respecto a la numeración de las versiones del *kernel* de Linux ([lkm][DBo]), cabe tener en cuenta los aspectos siguientes:

a) Hasta la rama del *kernel* 2.6.x, las versiones del *kernel* Linux se regían por una división en dos series: una era la denominada "experimental" (la numerada impar en la segunda cifra, como 1.3.xx, 2.1.x o 2.5.x) y la otra es la de producción (serie par, como 1.2.xx, 2.0.xx, 2.2.x, 2.4.x y más). La serie expe-

rimental eran versiones que se movían rápidamente y se utilizaban para probar nuevas prestaciones, algoritmos o controladores de dispositivo, etc. Por la propia naturaleza de los *kernel*s experimentales, podían tener comportamientos impredecibles, como pérdidas de datos, bloqueos aleatorios de la máquina, etc. Por lo tanto, no debían utilizarse en máquinas destinadas a producción, a no ser que se quisiese probar una característica determinada (con los consecuentes peligros).

Los *kernel*s de producción (serie par) o estables eran *kernel*s con un conjunto de prestaciones bien definido, con un número bajo de errores conocidos y controladores de dispositivos probados. Se publicaban con menos frecuencia que los experimentales, y existían variedad de versiones, unas más buenas que otras. Los sistemas GNU/Linux se suelen basar en un determinado *kernel* estable elegido, no necesariamente el último *kernel* de producción publicado.

b) En la numeración actual del *kernel* Linux (utilizada en la rama 2.6.x), se siguen conservando algunos aspectos básicos: la versión viene indicada por unos números *X*.*Y*.*Z*, donde normalmente *X* es la versión principal, que representa los cambios importantes del núcleo; *Y* es la versión secundaria, y normalmente implica mejoras en las prestaciones del núcleo: *Y* es par en los núcleos estables e impar en los desarrollos o pruebas. Y *Z* es la versión de construcción, que indica el número de la revisión de *X*.*Y*, en cuanto a parches o correcciones hechas. Los distribuidores no suelen incluir la última versión del núcleo, sino la que ellos hayan probado con más frecuencia y puedan verificar que es estable para el software y componentes que ellos incluyen. Partiendo de este esquema de numeración clásico (que se siguió durante las ramas 2.4.x, hasta los inicios de la 2.6), tuvo algunas modificaciones para adaptarse al hecho de que el *kernel* (rama 2.6.x) se vuelve más estable (fijando X.Y a 2.6), y cada vez las revisiones son menores (por significar un salto de versión de los primeros números), pero el desarrollo continuo y frenético.

En los últimos esquemas, se llegan a introducir cuartos números, para especificar de Z cambios menores, o diferentes posibilidades de la revisión (con diferentes parches añadidos). La versión así definida con cuatro números es la que se considera estable (*stable)*. También son usados otros esquemas para las diversas versiones de test (normalmente no recomendables para entornos de producción), como sufijos -rc (*release candidate*), los -mm que son *kernel*s experimentales con pruebas de diferentes técnicas novedosas, o los -git que son una especie de "foto" diaria del desarrollo del *kernel*. Estos esquemas de numeración están en constante cambio para adaptarse a la forma de trabajar de la comunidad del *kernel*, y a sus necesidades para acelerar el desarrollo.

c) Para obtener el último *kernel* publicado, hay que acudir al archivo de *kernel*s Linux (en http://www.*kernel*.org) o al *mirror* local (en España http://www.es.*kernel*.org). También podrán encontrarse algunos parches al *kernel* original, que corrijan errores detectados a posteriori de la publicación del *kernel*.

Algunas de las características técnicas ([DBo][Arc]) del *kernel* Linux que podríamos destacar son:

- *Kernel* de tipo monolítico: básicamente es un gran programa creado como una unidad, pero conceptualmente dividido en varios componentes lógicos.

- Tiene soporte para carga/descarga de porciones del *kernel* bajo demanda, estas porciones se llaman módulos, y suelen ser características del *kernel* o controladores de dispositivo.

- *Threads* de *kernel*: para el funcionamiento interno se usan varios hilos (*threads*) de ejecución internos al *kernel*, que pueden estar asociados a un programa de usuario o bien a una funcionalidad interna del *kernel*. En Linux no se hacía un uso intensivo de este concepto. En las revisiones de la rama 2.6.x se ofreció mejor soporte, y gran parte del *kernel* se ejecuta usando diversos *threads* de ejecución.

- Soporte de aplicaciones *multithread*: soporte de aplicaciones de usuario de tipo *multithread*, ya que muchos paradigmas de computación de tipo cliente/servidor necesitan servidores capaces de atender múltiples peticiones simultáneas dedicando un hilo de ejecución a cada petición o grupo de ellas. Linux tiene una biblioteca propia de *threads* que puede usarse para las aplicaciones *multithread*, con las mejoras que se introdujeron en el *kernel*, también han permitido un mejor uso para implementar bibliotecas de *threads* para el desarrollo de aplicaciones.

- El *kernel* es de tipo no apropiativo (*nonpreemptive*): esto supone que dentro del *kernel* no pueden pararse llamadas a sistema (en modo supervisor) mientras se está resolviendo la tarea de sistema, y cuando ésta acaba, se reasume la ejecución de la tarea anterior. Por lo tanto, el *kernel* dentro de una llamada no puede ser interrumpido para atender otra tarea. Normalmente, los *kernel*s apropiativos están asociados a sistemas que trabajan en tiempo real, donde debe permitirse lo anterior para tratar eventos críticos. Hay algunas versiones especiales del *kernel* de Linux para tiempo real, que permiten esto por medio de la introducción de unos puntos fijos donde pueden intercambiarse. También se ha mejorado especialmente este concepto en la rama 2.6.x del *kernel*, permitiendo en algunos casos interrumpir algunas tareas del *kernel*, reasumibles, para tratar otras , resumiendo posteriormente su ejecución. Este concepto de *kernel* apropiativo también puede ser útil para mejorar tareas interactivas, ya que si se producen llamadas costosas al sistema, pueden provocar retardos en las aplicaciones interactivas.

- Soporte para multiprocesador, lo que llama multiprocesamiento simétrico (SMP). Este concepto suele englobar máquinas que van desde el caso simple de 2 hasta 64 CPU. Este tema se ha puesto de especial actualidad con

las arquitecturas de tipo *multicore,* permitiendo de 2 a 4 o más *cores* de CPU en máquinas accesibles a los usuarios domésticos. Linux puede usar múltiples procesadores, donde cada procesador puede manejar una o más tareas. Pero había algunas partes del *kernel* que disminuían el rendimiento, ya que están pensadas para una única CPU y obligan a parar el sistema entero en determinados bloqueos. SMP es una de las técnicas más estudiadas en la comunidad del *kernel* de Linux, y se han obtenido mejoras importantes en la rama 2.6. Ya del rendimiento SMP, depende en gran medida la adopción de Linux en los sistemas empresariales, en la faceta de sistema operativo para servidores.

- Sistemas de ficheros: el *kernel* tiene una buena arquitectura de los sistemas de ficheros, el trabajo interno se basa en una abstracción de un sistema virtual (VFS, *virtual file system*), que puede ser adaptada fácilmente a cualquier sistema real. Como resultado, Linux es quizás el operativo que más sistemas de ficheros soporta, desde el propio ext2, hasta msdos, vfat, ntfs, sistemas con *journal* como ext3, ReiserFS, JFS(IBM), XFS(Silicon), NTFS, iso9660 (CD), udf, etc. y se van añadiendo más en las diferentes revisiones.

Otras características menos técnicas (un poco de marketing):

a) Linux es gratis: junto con el software GNU, y el incluido en cualquier distribución, podemos tener un sistema UNIX completo prácticamente por el coste del hardware, y por la parte de los costes de distribución GNU/Linux, podemos obtenerla prácticamente gratis. Pero no está de más pagar por una distribución completa, con los manuales y apoyo técnico, a un coste menor comparado con lo que se paga por algunos sistemas propietarios, o contribuir con la compra al desarrollo de las distribuciones que más nos gusten, o nos sean prácticas.

b) Linux es personalizable: la licencia GPL nos permite leer y modificar el código fuente del *kernel* (siempre que tengamos los conocimientos adecuados).

c) Linux se ejecuta en hardware antiguo bastante limitado; es posible, por ejemplo, crear un servidor de red con un 386 con 4 MB de RAM (hay distribuciones especializadas en bajos recursos).

d) Linux es un sistema potente: el objetivo principal en Linux es la eficiencia, se intenta aprovechar al máximo el hardware disponible.

e) Alta calidad: los sistemas GNU/Linux son muy estables, bajo ratio de fallos, y reducen el tiempo dedicado a mantener los sistemas.

f) El *kernel* es bastante reducido y compacto: es posible colocarlo, junto con algunos programas fundamentales en un sólo disco de 1.44 MB (existen varias distribuciones de un sólo disquete con programas básicos).

g) Linux es compatible con una gran parte de los sistemas operativos, puede leer ficheros de prácticamente cualquier sistema de ficheros y puede comunicarse por red para ofrecer/recibir servicios de cualquiera de estos sistemas. Además, también con ciertas bibliotecas puede ejecutar programas de otros sistemas (como msdos, Windows, BSD, Xenix, etc.) en la arquitectura x86.

h) Linux está ampliamente soportado: no hay ningún otro sistema que tenga la rapidez y cantidad de parches y actualizaciones que Linux, ni en los sistemas propietarios. Para un problema determinado, hay infinidad de listas de correo y foros, que en pocas horas pueden permitir solucionar cualquier problema. El único problema está en los controladores de hardware reciente, que muchos fabricantes todavía se resisten a proporcionar si no es para sistemas propietarios. Pero esto está cambiando poco a poco, y varios de los fabricantes más importantes de sectores como tarjetas de vídeo (NVIDIA, ATI) e impresoras (Epson, HP,) comienzan ya a proporcionar los controladores para sus dispositivos.

2. Personalizar o actualizar el *kernel*

Como usuarios o administradores de sistemas GNU/Linux, tenemos que tener en cuenta las posibilidades que nos ofrece el *kernel* para adaptarlo a nuestras necesidades y equipos.

Normalmente, construimos nuestros sistemas GNU/Linux a partir de la instalación en nuestros equipos de alguna de las distribuciones de GNU/Linux, ya sean comerciales como Red Hat, Mandriva, Suse o "comunitarias" como Debian, Fedora.

Estas distribuciones aportan, en el momento de la instalación, una serie de *kernel*s Linux binarios ya preconfigurados y compilados, y normalmente tenemos que elegir qué *kernel* del conjunto de los disponibles se adapta mejor a nuestro hardware. Hay *kernel*s genéricos, o bien orientados a dispositivos IDE, otros a SCSI, otros que ofrecen una mezcla de controladores de dispositivos [AR01], etc.

Otra opción de instalación suele ser la versión del *kernel*. Normalmente las distribuciones usan una instalación que consideran lo suficientemente estable y probada como para que no cause problemas a los usuarios. Por ejemplo, a día de hoy muchas distribuciones vienen con versiones 2.6.x del *kernel* por defecto, ya que se considera la versión más estable del momento (en que salió la distribución). En algunos casos en el momento de la instalación puede ofrecerse la posibilidad de usar como alternativa, versiones más modernas, con mejor soporte para dispositivos más modernos (de última generación), pero quizás no tan probadas, en el momento que se publica la distribución.

Los distribuidores suelen, además, modificar el *kernel* para mejorar el comportamiento de su distribución o corregir errores que han detectado en el *kernel* en el momento de las pruebas. Otra técnica bastante común en las comerciales es deshabilitar prestaciones problemáticas, que pueden causar fallos a los usuarios, o necesitan una configuración especifica de la máquina, o bien una determinada prestación no se considera lo suficientemente estable para incluirla activada.

Nota

La posibilidad de actualizar y personalizar el *kernel* a medida ofrece buena adaptación a cualquier sistema, permitiendo una optimización y sintonización al mismo.

Esto nos lleva a considerar que, por muy bien que un distribuidor haga el trabajo de adaptar el *kernel* a su distribución, siempre nos podemos encontrar con una serie de problemas:

- El *kernel* no está actualizado en la última versión estable disponible; algunos dispositivos modernos no se soportan.
- El *kernel* estándar no dispone de soporte de los dispositivos que tenemos, porque no han estado habilitados.

- Los controladores que nos ofrece un fabricante necesitan una nueva versión del *kernel* o modificaciones.
- A la inversa, el *kernel* es demasiado moderno, tenemos hardware antiguo que ya no tiene soporte en los *kernel*s modernos.
- El *kernel*, tal como está, no obtiene las máximas prestaciones de nuestros dispositivos.
- Algunas aplicaciones que queremos usar requieren soporte de un *kernel* nuevo o de algunas de sus prestaciones.
- Queremos estar a la última, nos arriesgamos, instalando últimas versiones del *kernel* Linux.
- Nos gusta investigar o probar los nuevos avances del *kernel* o bien queremos tocar o modificar el *kernel*.
- Queremos programar un controlador para un dispositivo no soportado.
- ...

Por éstos y otros motivos podemos no estar contentos con el *kernel* que tenemos; se nos plantean entonces dos posibilidades: actualizar el *kernel* binario de la distribución o bien personalizarlo a partir de las fuentes.

Vamos a ver algunas cuestiones relacionadas con las diferentes opciones y qué suponen:

1) Actualización del *kernel* de la distribución: el distribuidor normalmente publica también las actualizaciones que van surgiendo del *kernel*. Cuando la comunidad Linux crea una nueva versión del *kernel*, cada distribuidor la une a su distribución y hace las pruebas pertinentes. Posteriormente al periodo de prueba, se identifican posibles errores, los corrige y produce la actualización del *kernel* pertinente respecto a la que ofrecía en los CD de la distribución. Los usuarios pueden descargar la nueva revisión de la distribución del sitio web, o bien actualizarla mediante algún sistema automático de paquetes vía repositorio de paquetes. Normalmente, se verifica qué versión tiene el sistema, se descarga el *kernel* nuevo y se hacen los cambios necesarios para que la siguiente vez el sistema funcione con el nuevo *kernel*, y se mantiene la versión antigua por si hay problemas.

> Este tipo de actualización nos simplifica mucho el proceso, pero no tiene por qué solucionar nuestros problemas, ya que puede ser que nuestro hardware no esté todavía soportado o la característica por probar del *kernel* no esté todavía en la versión que tenemos de la distribución; cabe recordar que no tiene por qué usar la última versión disponible (por ejemplo en *kernel*.org), sino aquella que el distribuidor considere estable para su distribución.

Si nuestro hardware tampoco viene habilitado por defecto en la nueva versión, estamos en la misma situación. O sencillamente, si queremos la última versión, este proceso no nos sirve.

2) Personalizar el *kernel* (proceso descrito con detalle en los apartados siguientes). En este caso, iremos a las fuentes del *kernel* y adaptaremos "a mano" el hardware o las características deseadas. Pasaremos por un proceso de configuración y compilación de las fuentes del *kernel* para, finalmente, crear un *kernel* binario que instalaremos en el sistema, y tenerlo, así, disponible en el siguiente arranque del sistema.

También aquí podemos encontrarnos con dos opciones más, o bien por defecto obtenemos la versión "oficial" del *kernel* (*kernel*.org), o bien podemos acudir a las fuentes proporcionadas por la propia distribución. Hay que tener en cuenta que distribuciones como Debian y Fedora hacen un trabajo importante de adecuación del *kernel*, y corrección de errores del *kernel* que afectan a su distribución, con lo cual podemos, en algunos casos, disponer de correcciones adicionales al código original del *kernel*. Otra vez más las fuentes ofrecidas por la distribución no tienen por qué corresponder a la última versión publicada.

Este sistema nos permite la máxima fiabilidad y control, pero a un coste de administración alto; ya que debemos disponer de conocimientos amplios de los dispositivos y de las características que estamos escogiendo (qué significan y qué implicaciones pueden tener), así como de las consecuencias que puedan tener las decisiones que tomemos.

3. Proceso de configuración y compilación

La personalización del *kernel* [Vasb] es un proceso costoso y necesita amplios conocimientos de lo que se está haciendo, y además, es una de las tareas críticas, de la cual depende la estabilidad del sistema, por la propia naturaleza del *kernel*, puesto que es el elemento central del sistema.

Nota

El proceso de obtención de un nuevo *kernel* personalizado pasa por obtener las fuentes, adaptar la configuración, compilar e instalar el kernel obtenido en el sistema.

Cualquier error de procedimiento puede comportar la inestabilidad o la pérdida del sistema. Por lo tanto, no está de más llevar a cabo cualquier tarea de *backup* de los datos de usuarios, datos de configuraciones que hayamos personalizado o, si disponemos de dispositivos adecuados, el *backup* completo del sistema. También es recomendable disponer de algún disquete de arranque (o distribución LiveCD con herramientas) que nos sirva de ayuda por si surgen problemas, o bien un disquete de rescate (*rescue disk*) que la mayoría de distribuciones permiten crear desde los CD de la distribución (o directamente proporcionan como CD de rescate para la distribución).

Sin ánimo de exagerar, casi nunca aparecen problemas si se siguen los pasos adecuadamente, se tiene conciencia de lo que se está haciendo y se toman algunas precauciones.

Vamos a ver el proceso necesario para instalar y configurar un *kernel* Linux. En los apartados siguientes, examinamos:

1) El caso de las versiones antiguas 2.4.x.

2) Algunas consideraciones sobre la migración a las 2.6.x

3) Detalles específicos de las versiones 2.6.x.

4) Un caso particular para la distribución Debian, que dispone de un sistema propio (*debian way*) de compilación más flexible.

Las versiones 2.4.x prácticamente ya no son ofrecidas por las distribuciones actuales, pero debemos considerar que en más de una ocasión nos veremos obligados a migrar un determinado sistema a nuevas versiones, o bien a mantenerlo en las antiguas, debido a incompatibilidades o existencia de hardware antiguo no soportado.

Los conceptos generales del proceso de compilación y configuración se explicarán en el primer apartado (2.4.x), ya que la mayoría de ellos son genéricos, y observaremos posteriormente las diferencias respecto las nuevas versiones.

3.1. Compilación *kernel* versiones 2.4.x

Las instrucciones son específicas para la arquitectura x86 Intel, mediante usuario *root* (aunque parte del proceso puede hacerse como usuario normal):

1) Obtener el *kernel*: por ejemplo, podemos acudir a www.kernel.org (o su servidor ftp) y descargar la versión que queramos probar. Hay *mirrors* para diferentes países, podemos, por ejemplo, acudir a www.es.kernel.org. En la mayoría de las distribuciones de GNU/Linux, como Fedora/Red Hat o Debian, también se ofrece como paquete el código fuente del *kernel* (normalmente con algunas modificaciones incluidas), si se trata de la versión del *kernel* que necesitamos, quizás sea preferible usar éstas (mediante los paquetes *kernel-source* o similares). Si queremos los últimos *kernel*s, quizás no estén disponibles en la distribución y tendremos que ir a *kernel.org*.

2) Desempaquetar el *kernel*: las fuentes del *kernel* solían colocarse y desempaquetarse sobre el directorio */usr/src*, aunque se recomienda utilizar algún directorio aparte para no mezclar con ficheros fuente que pueda traer la distribución. Por ejemplo, si las fuentes venían en un fichero comprimido de tipo bzip2:

```
bzip2 -dc linux-2.4.0.tar.bz2 | tar xvf -
```

Si las fuentes venían en un fichero gz, reemplazamos bzip2 por gzip. Al descomprimir las fuentes, se habrá generado un directorio linux-version_kernel, donde entraremos para establecer la configuración del *kernel*.

Antes de comenzar los pasos previos a la compilación, debemos asegurarnos de disponer de las herramientas correctas, en especial del compilador gcc, make y otras utilidades gnu complementarias en el proceso. Un ejemplo son las *modutils,* que disponen las diferentes utilidades para el uso y gestión de los módulos de *kernel* dinámicos. Asimismo, para las diferentes opciones de configuración hay que tener en cuenta una serie de prerrequisitos en forma de librerías asociadas a la interfaz de configuración usada (por ejemplo las ncurses para la interfaz menuconfig).

Se recomienda, en general, consultar la documentación del *kernel* (ya sea vía paquete, o en el directorio raíz de las fuentes) para conocer qué prerrequisitos, así como versiones de éstos, son necesarios para el proceso. Se recomienda examinar los ficheros *README* en el directorio "raíz", y el Documentation/Changes, o el índice de documentación del *kernel* en Documentation/00-INDEX.

Si hemos realizado anteriores compilaciones en el mismo directorio, deberemos asegurar que el directorio utilizado esté limpio de compilaciones anteriores; podemos limpiarlo con *make mrproper* (realizado desde el directorio "raíz").

Para el proceso de configuración del *kernel* [Vasb], tenemos varios métodos alternativos, que nos presentan interfaces diferentes para ajustar los múltiples

parámetros del *kernel* (que suelen almacenarse en un fichero de configuración, normalmente *.config* en el directorio "raíz" de las fuentes). Las diferentes alternativas son:

- **make config**: desde la línea de comandos se nos pregunta por cada opción, y se nos pide confirmación (y/n) –si deseamos o no, la opción o se nos piden los valores necesarios. O bien la configuración larga, en la que se nos piden muchas respuestas, y dependiendo de la versión, igual tenemos que responder a casi un centenar de preguntas (o más dependiendo de la versión).

- **make oldconfig**: sirve por si queremos reutilizar una configuración ya usada (normalmente almacenada en un fichero *.config*, en el directorio "raíz" de las fuentes), hay que tener en cuenta que sólo es válida si estamos compilando la misma versión del *kernel*, ya que diferentes versiones del *kernel* pueden variar en sus opciones.

- **make menuconfig**: configuración basada en menús textuales, bastante cómoda; podemos habilitar o inhabilitar lo que queramos y es más rápida que el *make config*.

- **make xconfig**: la más cómoda, basada en diálogos gráficos en X Window. Es necesario tener instaladas las bibliotecas de tcl/tk, ya que esta configuración está programada en este lenguaje. La configuración se basa en cuadros de diálogos y botones / casillas de activación, se hace bastante rápida y dispone de ayuda con comentarios de muchas de las opciones. Pero hay un defecto, puede ser que algunas de las opciones no aparezcan (depende de que el programa de configuración esté actualizado, y a veces no lo está). En este último caso, el *make config* (o el *menuconfig*) es el único que asegura disponer de todas las opciones elegibles; en los otros tipos de configuración depende de que se hayan podido adaptar a tiempo los programas a las nuevas opciones cuando se libera el *kernel*. Aunque en general se intentan mantener de forma equivalente.

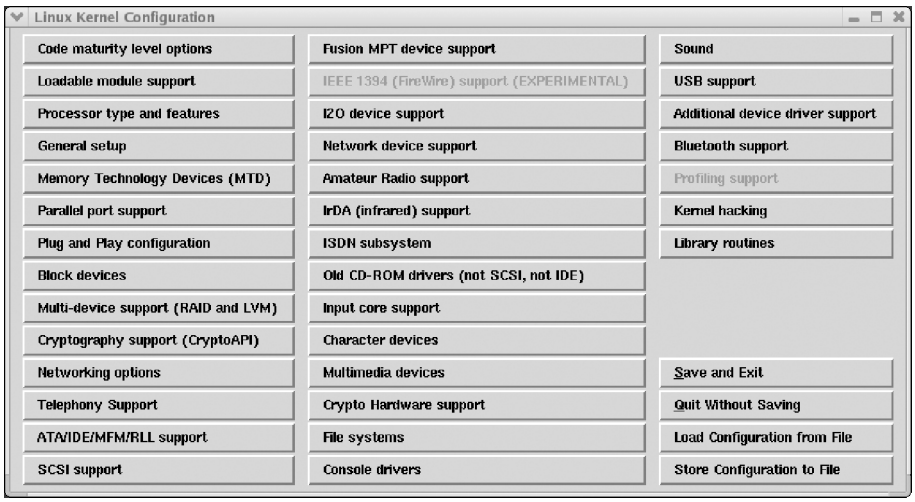

Figura 2. Configuración del *kernel* (make xconfig) desde interfaz gráfica en X Window

Una vez se haya hecho el proceso de configuración, hay que guardar el fichero (*.config),* ya que la configuración consume un tiempo importante. Además, puede ser de utilidad disponer de la configuración realizada si está planeado hacerla en varias máquinas parecidas o idénticas.

Otro tema importante en las opciones de configuración es que en muchos casos se nos va a preguntar por si una determinada característica la queremos integrada en el *kernel* o como módulo (en el apartado dedicado a los módulos daremos más detalles sobre los mismos). Ésta es una decisión más o menos importante, ya que el rendimiento del *kernel* (y por lo tanto, del sistema entero) en algunos casos puede depender de nuestra elección.

El *kernel* de Linux ha comenzado a ser de gran tamaño, tanto por complejidad como por los controladores (*drivers*) de dispositivo [AR01] que incluye. Si lo integrásemos todo, se podría crear un fichero del *kernel* bastante grande y ocupar mucha memoria, ralentizando, de ese modo, algunos aspectos de funcionamiento. Los módulos del *kernel* [Hen] son un método que permite separar parte del *kernel* en pequeños trozos, que serán cargados bajo demanda dinámicamente, cuando o bien por carga explícita o por uso de la característica sean necesarios.

La elección más normal es integrar lo que se considere básico para el funcionamiento o crítico en rendimiento dentro del *kernel*. Y aquellas partes o controladores de los que se vaya a hacer un uso esporádico o que se necesite dejarlos como módulos para futuras ampliaciones del equipo.

- Un caso claro son los controladores de dispositivo: si estamos actualizando la máquina, puede ser que a la hora de crear el *kernel* no conozcamos con seguridad qué hardware va a tener: por ejemplo, qué tarjeta de red; pero sí que sabemos que estará conectada a red, entonces, el soporte de red estará integrado en el *kernel*, pero en los controladores de las tarjetas podremos seleccionar unos cuantos (o todos) y ponerlos como módulos. Así, cuando tengamos la tarjeta podremos cargar el módulo necesario, o si después tenemos que cambiar una tarjeta por otra, sólo tendremos que cambiar el módulo que se va a cargar. Si hubiese sólo un controlador integrado en el *kernel* y cambiamos la tarjeta, esto obligaría a reconfigurar y recompilar el *kernel* con el controlador de la tarjeta nueva.

- Otro caso que suele aparecer (aunque no es muy común) es cuando necesitamos dos dispositivos que son incompatibles entre sí, o está funcionando uno o el otro (esto puede pasar, por ejemplo, con la impresora con cable paralelo y hardware que se conecta al puerto paralelo). Por lo tanto, en este caso tenemos que colocar como módulos los controladores y cargar o descargar el que sea necesario.

- Otro ejemplo podrían conformarlo los sistemas de ficheros (*filesystems*) normalmente esperaremos que nuestro sistema tenga acceso a algunos de

ellos, por ejemplo ext2 o ext3 (propios de Linux), vfat (de los Windows 95/98/ME), y los daremos de alta en la configuración del *kernel*. Si en otro momento tuviéramos que leer otro tipo no esperado, por ejemplo, datos guardados en un disco o partición de sistema NTFS de Windows NT/XP, no podríamos: el *kernel* no sabría o no tendría soporte para hacerlo. Si tenemos previsto que en algún momento (pero no habitualmente) se tenga que acceder a estos sistemas, podemos dejar los demás *filesystems* como módulos.

1) Compilación del *kernel*

Mediante el *make* comenzaremos la compilación, primero hay que generar las posibles dependencias entre el código, y luego el tipo de imagen de *kernel* que se quiere (en este caso una imagen comprimida, que suele ser la normal):

```
make dep
make bzImage
```

Cuando este proceso acabe, tenemos la parte integrada del *kernel*; nos faltan las partes que hayamos puesto como módulos:

```
make modules
```

Hasta este momento hemos hecho la configuración y compilación del *kernel*. Esta parte podía hacerse desde un usuario normal o bien desde el *root*, pero ahora necesitaremos forzosamente un usuario *root*, porque pasaremos a la parte de la instalación.

2) Instalación

Comenzamos instalando los módulos:

```
make modules_install
```

y la instalación del nuevo *kernel* (desde el directorio /usr/src/linux-version o el que hayamos utilizado como temporal):

```
cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.0
cp System.map /boot/System.map-2.4.0
```

el archivo bzImage es el *kernel* recién compilado, que se coloca en el directorio /boot. Normalmente, el *kernel* antiguo se encontrará en el mismo directorio /boot con el nombre vmlinuz, o bien vmlinuz-version-anterior y vmlinuz como un enlace simbólico al *kernel* viejo. Una vez tengamos nuestro *kernel*, es mejor conservar el antiguo, por si se producen fallos o mal funcionamiento del nuevo, poder recuperar el viejo. El fichero System.map contiene los símbolos disponibles en el *kernel* y es necesario para el proceso de arranque del mismo; también se coloca en el mismo directorio.

En este punto también hay que tener en cuenta que en el arranque el *kernel* puede necesitar la creación de ficheros tipo initrd, que sirve de imagen compuesta de algunos *drivers* básicos y se utiliza durante la carga del sistema, como primera fase, si el sistema necesita de esos *drivers* antes de proceder al arranque de algunos componentes. En algunos casos es imprescindible, debido a que, para poder cargar el resto del sistema, son necesarios controladores que deberán cargarse en una primera fase; un ejemplo de ello serían controladores específicos de disco como controladores RAID o de volúmenes, que serían necesarios para después, en una segunda fase, acceder al disco para cargar el resto del sistema.

El *kernel* puede generarse con o sin imagen initrd, dependerá de las necesidades del hardware o sistema concreto. En algunos casos la distribución impone la necesidad de usar imagen initrd, en otros casos dependerá de nuestro hardware. También se suele utilizar para vigilar el tamaño del *kernel*, de manera que puede cargarse lo básico de éste a través de la imagen initrd, y posteriormente cargar el resto en forma de módulos en una segunda fase. En el caso de necesitar la imagen initrd, se crearía con la utilidad mkinitrd (ver man, o taller del capítulo), dentro del directorio /boot.

3) El siguiente paso es decirle al sistema con qué *kernel* tiene que arrancar, aunque esto depende del sistema de arranque de Linux:

a) Desde arranque con lilo [Zan][Skoa], ya sea en el MBR (*master boot record*) o desde partición propia, hay que añadir al fichero de configuración (en: /etc/lilo.conf), por ejemplo las líneas:

```
image = /boot/vmlinuz-2.4.0
  label = 2.4.0
```

donde *image* es el *kernel* que se va arrancar, y *label* será el nombre con el que aparecerá la opción en el arranque. Podemos añadir estas líneas o modificar las que hubiera del *kernel* antiguo. Se recomienda añadirlas y dejar el *kernel* antiguo, por si aparecen problemas, poder recuperar el antiguo. En el fichero /etc/lilo.conf puede haber una o más configuraciones de arranque, tanto de Linux como de otros sistemas (como Windows).

Cada arranque se identifica por su línea *image* y el *label* que aparece en el menú de arranque. Hay una línea *default* = *label* donde se indica el *label* que se arranca por defecto. También podemos añadirle a las líneas anteriores un root = /dev/... para indicar la partición de disco donde está el sistema de archivos principal (el '/'), recordar que los discos tienen dispositivos como /dev/hda (1.r disco ide) /dev/hdb (2.º disco ide) o bien /dev/sdx para discos SCSI (o emulados), y la partición se indicaría como *root = /dev/hda2* si el '/' de nuestro Linux estuviese en la segunda partición del primer disco ide. Además con "*append =*" podemos añadir parámetros al arranque del *kernel*

[Gor]. Si el sistema utiliza, initrd, también habrá que indicar cuál es el fichero (que también estará localizado en /boot/initrd-versiónkernel), con la opción "initrd=". Después de cambiar la configuración del lilo, hay que escribirla para que arranque:

```
/sbin/lilo -v
```

Reiniciamos (*reboot*) y arrancamos con el nuevo *kernel*.

Si tuviésemos problemas, podemos recuperar el antiguo *kernel*, escogiendo la opción del viejo *kernel*, y luego, mediante la acción de retocar el lilo.conf, podemos devolver la antigua configuración o estudiar el problema y reconfigurar y recompilar el *kernel* de nuevo.

b) Arranque con grub [Kan01][Pro]. El manejo en este caso es bastante simple, cabe añadir una nueva configuración formada por el *kernel* nuevo y sumarla como una opción más al fichero del grub. A continuación, reiniciar procediendo de forma parecida a la del lilo, pero recordando que en grub es suficiente con editar el fichero (típicamente /boot/grub/menu.lst) y reiniciar. También es mejor dejar la antigua configuración para poder recuperarse de posibles errores.

3.2. Migración a *kernel* 2.6.x

En el caso de tener que actualizar versiones de distribuciones antiguas, o bien realizar el cambio de generación del *kernel* mediante las fuentes, habrá que tomar una serie de consideraciones que tomar en cuenta, debido a las novedades introducidas en la rama 2.6.x del *kernel*.

Listamos algunos puntos concretos que cabe observar:

- Algunos de los módulos del *kernel* han cambiado de nombre, y algunos han podido desaparecer, hay que comprobar la situación de los módulos dinámicos que se cargan (por ejemplo, examinar /etc/modules y/o /etc/modules.conf), y editarlos para reflejar los cambios.

- Se han añadido nuevas opciones para la configuración inicial del *kernel*:como make gconfig, una interfaz basada en gtk (Gnome). Habrá que observar, como prerrequisito en este caso, las librerías de Gnome. La opción de make xconfig se ha implementado ahora con las librerías de qt (KDE).

- Se incrementan las versiones mínimas necesarias de varias utilidades necesarias para el proceso de compilación (consultar Documentation/Changes en las fuentes del *kernel*). En especial la versión del compilador gcc mínima.

- Ha cambiado el paquete por defecto para las utilidades de módulos, pasa a ser module-init-tools (en lugar de modutils usado en 2.4.x). Este paquete es un prerrequisito para la compilación de *kernel*s 2.6.x, ya que el cargador de módulos está basado en esta nueva versión.

- El sistema devfs, queda obsoleto en favor de udev, el sistema que controla el arranque (conexión) en caliente (*hotplug*) de dispositivos (y su reconocimiento inicial, de hecho simulando un arranque en caliente al iniciar el sistema), creando dinámicamente las entradas en el directorio /dev, sólo para los dispositivos que estén actualmente presentes.

- En Debian a partir de ciertas versiones del 2.6.x, para las imágenes binarias de *kernel*s, *headers* y código fuente cambia el nombre de los paquetes de kernel-images/source/headers a linux-image/source/headers.

- En algunos casos, los dispositivos de tecnologías nuevas (como SATA) pueden haber pasado de /dev/hdX a /dev/sdX. En estos casos habrá que editar las configuraciones de /etc/fstab y el *bootloader (*lilo o grub*)* para reflejar los cambios.

- Pueden existir algunos problemas con dispositivos de entrada/salida concretos. El cambio de nombres de módulos de *kernel* ha afectado entre otros a los dispositivos de ratón, lo que puede afectar asimismo a la ejecución de X-Window, a la verificación de qué modelos son necesarios y cargar los módulos correctos (por ejemplo el *psmouse*). Por otra parte, en el *kernel*, se integran los *drivers* de sonido Alsa. Si disponemos de los antiguos OSS, habrá que eliminarlos de la carga de módulos, ya que Alsa ya se encarga de la emulación de estos últimos.

- Respecto a las arquitecturas que soporta el *kernel*, hay que tener en cuenta que con los *kernel* 2.6.x, en las diferentes revisiones, se han ido incrementando las arquitecturas soportadas, lo que nos permitirá disponer de imágenes binarias del *kernel* en las distribuciones (o las opciones de compilación del *kernel*) más adecuadas para el soporte de nuestros procesadores. En concreto podemos encontrarnos con arquitecturas como i386 (para intel y AMD): soportando la compatibilidad de Intel en 32 bits para toda la familia de procesadores (en algunas distribuciones es usada 486 como arquitectura general), en algunas distribuciones se integran versiones diferenciadas para i686 (Intel a partir de pentium pro en adelante), para k7 (AMD Athlon en adelante), y las específicas de 64 bits, para AMD 64 bits, e Intel con extensiones em64t de 64 bits como los Xeon, y Multicores. Por otra parte, también existe la arquitectura IA64 para los modelos 64bits Intel Itanium. En la mayoría de los casos, las arquitecturas disponen de las capacidades SMP activadas en la imagen del *kernel* (a no ser que la distribución soporte versiones con SMP o sin, creadas independientemente, en este caso, suele añadirse el sufijo -smp a la imagen que lo soporta).

- En Debian, para la generación de imágenes inirtrd, a partir de ciertas versiones del *kernel* (>=2.6.12) se consideran obsoletas las mkinitrd-tools, que son substituidas por nuevas utilidades como initramfs-tools o yaird. Ambas permiten la construcción de la imagen initrd, siendo la primera la recomendada (por Debian).

3.3. Compilación de *kernel* versiones 2.6.x

En las versiones 2.6.x, teniendo en cuenta las consideraciones comentadas anteriormente, la compilación se desarrolla de forma semejante a la expuesta anteriormente:

Una vez descargado el *kernel* 2.6.x (sea x el número de revisión del *kernel*, o el par de números) en el directorio que se usará para la compilación, y comprobadas las versiones necesarias de las utilidades básicas, se procede al proceso de compilación, y limpieza de anteriores compilaciones:

```
# make clean mrproper
```

configuración de parámetros (recordar que si disponemos de un .config previo, nos permitirá no comenzar de cero la configuración). La configuración la realizamos a través de la opción escogida de *make* (dependiendo de la interfaz usada):

```
# make menuconfig
```

construcción de la imagen binaria del *kernel*

```
# make dep
# make bzImage
```

construcción de los módulos (aquellos especificados como tales):

```
# make modules
```

instalación de los módulos creados (/lib/modules/version)

```
# make modules_install
```

copia de la imagen a su posición final (suponiendo i386 como arquitectura):

```
# cp arch/i386/boot/bzimage /boot/vmlinuz-2.6.x.img
```

y finalmente, creación de la imagen initrd que consideremos necesaria, con las utilidades necesarias según la versión (ver comentario posterior). Y ajuste del *bootloader* lilo o grub según sea el utilizado.

Los últimos pasos (vmlinuz, system.map y initrd) de movimiento de archivos a /boot pueden realizarse también normalmente con el proceso:

```
# make install
```

pero hay que tener en cuenta que realiza todo el proceso, y actualizará los *bootloaders,* quitando antiguas configuraciones o alterándolas; asimismo, pueden alterarse los enlaces por defecto en el directorio /boot. Hay que tenerlo en cuenta a la hora de pensar en las configuraciones pasadas que queremos guardar.

Respecto a la creación del initrd, en Fedora/Red Hat éste se creara automáticamente con la opción "install". En Debian deberemos, o bien utilizar las técnicas del siguiente apartado, o bien crearlo expresamente con mkinitrd (versiones <=2.6.12) o, posteriormente, con mkinitramfs, o una utilidad denominada update-initramfs, especificando la versión del *kernel* (se asume que éste se llama vmlinuz-version dentro del directorio /boot):

```
# update-initramfs -c -k 'version'
```

3.4. Compilación en Debian del *kernel* (Debian way)

En Debian, además de los métodos examinados hay que añadir la configuración por el método denominado Debian Way. Un método que nos permite construir el *kernel* de una forma flexible y rápida.

Para el proceso nos serán necesarias una serie de utilidades (instalar los paquetes, o similares): kernel-package, ncurses-dev, fakeroot, wget, bzip2.

Podemos observar el método desde dos perspectivas, reconstruir un *kernel* equivalente al proporcionado por la distribución, o bien personalizarlo y utilizar así el método para construir un *kernel* equivalente personalizado.

En el primer caso, pasamos por obtener la versión de las fuentes del *kernel* que nos proporciona la distribución (sea x la revisión del *kernel* 2.6):

Nota

Puede verse el proceso "Debian way" de forma detallada en: http://kernel-handbook.alioth.debian.org/

```
# apt-get install linux-source-2.6.x
$ tar -xvjf /usr/src/linux-source-2.6.x.tar.bz2
```

donde obtenemos las fuentes y las descomprimimos (el paquete deja el archivo en /usr/src).

Instalación de herramientas básicas:

```
# apt-get install build-essential fakeroot
```

Comprobar dependencias de las fuentes

```
# apt-get build-dep linux-source-2.6.x
```

Y construcción del binario, según configuración preestablecida del paquete (semejante a la incluida en los paquetes *image* oficiales del *kernel* en Debian):

```
$ cd linux-source-2.6.x
$ fakeroot debian/rules binary
```

Existen algunos procedimientos extra para la creación de *kernel*s en base a diferentes niveles de *patch* proporcionados por la distribución, y posibilidades de generar diferentes configuraciones finales (puede verse la referencia de la nota para complementar estos aspectos).

En el segundo caso, más habitual, cuando deseamos un *kernel* personalizado, deberemos realizar un proceso semejante a través de un paso de personalización típico (por ejemplo, mediante make menuconfig), los pasos:

Obtención y preparación del directorio (aquí obtenemos los paquetes de la distribución, pero es equivalente obteniendo las fuentes desde *kernel.org*):

```
# apt-get install linux-source-2.6.x
$ tar xjf /usr/src/linux-source-2.6.x.tar.bz2
$ cd linux-source-2.6.x
```

a continuación realizamos la configuración de parámetros, como siempre podemos basarnos en ficheros .config que hayamos utilizado anteriormente, para partir de una configuración conocida (para la personalización, también puede usarse cualquiera de los otros métodos, xconfig, gconfig...):

```
$ make menuconfig
```

construcción final del *kernel* dependiendo de initrd o no, sin initrd disponible (hay que tener cuidado con la versión utilizada; a partir de cierta versión del *kernel*, puede ser obligatorio el uso de imagen initrd):

```
$ make-kpkg clean
$ fakeroot make-kpkg --revision=custom.1.0 kernel_image
```

o bien si disponemos de initrd disponible (construido ya)

```
$ make-kpkg clean
$ fakeroot make-kpkg --initrd --revision=custom.1.0 kernel_image
```

El proceso finalizará con la obtención del paquete asociado a la imagen del *kernel*, que podremos finalmente instalar:

```
# dpkg -i ../linux-image-2.6.x_custom.1.0_i386.deb
```

Añadimos, también en este apartado, otra peculiaridad a tener en cuenta en Debian, que es la existencia de utilidades para añadir módulos dinámicos de *kernel* proporcionados por terceros. En particular la utilidad module-assistant permite automatizar todo este proceso a partir de las fuentes del módulo.

Necesitamos disponer de los *headers* del *kernel* instalado (paquete linux-headers-version) o bien de las fuentes que utilizamos en la compilación del *kernel*. A partir de aquí module-assistant puede utilizarse interactivamente, permitiendo seleccionar entre una amplia lista de módulos registrados previamente en la aplicación, y puede encargarse de descargar el módulo, compilarlo e instalarlo en el *kernel* existente.

También en la utilización desde línea de comandos, podemos simplemente especificar (m-a es equivalente a module-assistant):

```
# m-a prepare
# m-a auto-install nombre_modulo
```

Lo cual prepara el sistema para posibles dependencias, descarga fuentes del módulo, compila y, si no hay problemas, instala para el presente *kernel*. El nombre del módulo podemos observarlo de la lista interactiva de module-assistant.

4. Parchear el *kernel*

En algunos casos también puede ser habitual la aplicación de parches (*patch)* al *kernel* [lkm].

> Un fichero de parche (*patch file*) respecto al *kernel* de Linux es un fichero de texto ASCII que contiene las diferencias entre el código fuente original y el nuevo código, con información adicional de nombres de fichero y líneas de código. El programa *patch (*ver *man patch*) sirve para aplicarlo al árbol del código fuente del *kernel* (normalmente en /usr/src).

Los parches suelen necesitarse cuando un hardware especial necesita alguna modificación en el *kernel*, o se han detectado algunos *bugs* (errores) posteriores a alguna distribución amplia de una versión del *kernel*, o bien quiere añadirse una nueva prestación concreta. Para corregir el problema (o añadir la nueva prestación), se suele distribuir un parche en lugar de un nuevo *kernel* entero. Cuando ya existen varios de estos parches, se unen con diversas mejoras del *kernel* anterior para formar una nueva versión del *kernel*. En todo caso, si tenemos el hardware problemático, o el error afecta a la funcionalidad o a la estabilidad del sistema y no podemos esperar a la siguiente versión del *kernel*, será necesario aplicar el parche.

El parche se suele distribuir en un fichero comprimido tipo bz2 (bunzip2, aunque también puede encontrarse en gzip con extensión .gz), como por ejemplo podría ser:

```
patchxxxx-2.6.21-pversion.bz2
```

donde xxxx suele ser algún mensaje sobre el tipo o finalidad del parche. 2.6.21 sería la versión del *kernel* al cual se le va a aplicar el parche, y pversion haría referencia a la versión del parche, del que también pueden existir varias. Hay que tener en cuenta que estamos hablando de aplicar parches a las fuentes del *kernel* (normalmente instaladas, como vimos, en /usr/src/linux o directorio similar).

Una vez dispongamos del parche, tendremos que aplicarlo, veremos el proceso a seguir en algún fichero readme que acompañe al parche, pero generalmente el proceso sigue los pasos (una vez comprobados los requisitos previos) de descomprimir el parche en el directorio de los ficheros fuente y aplicarlo sobre las fuentes del *kernel*, como por ejemplo:

```
cd /usr/src/linux (o /usr/src/linux-2.6.21 o la versión que sea).
```

```
bunzip2 patch-xxxxx-2.6.21-version.bz2
patch -p1 < patch-xxxxx-2.6.21-version
```

y posteriormente, tendremos que recompilar el *kernel* para volverlo a generar.

Los parches pueden obtenerse de diferentes lugares. Lo más normal es encontrarlos en el sitio de almacén de los *kernel*s (www.kernel.org) o bien en www.linuxhq.com, que tiene un archivo completo de ellos. En determinadas comunidades Linux (o usuarios individuales) también suelen ofrecer algunas correcciones, pero es mejor buscar en los sitios estándar para asegurar un mínimo de confianza en estos parches, y evitar problemas de seguridad con posibles parches "piratas". Otra vía es el fabricante de hardware que puede ofrecer ciertas modificaciones del *kernel* (o de controladores) para que funcionen mejor sus dispositivos (un ejemplo conocido es NVIDIA y sus *drivers* Linux para sus tarjetas gráficas).

Por último, señalaremos que en las distribuciones de GNU/Linux, muchas de ellas (Fedora/Red Hat, Mandriva...) ya ofrecen *kernel*s parcheados por ellos mismos, y sistemas para actualizarlos (algunos incluso de forma automática, como en el caso de Fedora/Red Hat y Debian). Normalmente, en sistemas de producción es más recomendable seguir las actualizaciones del fabricante, aunque éste no ofrecerá necesariamente el último *kernel* publicado, sino el que crea más estable para su distribución, bajo pena de perder prestaciones de última generación, o alguna novedad en las técnicas incluidas en el *kernel*.

Nota

En sistemas que se quieran tener actualizados, por razones de test o de necesidad de las últimas prestaciones, siempre se puede acudir a www.kernel.org y obtener el kernel más moderno publicado.

5. Los módulos del *kernel*

El *kernel* es capaz de cargar dinámicamente porciones de código (módulos) bajo demanda [Hen], para complementar su funcionalidad (se dispone de esta posibilidad desde la versión 1.2 del *kernel*). Por ejemplo, los módulos pueden añadir soporte para un sistema de ficheros o para dispositivos hardware específicos. Cuando la funcionalidad proporcionada por el módulo no es necesaria, el módulo puede ser descargado, liberando memoria.

Normalmente, bajo demanda, el *kernel* identifica una característica no presente en el *kernel* en ese momento, contacta con un *thread* del *kernel* denominado kmod (en las versiones *kernel* 2.0.x el daemon era llamado kerneld*),* éste ejecuta un comando modprobe para intentar cargar el módulo asociado, a partir o de una cadena con el nombre de módulo, o bien de un identificador genérico; esta información se consulta en el fichero /etc/modules.conf en forma de alias entre el nombre y el identificador.

A continuación se busca en

```
/lib/modules/version_kernel/modules.dep
```

para saber si hay dependencias con otros módulos. Finalmente, con el comando insmod se carga el módulo desde /lib/modules/version_kernel/ (el directorio estándar para los módulos), la version_kernel es la versión del *kernel* actual, se utiliza el comando uname -r para determinarla. Por lo tanto, los módulos en forma binaria están relacionados con una versión concreta del *kernel*, y suelen colocarse en /lib/modules/version-kernel.

Si hay que compilarlos, se tiene que disponer de las fuentes y/o *headers* de la versión del núcleo al cual está destinado.

Nota

Los módulos aportan una flexibilidad importante al sistema, permitiendo que se adapte a situaciones dinámicas.

Hay unas cuantas utilidades que nos permiten trabajar con módulos (suelen aparecer en un paquete software llamado modutils*,* que se reemplazó por module-init-tools para la gestión de módulos de la rama 2.6.x):

- **lsmod:** podemos ver los módulos cargados en el *kernel* (la información se obtiene del pseudofichero /proc/modules). Se listan los nombres, las dependencias con otros (en [ ]), el tamaño del módulo en bytes, y el contador de uso del módulo; esto permite descargarlo si la cuenta es cero.

  Ejemplo

  Algunos módulos en una Debian:

  ```
  Module        Size      Used by     Tainted: P
  agpgart       37.344          3     (autoclean)
  ```

```
apm             10.024          1       (autoclean)
parport_pc      23.304          1       (autoclean)
lp              6.816           0       (autoclean)
parport         25.992          1       [parport_pc lp]
snd             30.884          0
af_packet       13.448          1       (autoclean)
NVIDIA          1.539.872       10
es1371          27.116          1
soundcore       3.972           4       [snd es1371]
ac97_codec      10.9640         0       [es1371]
gameport        1.676           0       [es1371]
3c59x           26.960          1
```

b) modprobe: intenta la carga de un módulo y de sus dependencias.

c) insmod: carga un módulo determinado.

d) depmod: analiza dependencias entre módulos y crea fichero de dependencias.

e) rmmod: saca un módulo del *kernel*.

f) Otros comandos pueden ser utilizados para depuración o análisis de los módulos, como modinfo*:* lista algunas informaciones asociadas al módulo, o ksyms, (sólo en versiones 2.4.x) permite examinar los símbolos exportados por los módulos (también en /proc/ksyms).

Ya sea por el mismo *kernel*, o por el usuario manualmente con insmod, normalmente para la carga se especificará el nombre del módulo, y opcionalmente determinados parámetros. Por ejemplo, en el caso de que sean dispositivos, suele ser habitual especificar las direcciones de los puertos de E/S o bien los recursos de IRQ o DMA. Por ejemplo:

```
insmod soundx io = 0x320 irq = 5
```

6. Futuro del *kernel* y alternativas

Los avances en el *kernel* de Linux en determinados momentos fueron muy rápidos, pero actualmente, ya con una situación bastante estable con los *kernel*s de la serie 2.6.x, cada vez pasa más tiempo entre las versiones que van apareciendo, y en cierta manera esto es bastante positivo. Permite tener tiempo para corregir errores cometidos, ver aquellas ideas que no funcionaron bien y probar nuevas ideas, que, si resultan, se incluyen.

Comentaremos en este apartado algunas de las ideas de los últimos *kernel*s, y algunas que están previstas, para dar indicaciones de lo que será el futuro próximo en el desarrollo del *kernel*.

Nota

El *kernel* continúa evolucionando, incorporando las últimas novedades en soporte hardware y mejoras en las prestaciones.

En la anterior serie, serie 2.4.x [DBo], incluida en la mayoría de distribuciones actuales, se realizaron algunas aportaciones en:

- Cumplimiento de los estándares IEEE POSIX, esto permite que muchos de los programas existentes de UNIX pueden recompilarse y ejecutarse en Linux.
- Mejor soporte de dispositivos: PnP, USB, Puerto Paralelo, SCSI...
- Soporte para nuevos *filesystems*, como UDF (CD-ROM reescribibles como un disco). Otros sistemas con *journal*, como los Reiser de IBM o el ext3, éstos permiten tener un *log* (*journal*) de las modificaciones de los sistemas de ficheros, y así poder recuperarse de errores o tratamientos incorrectos de los ficheros.
- Soporte de memoria hasta 4 GB, en su día surgieron algunos problemas (con *kernel*s 1.2.x) que no soportaban más de 128 MB de memoria (en aquel tiempo era mucha memoria).
- Se mejoró la interfaz /proc. Éste es un pseudosistema de ficheros (el directorio /proc) que no existe realmente en disco, sino que es simplemente una forma de acceder a datos del *kernel* y del hardware de una manera organizada.
- Soporte del sonido en el *kernel*, se añadieron parcialmente los controladores Alsa que antes se configuraban por separado.
- Se incluyó soporte preliminar para el RAID software, y el gestor de volúmenes dinámicos LVM1.

En serie actual, la rama del *kernel* 2.6.x [Pra], dispuso de importantes avances respecto a la anterior (con las diferentes revisiones .x de la rama 2.6):

- Mejores prestaciones en SMP, importante para sistemas multiprocesadores muy utilizados en entornos empresariales y científicos.

- Mejoras en el planificador de CPU (*scheduler*).

- Mejoras en el soporte multithread para las aplicaciones de usuario. Se incorporan nuevos modelos de *threads* NGPT (IBM) y NPTL (Red Hat) (con el tiempo se consolidó finalmente la NPTL).

- Soporte para USB 2.0.

- Controladores Alsa de sonido incorporados en el *kernel*.

- Nuevas arquitecturas de CPU de 64 bits, se soportan AMD x86_64 (también conocida como amd64) y PowerPC 64, y IA64 (arquitectura de los Intel Itanium).

- Sistemas de ficheros con *journal*: JFS, JFS2 (IBM), y XFS (Silicon Graphics).

- Mejoras de prestaciones en E/S, y nuevos modelos de controladores unificados.

- Mejoras en implementación TCP/IP, y sistema NFSv4 (compartición sistema de ficheros por red con otros sistemas).

- Mejoras significativas para *kernel* apropiativo: permite que internamente el *kernel* gestione varias tareas que se pueden interrumpir entre ellas, imprescindible para implementar eficazmente sistemas de tiempo real.

- Suspensión del sistema y restauración después de reiniciar (por *kernel*).

- UML, User Mode Linux, una especie de máquina virtual de Linux sobre Linux que permite ver un Linux (en modo usuario) ejecutándose sobre una máquina virtual. Esto es ideal para la propia depuración, ya que se puede desarrollar y testear una versión de Linux sobre otro sistema, esto es útil tanto para el propio desarrollo del *kernel*, como para un análisis de seguridad del mismo.

- Técnicas de virtualización incluidas en el *kernel*: en las distribuciones se han ido incorporando diferentes técnicas de virtualización, que necesitan extensiones en el *kernel*; cabe destacar, por ejemplo, *kernel*s modificados para Xen, o Virtual Server (Vserver).

- Nueva versión del soporte de volúmenes LVM2.

- Nuevo pseudo sistema de ficheros */sys,* destinado a incluir la información del sistema, y dispositivos que se irán migrando desde el sistema */proc,* dejando este último con información relacionada con los procesos, y su desarrollo en ejecución.

- Módulo FUSE para implementar sistemas de ficheros en espacio de usuario.

En el futuro se tiene pensado mejorar los aspectos siguientes:

- Incremento de la tecnología de virtualización en el *kernel*, para soportar diferentes configuraciones de operativos, y diferentes tecnologías de virtualización, así como mejor soporte del hardware para virtualización incluido en los procesadores que surjan en las nuevas arquitecturas.
- El soporte de SMP (máquinas multiprocesador), de CPU de 64 bits (Itanium de Intel, y Opteron de AMD), el soporte de CPUs multiCore.
- La mejora de sistemas de ficheros para *clustering*, y sistemas distribuidos.
- Por el contrario, la mejora para *kernel*s más optimizados para dispositivos móviles (PDA, teléfonos...).
- Mejora en el cumplimiento de los estándar POSIX, etc.
- Mejora de la planificación de la CPU, aunque en la serie inicial de la rama 2.6.x se hicieron muchos avances en este aspecto, todavía hay bajo rendimiento en algunas situaciones, en particular en el uso de aplicaciones interactivas de escritorio, se están estudiando diferentes alternativas, para mejorar éste y otros aspectos.

Nota

POSIX:
www.UNIX-systems.org/

También, aunque se aparta de los sistemas Linux, la FSF (Free Software Foundation) y su proyecto GNU siguen trabajando en su proyecto de acabar un sistema operativo completo. Cabe recordar que el proyecto GNU tenía como principal objetivo conseguir un clon UNIX de software libre, y las utilidades GNU sólo son el software de sistema necesario. A partir de 1991, cuando Linus consigue conjuntar su *kernel* con algunas utilidades GNU, se dio un primer paso que ha acabado en los sistemas GNU/Linux actuales. Pero el proyecto GNU sigue trabajando en su idea de terminar el sistema completo. En este momento disponen ya de un núcleo en el que pueden correr sus utilidades GNU. A este núcleo se le denomina Hurd; y a un sistema construido con él se le conoce como GNU/Hurd. Ya existen algunas distribuciones de prueba, en concreto, una Debian GNU/Hurd.

Nota

El proyecto GNU:
http://www.gnu.org/gnu/thegnuproject.html

Hurd fue pensado como el núcleo para el sistema GNU sobre 1990 cuando comenzó su desarrollo, ya que entonces la mayor parte del software GNU estaba desarrollado, y sólo faltaba el *kernel*. Fue en 1991 cuando Linus combinó GNU con su *kernel* Linux y creó así el inicio de los sistemas GNU/Linux. Pero Hurd sigue desarrollándose. Las ideas de desarrollo en Hurd son más complejas, ya que Linux podría considerarse un diseño "conservador", que partía de ideas ya conocidas e implantadas.

Nota

GNU y Linux, por Richard Stallman:
http://www.gnu.org/gnu/linux-and-gnu.html

En concreto, Hurd estaba pensada como una colección de servidores implementados sobre un *microkernel* Mach [Vah96], el cual es un diseño de núcleo

tipo *microkernel* (a diferencia de Linux, que es de tipo monolítico) desarrollado por la Universidad de Carnegie Mellon y posteriormente por la de Utah. La idea base era modelar las funcionalidades del *kernel* de UNIX como servidores que se implementarían sobre un núcleo básico Mach. El desarrollo de Hurd se retrasó mientras se estaba acabando el diseño de Mach, y éste se publicó finalmente como software libre, que permitiría usarlo para desarrollar Hurd. Comentar en este punto la importancia de Mach, ya que muchos operativos se han basado en ideas extraídas de él; el más destacado es el MacOS X de Apple.

El desarrollo de Hurd se retrasó más por la complejidad interna, ya que existían varios servidores con diferentes tareas de tipo *multithread* (de ejecución de múltiples hilos), y la depuración era extremadamente difícil. Pero hoy en día, ya se dispone de las primeras versiones de producción, así como de versiones de prueba de distribución GNU/Hurd.

Puede que en un futuro no tan lejano coexistan sistemas GNU/Linux con GNU/Hurd, o incluso sea sustituido el núcleo Linux por el Hurd, si prosperan algunas demandas judiciales contra Linux (léase caso SCO frente a IBM), ya que sería una solución para evitar problemas posteriores. En todo caso, tanto los unos como los otros sistemas tienen un prometedor futuro por delante. El tiempo dirá hacia dónde se inclina la balanza.

7. Taller: configuración del *kernel* a las necesidades del usuario

En este apartado vamos a ver un pequeño taller interactivo para el proceso de actualización y configuración del *kernel* en el par de distribuciones utilizadas: Debian y Fedora.

Una primera cosa imprescindible, antes de comenzar, es conocer la versión actual que tenemos del *kernel* con uname -r, para poder determinar cuál es la versión siguiente que queremos actualizar o personalizar. Y otra es la de disponer de medios para arrancar nuestro sistema en caso de fallos: el conjunto de CD de la instalación, el disquete (o CD) de rescate (actualmente suele utilizarse el primer CD de la distribución), o alguna distribución en LiveCD que nos permita acceder al sistema de ficheros de la máquina, para rehacer configuraciones que hayan causado problemas. Además de hacer *backup* de nuestros datos o configuraciones importantes.

Veremos las siguientes posibilidades:

1) Actualización del *kernel* de la distribución. Caso automático de Debian.
2) Actualización automática en Fedora.
3) Personalización de un *kernel* genérico (tanto Debian como Fedora). En este último caso los pasos son básicamente los mismos que los que se presentan en el apartado de configuración, pero haremos algunos comentarios más.

7.1. Actualizar *kernel* en Debian

En el caso de la distribución Debian, la instalación puede hacerse también de forma automática, mediante el sistema de paquetes de APT. Puede hacerse tanto desde línea de comandos, como con gestores APT gráficos (synaptic, gnome-apt...).

Vamos a efectuar la instalación por línea de comandos con apt-get, suponiendo que el acceso a las fuentes apt (sobre todo a los Debian originales) está bien configurado en el fichero de /etc/apt/sources.list. Veamos los pasos:

1) Actualizar la lista de paquetes:

```
# apt-get update
```

2) Listar paquetes asociados a imágenes del *kernel*:

```
# apt-cache search linux-image
```

3) Elegir una versión adecuada a nuestra arquitectura (genérica, 386/486/686 para Intel, k6 o k7 para amd, o en particular para 64Bits versiones amd64, intel

y amd, o ia64, para Intel Itanium). La versión va acompañada de versión *kernel*, revisión de Debian del *kernel* y arquitectura. Por ejemplo: 2.6.21-4-k7, *kernel* para AMD Athlon, revisión Debian 4 del *kernel* 2.6.21.

4) Comprobar, para la versión elegida, que existan los módulos accesorios extras (con el mismo número de versión). Con apt-cache buscamos si existen otros módulos dinámicos que puedan ser interesantes para nuestro hardware, según la versión del *kernel* a instalar. Recordar que, como vimos en la "Debian way", también existe la utilidad module-assistant, que nos permite automatizar este proceso después de la compilación del *kernel*. Si los módulos necesarios no fueran soportados, esto nos podría impedir actualizar el *kernel* si consideramos que el funcionamiento del hardware problemático es vital para el sistema.

5) Buscar, si queremos disponer también del código fuente del *kernel*, los Linux-source-version (sólo el 2.6.21, o sea, los números principales), y los *kernel-headers* correspondientes, por si más tarde queremos hacer un *kernel* personalizado: en este caso, el *kernel* genérico correspondiente parcheado por Debian.

6) Instalar lo que hayamos decidido; si queremos compilar desde las fuentes o simplemente disponer del código:

```
# apt-get install linux-image-version
# apt-get install xxxx-modules-version (si fuera necesarios
algunos módulos)
```

y

```
# apt-get install linux-source-version-generica
# apt-get install linux-headers-version
```

7) Instalar el nuevo *kernel*, por ejemplo en el *bootloader* lilo. Normalmente esto se hace automáticamente. Si se nos pregunta si tenemos el initrd activado, habrá que verificar el fichero de lilo (/etc/lilo.conf) e incluir en la configuración lilo de la imagen nueva la nueva línea:

initrd = /initrd.img-version (o /boot/initrd.img-version)

una vez hecha esta configuración, tendríamos que tener un lilo del modo (fragmento), suponiendo que initrd.img y vmlinuz sean enlaces a la posición de los ficheros del nuevo *kernel*:

```
default = Linux

image = /vmlinuz
      label = Linux
      initrd = /initrd.img
#     restricted
#     alias = 1
```

```
image = /vmlinuz.old
      label = LinuxOLD
      initrd = /initrd.img.old
#     restricted
#     alias = 2
```

Tenemos la primera imagen por defecto, la otra es el *kernel* antiguo. Así, desde el menú lilo podremos pedir una u otra, o simplemente cambiando el *default* recuperar la antigua. Siempre que realicemos cambios en /etc/lilo.conf no debemos olvidarnos de reescribirlos en el sector correspondiente con el comando /sbin/lilo o /sbin/lilo -v.

7.2. Actualizar *kernel* en Fedora/Red Hat

La actualización del *kernel* en la distribución Fedora/Red Hat es totalmente automática por medio de su servició de gestión de paquetes, o bien por los programas gráficos que incluye la distribución para la actualización; por ejemplo, en los Red Hat empresariales existe uno llamado up2date. Normalmente, lo encontraremos en la barra de tareas o en el menú de Herramientas de sistema de Fedora/Red Hat.

Este programa de actualización básicamente verifica los paquetes de la distribución actual frente a una base de datos de Fedora/Red Hat, y da la posibilidad de bajarse los paquetes actualizados, entre ellos los del *kernel*. Este servicio de Red Hat empresarial funciona por una cuenta de servicio, y Red Hat lo ofrece por pago. Con este tipo de utilidades la actualización del *kernel* es automática.

Por ejemplo, en la figura 10, observamos que una vez puesto en ejecución nos ha detectado una nueva versión del *kernel* disponible y podemos seleccionarla para que nos la descargue:

Figura 3. El servicio de actualización de Red Hat (Red Hat Network up2date) muestra la actualización del *kernel* disponible y sus fuentes.

En Fedora podemos o bien utilizar las herramientas gráficas equivalentes, o simplemente con yum directamente, si conocemos la disponibilidad de nuevos *kernel*s:

```
# yum install kernel kernel-source
```

Una vez descargada, se procederá a su instalación, normalmente también de forma automática, ya dispongamos de grub o lilo, como gestores de arranque. En el caso de grub, suele ser automático y deja un par de entradas en el menú, una para la versión más nueva y otra para la antigua. Por ejemplo, en esta configuración de grub (el fichero está en /boot/grub/grub.conf o bien /boot/grub/menu.lst), tenemos dos *kernel*s diferentes, con sus respectivos números de versión:

```
#fichero grub.conf
default = 1
timeout = 10
splashimage = (hd0,1)/boot/grub/splash.xpm.gz

title Linux (2.6.20-2945)
root (hd0,1)
kernel /boot/vmlinuz-2.6.20-2945 ro root = LABEL = /
initrd /boot/initrd-2.6.20-18.9.img

title LinuxOLD (2.6.20-2933)
root (hd0,1)
kernel /boot/vmlinuz-2.4.20-2933 ro root = LABEL = /
initrd /boot/initrd-2.4.20-2933.img
```

Cada configuración incluye un título, que aparecerá en el arranque; el *root*, o partición del disco desde donde arrancar; el directorio donde se encuentra el fichero correspondiente al *kernel*, y el fichero initrd correspondiente.

En el caso de que dispongamos de lilo (por defecto se usa grub) en la Fedora/Red Hat como gestor, el sistema también lo actualiza (fichero /etc/lilo.conf), pero luego habrá que reescribir el arranque con el comando /sbin/lilo manualmente.

Cabe señalar, asimismo, que con la instalación anterior teníamos posibilidades de bajarnos las fuentes del *kernel*; éstas, una vez instaladas, están en /usr/src/linux-version, y pueden configurarse y compilarse por el procedimiento habitual como si fuese un *kernel* genérico. Hay que mencionar que la empresa Red Hat lleva a cabo un gran trabajo de parches y correcciones para el *kernel* (usado después en Fedora), y que sus *kernel* son modificaciones al estándar genérico con bastantes añadidos, por lo cual puede ser mejor utilizar las fuentes propias de Red Hat, a no ser que queramos un *kernel* más nuevo o experimental que el que nos proporcionan.

7.3. Personalizar e instalar un *kernel* genérico

Vamos a ver el caso general de instalación de un *kernel* a partir de sus fuentes. Supongamos que tenemos unas fuentes ya instaladas en /usr/src (o el prefijo correspondiente). Normalmente, tendremos un directorio linux, o linux-version o sencillamente el número version; éste será el árbol de las fuentes del *kernel*.

Estas fuentes pueden provenir de la misma distribución (o que las hayamos bajado en una actualización previa), primero será interesante comprobar si son las últimas disponibles, como ya hemos hecho antes con Fedora o Debian. O si queremos tener las últimas y genéricas versiones, podemos ir a kernel.org y bajar la última versión disponible, mejor la estable que las experimentales, a no ser que estemos interesados en el desarrollo del *kernel*. Descargamos el archivo y descomprimimos en /usr/src (u otro elegido, casi mejor) las fuentes del *kernel*. También podríamos buscar si existen parches para el *kernel* y aplicarlos (según hemos visto en el apartado 4.4).

A continuación comentaremos los pasos que habrá que realizar; lo haremos brevemente, ya que muchos de ellos los tratamos anteriormente cuando trabajamos la configuración y personalización.

Nota

Sería conveniente releer el apartado 3.4.3.

1) Limpiar el directorio de pruebas anteriores (si es el caso):

```
make clean mrproper
```

2) Configurar el *kernel* con, por ejemplo: make menuconfig (o xconfig, gconfig o oldconfig). Lo vimos en el apartado 4.3.

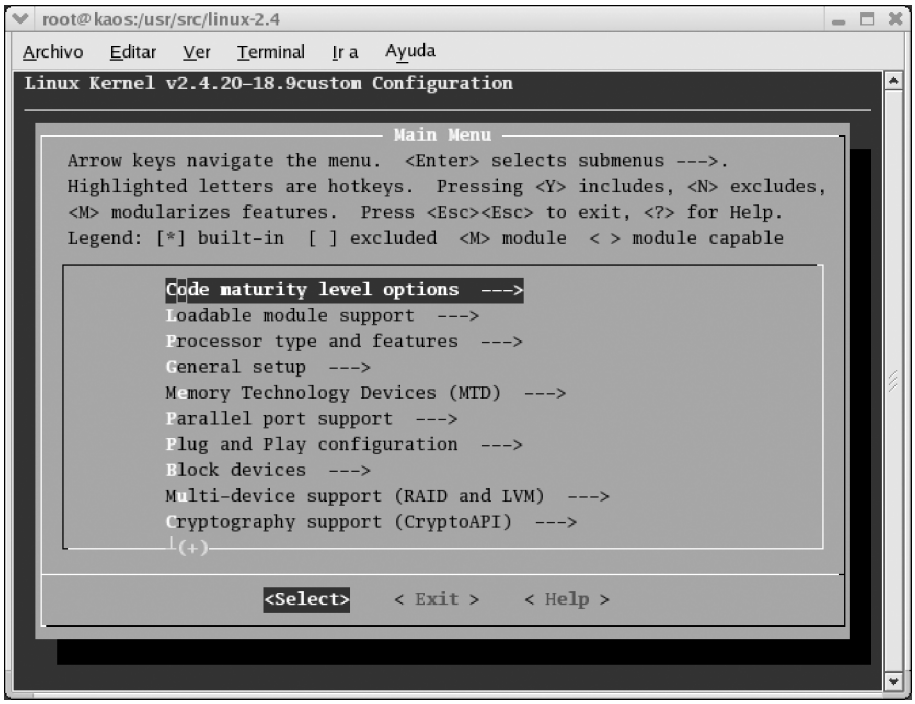

Figura 4. Configuración del *kernel* por menús textuales

4) Dependencias y limpieza de compilaciones anteriores:

```
make dep
```

5) Compilación y creación de la imagen del *kernel*: make bzImage. También sería posible zImage si la imagen fuera más pequeña, pero es más normal bzImage, que optimiza el proceso de carga y la compresión para *kernel*s más grandes. En algún hardware muy antiguo puede no funcionar y ser necesario zImage. El proceso puede durar desde unos pocos minutos a una hora en hardware moderno (CPU de 1-3 GHz) y horas en hardware muy antiguo. Cuando acaba, la imagen se halla en: /usr/src/directorio-fuentes/arch/i386/boot.

6) Ahora compilamos los módulos con *make modules*. Hasta este momento no hemos modificado nada en nuestro sistema. Ahora tendremos que proceder a la instalación.

7) En el caso de los módulos, si probamos alguna versión antigua del *kernel* (rama 2.2 o primeras de la 2.4), hay que tener cuidado, ya que en alguna se sobrescribían los antiguos (en las últimas 2.4.x o 2.6.x ya no es así).

Pero también cuidado si estamos compilando una versión que es la misma (exacta numeración) que la que tenemos (los módulos se sobrescribirán), mejor hacer un *backup* de los módulos:

```
cd /lib/modules
tar -cvzf old_modules.tgz versionkernel-antigua/
```

Así, tenemos una versión en .tgz que podríamos recuperar después en caso de problemas. Y, finalmente, instalamos los módulos con:

```
make modules install
```

8) Ahora podemos pasar a la instalación del *kernel*, por ejemplo con:

```
# cd /usr/src/directorio-Fuentes/arch/i386/boot
# cp bzImage /boot/vmlinuz-versionkernel
# cp System.map /boot/System.map-versionkernel
# ln -s /boot/vmlinuz-versionkernel /boot/vmlinuz
# ln -s /boot/System.map-versionkernel /boot/System.map
```

Así colocamos el fichero de símbolos del *kernel* (System.map) y la imagen del *kernel*.

9) Ya sólo nos queda poner la configuración necesaria en el fichero de configuración del gestor de arranque, ya sea lilo (/etc/lilo.conf) o grub (/boot/grub/grub.conf) según las configuraciones que ya vimos con Fedora o Debian. Y re-

cordar, en el caso de lilo, que habrá que volver a actualizar la configuración con /sbin/lilo o /sbin/lilo -v.

10) Reiniciar la máquina y observar los resultados (si todo ha ido bien).

Actividades

1) Determinar la versión actual del *kernel* Linux incorporada en nuestra distribución. Comprobar las actualizaciones disponibles de forma automática, ya sea en Debian *(apt)* o en Fedora/Red Hat (vía yum o up2date).

2) Efectuar una actualización automática de nuestra distribución. Comprobar posibles dependencias con otros módulos utilizados (ya sean pcmcia, u otros), y con el *bootloader* (lilo o grub) utilizado. Se recomienda *backup* de los datos importantes del sistema (cuentas de usuarios y ficheros de configuración modificados), o bien realizarlo en otro sistema del que se disponga para pruebas.

3) Para nuestra rama del *kernel*, determinar la última versión disponible (consultar www.kernel.org), y realizar una instalación manual con los pasos examinados en la unidad. La instalación final puede dejarse opcional, o bien poner una entrada dentro del *bootloader* para las pruebas del nuevo *kernel*.

4) En el caso de la distribución Debian, además de los pasos manuales, existe como vimos una forma especial (recomendada) de instalar el *kernel* a partir de sus fuentes mediante el paquete *kernel-package*.

Otras fuentes de referencia e información

[Kerb] Sitio que proporciona un almacén de las diversas versiones del *kernel* Linux y sus parches.

[Kera] [lkm] Sitios web que recogen una parte de la comunidad del *kernel* de Linux. Dispone de varios recursos de documentación y listas de correo de la evolución del *kernel*, su estabilidad y las nuevas prestaciones que se desarrollan.

[DBo] Libro sobre el *kernel* de Linux 2.4, que detalla los diferentes componentes y su implementación y diseño. Existe una primera edición sobre el *kernel* 2.2 y una nueva actualización al *kernel* 2.6.

[Pra] Artículo que describe algunas de las principales novedades de la nueva serie 2.6 del *kernel* Linux.

[Ker] [Mur] Proyectos de documentación del *kernel*, incompletos pero con material útil.

[Bac86] [Vah96] [Tan87] Algunos textos sobre los conceptos, diseño e implementación de los *kernel*s de diferentes versiones UNIX.

[Skoa][Zan01][Kan][Pro] Para tener más información sobre los cargadores lilo y grub.

Administración local

Josep Jorba Esteve

P07/M2103/02284

www.uoc.edu

Índice

Introducción

Una de las primeras tareas con la que tendrá que enfrentarse el administrador será la gestión de los recursos locales presentes en la máquina. En el curso de introducción a GNU/Linux, se cubrieron algunos de estos aspectos de forma básica. En el presente, veremos algunas de estas tareas de administración de forma más profunda, y algunos de los aspectos de personalización y rendimiento de los recursos.

Comenzaremos por analizar el proceso de arranque de un sistema GNU/Linux, que nos hará comprender la estructura inicial del sistema y su relación con los diversos servicios que éste proporciona.

A continuación aprenderemos cómo obtener una visión general del estado actual del sistema por medio de los diferentes procedimientos y comandos de que se dispone para evaluar las diversas 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 éstos; veremos cómo definir nuevos usuarios al 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 y cómo mantenerlo actualizado; 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. 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], 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 pasadas versiones de la mayoría de distribuciones que suelen 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 como opcional.

Nota

Es importante conocer los detalles de una distribución, ya que pueden ser básicos para resolver una tarea o acelerar su solución (por ejemplo, si dispone de herramientas propias).

- 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, dejando xinetd como opción.

- 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 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.

- Dispone de un software llamado kudzu, que examina el hardware en arranque para detectar posibles cambios de configuración y generar los dispositi-

vos o configuraciones adecuadas. Aunque se está migrando progresivamente a la API Hal que controla precisamente este tema.

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 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 gdm
```

  permite escoger el gestor de entrada para X, o:

```
dpkg-reconfigure X-Window-system
```

  nos permite configurar los diferentes elementos de X.

- 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.

- 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.

2. 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].

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

La activación o parada de servicios se realiza mediante la utilización de *scripts*. La mayoría de los servicios estándar, los cuales 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. 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 básicamente una configuración de programas y servicios que se ejecutarán orientados a un determinado funcionamiento.

Los niveles típicos, aunque 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 estándar LSB), suelen ser:

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 sólo la entrada del administrador (*root*). Se usa para tareas de mantenimiento y corrección de errores críticos.

Runlevel	Función	Descripción
2	Multiusuario sin red	No se inician servicios de red, permitiendo sólo 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 el 3, pero con soporte X para la entrada de usuarios (*login* gráfico).
6	Reinicio	Para todos los programas y servicios. Reinicia el sistema.

Por contra cabe señalar que Debian usa un modelo, en el que los niveles 2-5 son prácticamente 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 sysV). En el caso de Fedora y Debian, se utiliza el sistema SystemV, que es el que mostraremos, pero otros UNIX y alguna distribución GNU/Linux (como Slackware) utilizan el modelo BSD.

En el caso del modelo *runlevel* de SystemV, cuando el proceso init 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, nos permite cambiar de nivel de ejecución, sólo 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 también un intervalo de tiempo para hacerse, o bien inmediatamente. Para estas tareas también existen los comandos halt y reboot.

- 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 suele utilizarlosde 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: 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 e 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.

  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*; invoke-rc.d, 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. 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 subapartados 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 esta unidad, realizaremos un examen completo de un sistema ejemplo, para que se puedan ver algunas de estas técnicas.

3.1. Arranque del sistema

En el arranque de un sistema GNU/Linux, se produce todo un volcado de información interesante; cuando el sistema arranca, suelen aparecen 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.

En la mayoría de las distribuciones esto 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*).
- Comando uptime: indica cuánto tiempo hace que el sistema está activo.
- Sistema /proc: pseudo sistema de ficheros (procfs) que utiliza el kernel para almacenar la información de procesos y de sistema.
- Sistema /sys: pseudo sistema 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*).

3.2. *Kernel*: directorio /proc

El *kernel* durante su arranque pone en funcionamiento un pseudo-filesystem llamado /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 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 brevemente 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/drive	Información de algunos módulos *kernel* de hardware
/proc/filesystems	Sistemas de ficheros habilitados en el *kernel*
/proc/ide	Directorio de información del bus IDE, características 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 ha iniciado una transición progresiva de procfs (/proc) a sysfs (/sys) con 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.

3.3. *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 (u sus controladores). Suele ser utilizada por capas como HAL y 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 cómo después se accede a él a través del sistema de ficheros sysfs (la estructura del cual 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 éste estaba mezclando diferentes cosas (dispositivos, procesos, datos hardware, parámetros *kernel*) de forma no coherente, y ésta fue una de las decisiones para crear /sys. Se espera que progresivamente se migre información de /proc a /sys para centralizar la información de los dispositivos.

3.4. 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,

depende si se está ofreciendo el servicio (actuamos de servidor) o estamos recibiendo los resultados del servicio (actuamos de clientes). La mayoría de estos procesos aparecerán asociados al usuario *root*, aunque no estemos presentes en ese momento como usuarios. Puede haber algunos servicios asociados a otros usuarios de sistema (lp, bin, www, mail, etc.), estos son usuarios "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 (ver 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. Útil para procesos con comportamiento inestable o programas interactivos que han dejado de responder. Podemos ver una lista de las señales validas en el sistema con man 7 signal

3.5. *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 malfuncionamiento o situaciones críticas. Este tipo de *logs* son imprescindibles en muchos casos para las tareas de administración y se suele emplear bastante tiempo de administración en el procesamiento y análisis de sus contenidos.

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*. Este *daemon* 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. Éste 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, puede estar configurado de diferentes modos por defecto, en /var/log suele generar (por ejemplo) en Debian 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.

Otros comandos relacionados:

- 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 rearranques 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(Debian), log_analysis (Debian)...

3.6. Memoria

Respecto a la memoria del sistema, tendremos que tener en cuenta que disponemos: a) de la memoria física de la propia máquina, b) 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 32bits). Esto obligará a utilizar una zona de intercambio (*swap*) sobre disco, para implementar los procesos asociados a memoria virtual.

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 consulta 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 usan 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 permite recoger datos de los diferentes parámetros (memoria, swap, y otros) a intervalos de tiempo (de forma parecida a top).

3.7. Discos y *filesystems*

Examinaremos qué discos tenemos disponibles, cómo están organizados y de qué particiones y archivos de sistemas (*filesystems*) disponemos.

Cuando dispongamos de una partición, y dispongamos de un determinado *filesystem* accesible, tendremos que realizar un proceso de montaje, para inte-

grarla 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 (dmesg), donde se detectan los presentes, como los /dev/hdx para los dispositivos IDE o los SCSI con dispositivos /dev/sdx. 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 se verán como dispositivo de este tipo.

Cualquier dispositivo de almacenamiento presentará una serie de particiones de su espacio. Típicamente, un disco IDE soporta un máximo de cuatro particiones físicas, o más si éstas son lógicas (que 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.

Para examinar la estructura de un dispositivo conocido, o cambiar su estructura particionando el disco, podemos utilizar el comando fdisk, o cualquiera de sus variantes más o menos interactivas (cfdisk, sfdisk). Por ejemplo, al examinar un disco ejemplo ide /dev/hda, nos da la siguiente información:

```
# fdisk /dev/hda        (dentro opción p)
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 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 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.

De los discos y particiones de que dispongamos, algunos de ellos 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 (veremos 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 sólo 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 *filesystem* 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. Comando básico para controlar espacio de disco disponible.

Respecto a este ultimo comando df -k, una de nuestras tareas básicas de administración de la máquina es controlar los recursos de la máquina, 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* (y más si es el /) por debajo de un 10 o 15%, ya que hay muchos procesos *daemons* que están escribiendo normalmente información temporal o *logs,* que pueden generar gran información; un caso particular lo forman los ficheros core, ya comentados, que 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, según la configuración del sistema (por ejemplo de Syslogd*)* la información generada de mensajes puede ser muy grande. Normalmente, 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: a) ficheros core de los usuarios: podemos eliminarlos periódicamente o eliminar su generación; b) 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; c) las cachés de los navegadores u otras aplicaciones: también suelen tener tamaños grandes, otra limpieza que habrá que hacer periódicamente; d) las cuentas de los propios usuarios: pueden tener cuotas para no exceder los tamaños prefijados, etc.

4. Sistema de ficheros

En cada máquina con un sistema GNU/Linux podemos encontrarnos con diferentes sistemas de ficheros de diferentes tipos [Hin].

Para empezar, es habitual encontrarse con los propios sistemas de ficheros Linux creados en varias particiones de los discos [Koe]. La configuración habitual suele ser de dos particiones: la correspondiente a "/" (*root filesystem*) y la correspondiente al fichero de intercambio o de swap. Aunque 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). La idea es separar claramente partes estáticas y dinámicas del sistema, como 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), o el nuevo ext3, que es una mejora del ext2 compatible 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 o XFS.

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 estandard ext2, y el 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).

Nota

El documento Filesystems Howto, da breves explicaciones de los diversos sistemas de ficheros así como direcciones web de interés para cada uno de ellos.

- Compatibilidad con entornos no GNU/Linux: msdos, vfat, ntfs, acceso a los diferentes sistemas de fat16, fat32, y ntfs. En particular resaltar que el soporte *kernel*, en el caso del kernel esta limitado a lectura. Pero como ya hemos comentado, existen soluciones en espacio de usuario (mediante FUSE, un componente que permite escribir 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.

- Sistems asociados a soportes físicos, como 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), o ReiserFS.

- 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, Coda.

- Pseudo Sistemas 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.

4.1. 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 o grub), pueden existir varias particiones asignadas a los diferentes operativos. Muchas veces es inte-

resante compartir datos con estos sistemas, ya sea para leer sus ficheros o modificarlos. A diferencia de otros sistemas (que sólo 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 con sistemas NTFS (ntfs para Linux) y FAT32 (vfat) a la vez.

Nuestro sistema GNU/Linux puede leer datos (o sea 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 encriptados. Y acceder con según que *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 (http://www.ntfs-3g.org), 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. Se seguirá el mismo proceso si se trata de un dispositivo de almacenamiento, ya sea disquete o floppy.

Dependiendo de la distribución, se usan unos 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 preferido últimamente por las distribuciones. Según el es-

tandard 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)... (de los posibles).

El dispositivo 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 ejemplos:

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

montaría el CD-ROM (si es el IDE que está en el segundo IDE de forma máster) 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.

```
mount -t ntfs /dev/hda2 /mnt/winXP
```

montaría la segunda partición del primer dispositivo IDE (la C:), de tipo NTFS (por ejemplo un Windows XP).

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/hda2    /           ext3      errors = remountro  0  1
/dev/hdb3    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/hda2, la partición de swap que está en hdb3, 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).

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, sólo:

```
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, que se puede consultar, o ejecutar mount sin parámetros para obtener esta información.

4.2. Permisos

Otro tema 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, recordando 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. Usuarios y grupos

Los usuarios de un sistema GNU/Linux disponen normalmente 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 sólo puede ser usado por éste (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 sólo 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 sólo 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 le haya habilitado.

- Cuentas especiales de los servicios: lp, news, 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 nombre de *root*.

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 está incluida en los siguientes archivos:

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

Ejemplo de unas líneas del /etc/passwd:

```
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 en /etc/group.
- Juan García: comentario, suele ponerse el nombre completo del usuario.
- /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, sólo una "x" que indica que se encuentran en otro fichero, que es sólo 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 ":":

- Días desde 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 1 enero 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 un sistema denominado md5 (suele aparecer como opción a la hora de instalar el sistema) para proteger las palabras de paso de los usuarios. Veremos más detalles al respecto en la unidad dedicada a la seguridad.

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, ya que 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, 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. Recordar 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 informa-

ción 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*:* cambia 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 que 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, 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 éstos 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 de su propio 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 compat, 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 supo-

ne 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 éste y otros modos de configuración en las unidades de administración de red.

6. Servidores 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*). Éste 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.

LPD es un sistema bastante antiguo, ya 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. El sistema LPD no estuvo pensado como un sistema basado en controladores de dispositivo, ya que normalmente se producían sólo 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. Así, normalmente el proceso que se sigue es (básicamente):

Nota

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

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.

Nota

Ghostscript: http://www.cs.wisc.edu/ghost/

Además del sistema de impresión LPD (con origen en los BSD UNIX), también existe el denominado sistema System V (de origen en la otra rama UNIX de System V). Normalmente, por compatibilidad, la mayor parte de UNIX integran actualmente ambos sistemas, 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 sólo los comandos LPD de sistema de impresión, pero también será habitual disponer de los

comandos System V. Una forma sencilla de identificar los dos sistemas (BSD o System V), es con el comando principal de impresión (el que envía los trabajos al sistema), en BSD es lpr, y en System V es lp.

Éste 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. Siendo, de hecho, últimamente CUPS el estándar de facto para GNU/Linux, aunque los otros sistemas han de ser soportados por compatibilidad con sistemas UNIX existentes.

Nota

LPRng: http://www.lprng.org
CUPS: http://www.cups.org

Los dos (tanto CUPS como LPRng) son una especie de sistemas de más alto nivel, pero que no se diferencian en mucho de cara al usuario respecto a los BSD y System V 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 sólo 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 permitía cambiar el sistema, conmutar de sistema de impresión, aunque últimamente solo está disponible CUPS. En Debian pueden instalarse ambos sistemas, pero son exclusivos, sólo uno de ellos puede gestionar la impresión.

En el caso de Fedora Core, 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 utiliza BSD LPD, pero es común instalar CUPS (y es previsible que sea la opción por defecto en nuevas revisiones), 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 éstas.

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 básicamente para ser un reemplazo del BSD, por lo tanto, la mayor parte de la configuración es parecida y únicamente 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 System V.

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

- Sistema de impresión que se utiliza: BSD, LPRng o CUPS.

- 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 puestas simplemente en red, como una máquina más, o con protocolos especiales propietarios. Las conectadas a red, pueden normalmente actuar ellas mismas de servidor de impresión (por ejemplo, muchas láser HP 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 usados: cada sistema de impresión soporta uno o varios.

- 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 es 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.

Nota

Se puede encontrar información de las impresoras más adecuadas y de los controladores en:
http://www.linuxprinting.org/foomatic.html

Respecto a la parte cliente del sistema, los comandos básicos son iguales para los diferentes sistemas, y éstos 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 co-

rrespondiente, y asigna un número de trabajo, que será usado con los otros comandos. Normalmente, la impresora por defecto estaría indicada por una variable de sistema PRINTER, o se utilizará la primera que exista definida, o en algunos sistemas se utiliza la cola lp (como nombre por defecto).

Ejemplo

Ejemplo de lpr:

```
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".

- lpq: nos permite examinar los trabajos existentes en la cola.

Ejemplo

Ejemplo

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

Este comando nos muestra los trabajos en cola, con el orden y tamaños de éstos; los ficheros pueden aparecer con nombres diferentes, ya que depende de si los hemos enviado con lpr, o con otra aplicación que puede cambiarlos de nombre al enviarlos, o si han tenido que pasar por algún filtro al convertirlos.

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

Ejemplo

```
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, 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 System V, los comandos de impresión suelen también estar disponibles, normalmente 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 los tres sistemas principales. Estos servidores sirven tanto para la impresión local, como para atender las impresiones de clientes de red (si están habilitados).

6.1. 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 al tema de los permisos, por defecto BSD LPD sólo deja acceso local a la impresora, y por lo tanto, hay que habilitarlo expresamente en /etc/hosts.lpd.

Ejemplo

El fichero podría ser:

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

que 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 "-".

Respecto 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 éste 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 ésta.

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

Veamos un ejemplo:

```
# 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
```

6.2. LPRng

En el caso del sistema LPRng, ya que éste se hizo para mantener la compatibilidad con BSD, y entre otros mejorar aspectos de acceso, el sistema es compatible a nivel de configuración de colas, y se lleva a cabo a través del mismo formato de fichero de /etc/printcap, con algunos añadidos propios.

Donde la configuración resulta diferente es en el tema del acceso, en este caso se realiza a través de un fichero /etc/lpd.perms en general para todo el sistema, y pueden existir también configuraciones individuales de cada cola, con el fichero lpd.perms, colocado en el directorio correspondiente a la cola, normalmente /var/spool/lpd/nombre-cola.

Estos ficheros lpd.perms tienen una capacidad superior de configurar el acceso, permitiendo los siguientes comandos básicos:

```
DEFAULT ACCEPT
DEFAULT REJECT
ACCEPT [ key = value[,value]* ]*
REJECT [ key = value[,value]* ]*
```

donde los dos primeros nos permiten establecer el valor por defecto, de aceptar todo, o rechazar todo, y los dos siguientes, aceptar o rechazar una configuración concreta especificada en la línea. Se pueden aceptar (o rechazar) peticiones de un *host*, usuario, o puertos IP específicos. Asimismo, se puede configurar qué tipo de servicio se proporcionará al elemento: X (puede conectarse), P (impresión de trabajos), Q (examinar cola con lpq), M (borrar trabajos de la cola, lprm), C (control de impresoras, comando lpc), entre otros, así en el fichero:

```
ACCEPT SERVICE = M HOST = first USER = jose
ACCEPT SERVICE = M SERVER REMOTEUSER = root
REJECT SERVICE = M
```

Se permite borrar trabajos de la cola, al usuario (jose) de la máquina (*first*), y al usuario *root*, del servidor donde esté alojado el servicio de impresión (*localhost*), además, se rechazan cualesquiera otras peticiones de borrar de la cola trabajos que no sean las peticiones ya establecidas.

Con esta configuración hay que tener especial cuidado, porque en algunas distribuciones, los servicios LPRng están por defecto abiertos. Puede limitarse la conexión por ejemplo con:

```
ACCEPT SERVICE = X SERVER
REJECT SERVICE = X NOT REMOTEIP = 100.200.0.0/255
```

Servicio de conexión sólo accesible a la máquina local del servidor, y rechazado si no se pertenece a nuestra subred (en este caso, suponemos que sea 100.200.0.x).

Para la administración de línea de comandos, se usan las mismas herramientas que el BSD, estándar. En el terreno de la administración gráfica del sistema, cabe destacar la herramienta *lprngtool* (no disponible en todas las versiones del sistema LPRng).

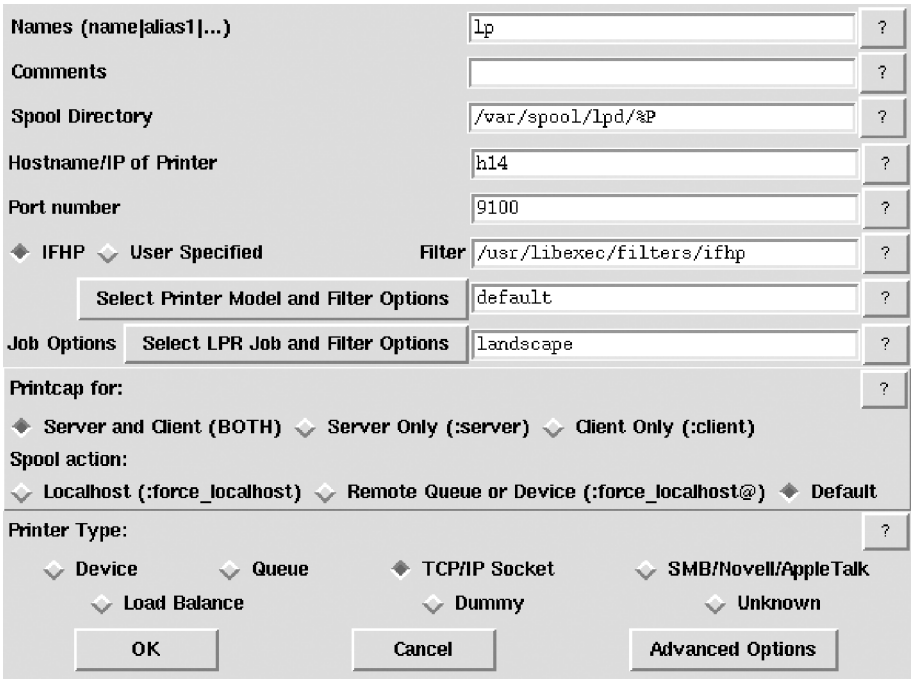

Figura 1. lprngtool, configuración de una impresora

Hay varios paquetes de software relacionados con LPRng, por ejemplo en una Debian encontramos:

```
lprng - lpr/lpd printer spooling system
lprng-doc - lpr/lpd printer spooling system (documentation)
lprngtool - GUI frontend to LPRng based /etc/printcap
printop - Graphical interface to the LPRng print system.
```

6.3. 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 sólo 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 (caso HP y Epson).

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

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, normalmente 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.

Destacar que CUPS en principio está pensado para que tanto clientes, como el servidor funcionen bajo el mismo sistema, si los clientes utilizan LPD o LPRng, hay que instalar un *daemon* de compatibilidad llamado cups-lpd (normalmente en paquetes como cupsys-bsd). En este caso, CUPS acepta trabajos que provengan de un sistema LPD o LPRng, pero no controla los accesos (cupsd.conf sólo sirve para el propio sistema CUPS), por lo tanto, habrá que implementar alguna estrategia de control de acceso, tipo *firewall* por ejemplo (ver unidad de seguridad).

Para la administración desde línea de comandos, CUPS es un tanto peculiar, ya que acepta tanto comandos LPD como System V 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 o la interfaz por web que trae el mismo sistema CUPS, accesible en http://localhost:631.

Figura 2. Interfaz para la administración del sistema CUPS

Respecto a los paquetes software relacionados con CUPS, en una Debian encontramos (entre otros):

```
cupsys - Common UNIX Printing System(tm) - server
cupsys-bsd - Common UNIX Printing System(tm) - BSD commands
cupsys-client - Common UNIX Printing System(tm) - client
programs (SysV)
cupsys-driver-gimpprint - Gimp-Print printer drivers for CUPS
cupsys-pt - Tool for viewing/managing print jobs under CUPS
cupsomatic-ppd - linuxprinting.org printer support -
transition package
foomatic-db - linuxprinting.org printer support - database
foomatic-db-engine - linuxprinting.org printer support -
programs
foomatic-db-gimp-print - linuxprinting - db Gimp-Print
printer drivers
foomatic-db-hpijs - linuxprinting - db HPIJS printers
foomatic-filters - linuxprinting.org printer support -
filters
foomatic-filters-ppds - linuxprinting - prebuilt PPD files
foomatic-gui - GNOME interface for Foomatic printer filter
system
gimpprint-doc - Users' Guide for GIMP-Print and CUPS
gimpprint-locales - Locale data files for gimp-print
gnome-cups-manager - CUPS printer admin tool for GNOME
gtklp - Frontend for cups written in gtk
```

7. Discos y gestión filesystems

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

- IDE: dispositivos

  /dev/hda disco máster, primer conector IDE;

  /dev/hdb disco *slave* del primer conector,

  /dev/hdc máster segundo conector,

  /dev/hdd *slave* segundo conector.

- SCSI: dispositivos /dev/sda, /dev/sdb... siguiendo la numeración que tengan los periféricos en el Bus SCSI.

- Disquetes: 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, éstos permiten cuatro particiones denominadas primarias, y un mayor número en lógicas. Así, si /dev/hdan, *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 los englobamos en:

- Creación de particiones, o modificación de éstas. Mediante comandos como fdisk o parecidos (cfdisk, sfdisk).

- 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 diversos mkfs.ext2, mkfs.ext3, y

también *filesystems* no linux: mkfs.ntfs, mkfs.vfat, mkfs.msdos, mkfs.minix, u otros. Para CD-ROM como mkisofs para 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 permitirá finalmente crear/grabar los CD/DVD. Otro caso particular es la orden mkswap, que permite crear áreas de swap en particiones, que después 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 que sea posible, corregirlos. El propio sistema activa automáticamente el comando 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 normalmente se realiza con el dispositivo en modo de "sólo 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 sólo *root*, y hagamos allí la verificación. En general, si hay que hacer la verificación, se recomienda hacer éstas en modo superusuario (podemos conmutar en modo *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, tamaño, bloque o sector, y puede enviarse también a cintas.

- Utilidades diversas: algunos comandos individuales, algunos de ellos utilizados por los procesos anteriores para hacer tratamientos diversos: *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 o ext3 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.

7.1. RAID software

Las configuraciones 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 se basa 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 como en algunos esquemas que están diseñados para aumentar las prestaciones del sistema de discos, ampliando el ancho de banda de éstos disponible hacia el sistema y las aplicaciones.

Hoy en día podemos encontrar RAID en hardware principalmente en servidores empresariales (aun cuando comienzan a tener cierta presencia en equipos de escritorio), donde se encuentran disponibles diferentes soluciones hardware que cumplen estos requisitos. En particular para aplicaciones intensivas en disco, como *streaming* de audio y/o vídeo, o grandes bases de datos.

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

En RAID se distinguen 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 discos iguales en tamaño (o iguales a tamaños de grupos). Por ejemplo, para realizar un caso de *array* podrían utilizarse 4 discos de 100GB, o en otro caso, por ejemplo, 2 grupos (a 100GB) de 2 discos, uno de 30GB y otro de 70GB. 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 (téngase 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. Sólo se están repartiendo datos, si el disco falla físicamente, la in-

formación se pierde y debemos recuperarla desde copias de seguridad. Lo que sí que aumenta es el rendimiento, dependiendo de la implementación de RAID0, ya que las operaciones de lectura y escritura se dividirán entre los diferentes discos.

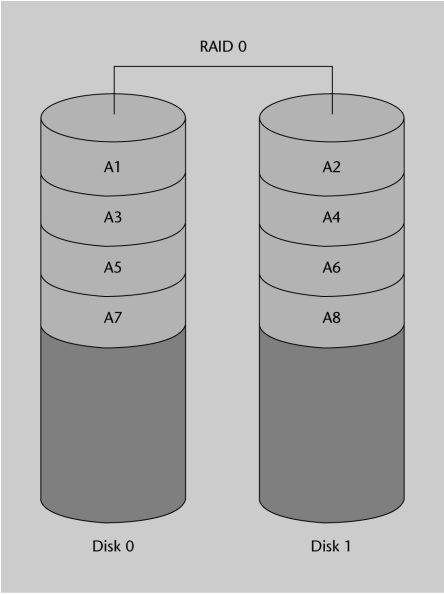

Figura 3

- 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 de los discos. Si detectamos el fallo en uno de ellos, podemos sustituirlo sin apagar el sistema, por un disco nuevo.

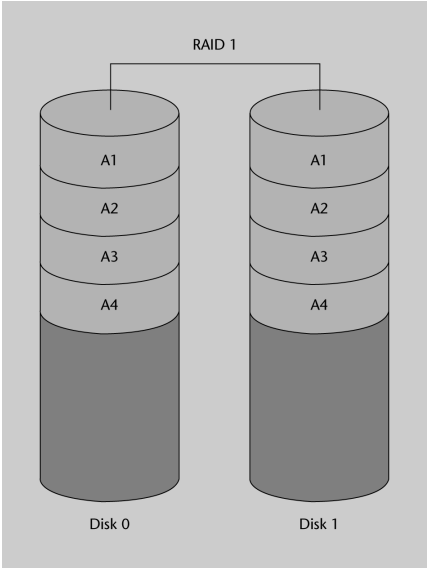

Figura 4

- RAID 2: En los anteriores se divide los datos en bloques a repartir, aquí, se divide en bits y se utilizan códigos de redundancia, para la corrección de datos. Prácticamente no se utiliza, 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 calculo 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 dividendo a nivel de bloques en lugar de bytes, permitiendo que sí que 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 calculo se implementa de forma sencilla por hardware, con buenas prestaciones.

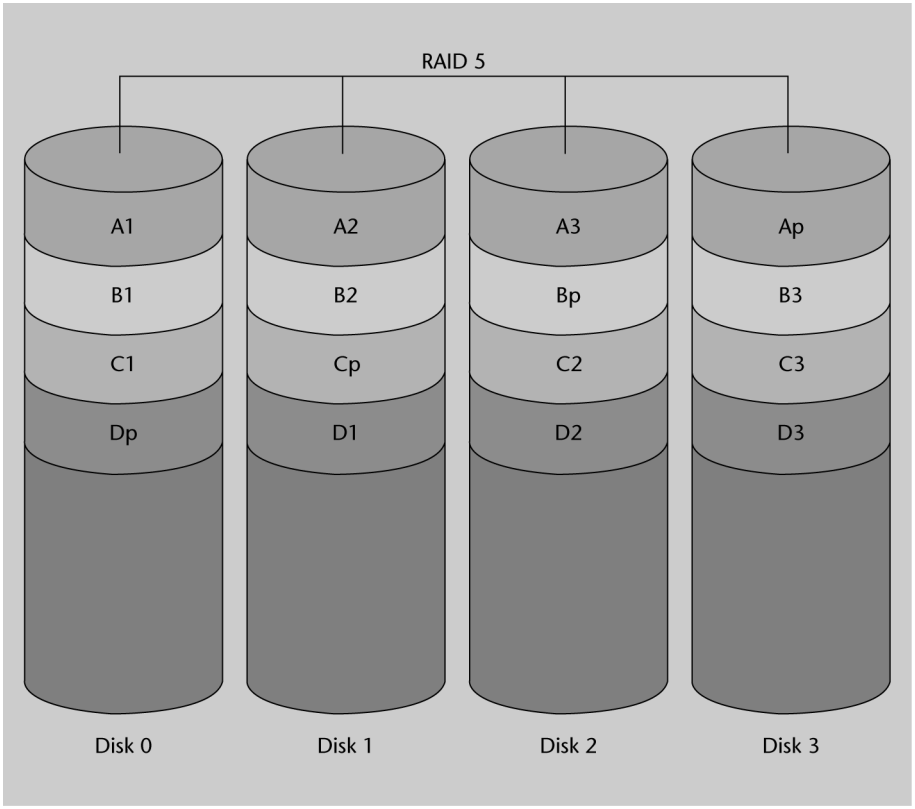

Figura 5

- RAID 0+1 (o 01): Un *mirror* de divisiones, se trata de un nivel de RAID anidado, se implementan, por ejemplo, dos grupos de RAID0, 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 RAID0 usado gracias a la otra copia, pero si quieren añadirse discos hay que añadirlos a todos los grupos de RAID0 de igual forma.

- RAID 10 (1+0): división de *mirrors*, grupos de RAID 1 bajo RAID 0. Así, en cada grupo de RAID1 puede llegar a fallar un disco sin que se pierdan datos. Claro que esto obliga a reemplazarlos, ya que, si no, el disco que queda 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).

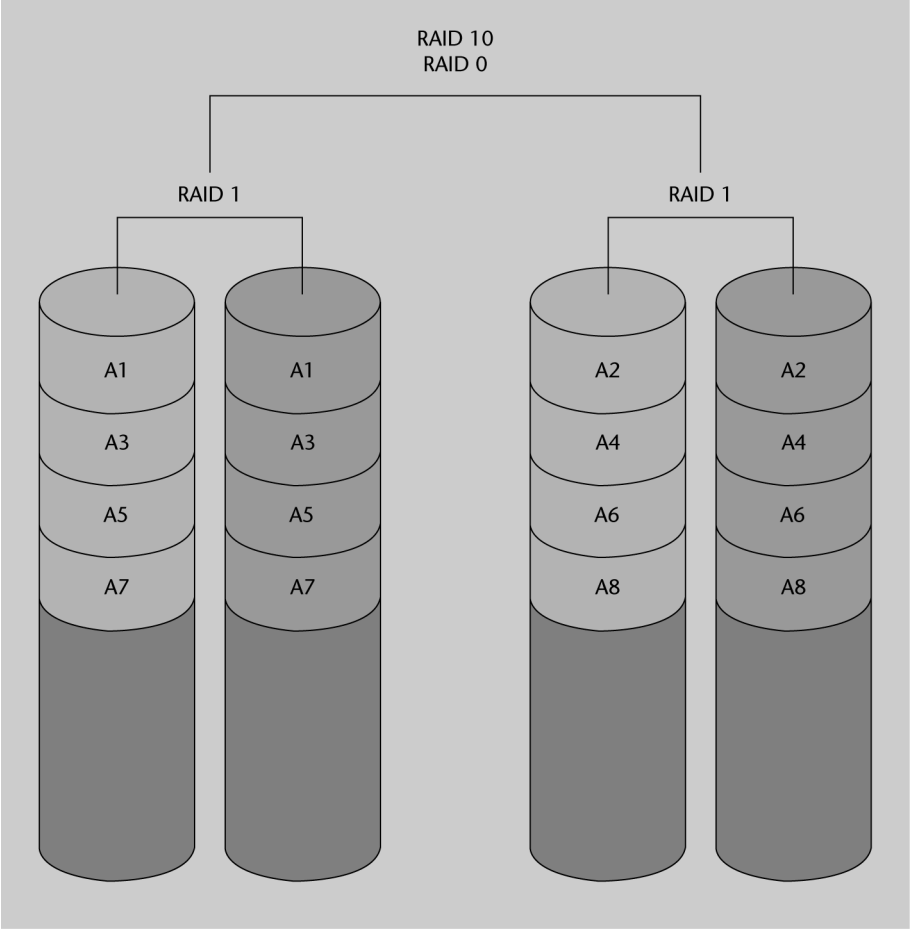

Figura 6

Algunas consideraciones a tener en cuenta sobre RAID en general:

- RAID mejora el *uptime* del sistema, ya que algunos de los niveles permiten que haya discos que fallan 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 permite que las operaciones lineales de lectura se incrementen significativamente, debido a la posibilidad de que los discos ofrezcan simultáneamente partes de esta lectura, aumentando la tasa de transferencia de datos.

- RAID no protege los datos, evidentemente la destrucción por otros medios (virus, malfuncionamientos generales, o desastres naturales) no está protegida. Hemos de basarnos en esquemas de copias de seguridad.

- 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, aunque sean de escritorio, debido a que estas aplicaciones tienen componentes altos de acceso aleatorio a datos, y a conjuntos de datos pequeños, por lo que puede que no se beneficien de lecturas lineales o de transferencias de datos sostenidas. En estos ambientes es posible que apenas se note mejoría de prestaciones.

- 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), mover un *array* de discos a otro sistema.

En el caso de GNU/Linux, se da soporte al hardware RAID mediante diversos módulos de *kernel*, asociados a diferentes conjuntos de fabricantes o circuitos base, *chipsets*, de estas controladoras RAID. Permitiendo así al sistema abstraerse de los mecanismos hardware y hacerlos transparentes al sistema y al usuario final. En todo caso 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, con la finalidad de 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 llamado Multiple Device (md), que podemos considerarlo como el soporte mediante *driver* del *kernel* para RAID. Mediante este *driver* podemos implementar niveles de RAID generalmente 0,1,4,5 y anidados (por ejemplo, RAID 10) sobre diferentes dispositivos de bloque como discos IDE 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 modulos md activos (además de algunos *drivers* específicos según el caso (ver *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 (o bien durante la instalación) o bien mediante la utilidad mdadm. Esta utilidad nos permite crear los *arrays* y gestionarlos.

Veamos algunos ejemplos (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 crea 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 /media/discoRAID):

```
# mkfs.ext2fs /dev/md0
# mount /dev/md0 /media/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 reconstrucción inicial 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 consideración importante son las optimizaciones a que se pueden someter los *arrays* RAID para mejorar su rendimiento, tanto por monitorizar su comportamiento por optimizar parámetros del sistema de ficheros, como 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, se recomienda examinar algunas cuestiones en:

Software-RAID-Howto, o en la propia página man de mdadm.

7.2. Volúmenes Lógicos (LVM)

Por otra parte surge la necesidad de abstraerse del sistema físico de discos, y su configuración, y numero 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 al sistema de volúmenes lógicos como una capa de virtualización del almacenamiento permitiendo una visión más simple que facilite la utilización fluida y sencilla.

En el kernel Linux se dispone de LVM (*logical volume manager*), que se basá a partir de ideas desarrolladas de gestores de volúmenes de almacenamiento usados en

HP-UX (una versión UNIX propietaria de HP). Actualmente existen dos versiones, siendo la LVM2 la más utilizada por una serie de prestaciones añadidas.

La arquitectura de una LVM consiste típicamente en los componentes (principales):

- **Volúmenes físicos** (PV): Son los discos duros, o particiones de éstos, o cualquier otro elemento que aparezca como un disco duro de cara al sistema (por ejemplo un RAID software o hardware).

- **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). Normalmente los volúmenes tienen más sentido para los administradores, ya que pueden usarse nombres para identifícarlos (así, podemos utilizar un dispositivo lógico, llamado stock, o marketing en lugar de hda6 o sdc3).

- **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 organicen en diferentes unidades lógicas LV.

Por ejemplo, en la siguiente figura, observamos un grupo de volúmenes, donde disponemos de 7 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) :

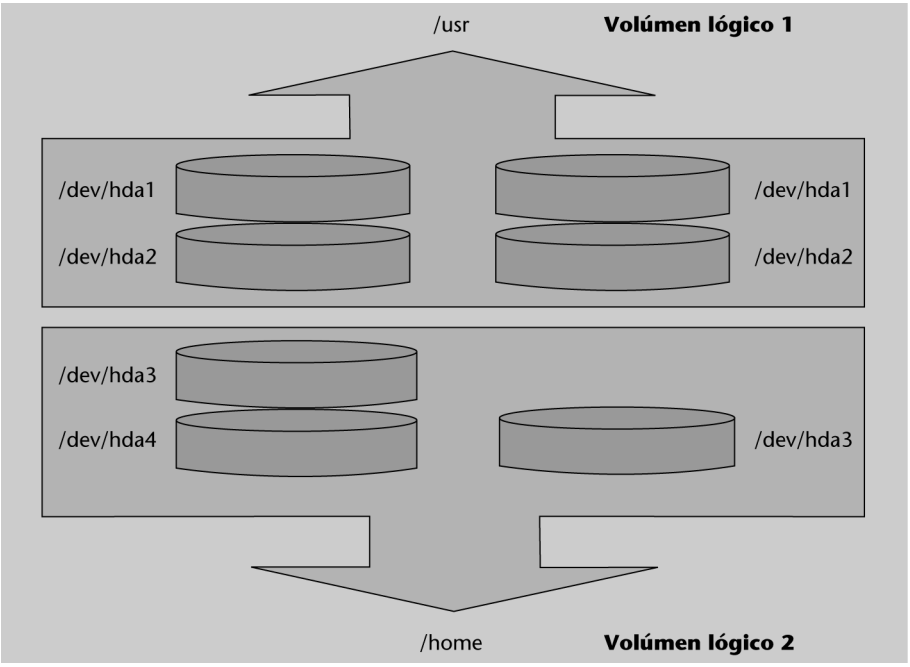

Figura 7. Esquema de un ejemplo de LVM

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, y poder 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:

- Redimensionar grupos y volúmenes lógicos, aprovechando nuevos PV, o extrayendo algunos de los disponibles inicialmente.

- Instantáneas del sistema de archivos (lectura en LVM1, y lectura y/o escritura en LVM2). Esto permite crear un nuevo dispositivo que sea una instantánea en el tiempo de la situación de una LV. Asimismo, permite crear la instantánea, montarla, probar diversas 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.

- RAID0 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), entonces o bien se utiliza software de RAID o controladora hardware RAID que lo implemente, y colocamos LVM como capa superior.

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 volúmenes físicos (PV). 2) la creación del grupo lógico (VG) y 3) la creación del volumen lógico, y finalmente la utilización para creación de un sistema de ficheros, y su posterior montaje:

1) ejemplo: disponemos de tres particiones de diferentes discos, creando tres PV e inicializando el contenido:

```
# dd if=/dev/zero of=/dev/hda1 bs=1k count=1
# dd if=/dev/zero of=/dev/hda2 bs=1k count=1
# dd if=/dev/zero of=/dev/hdb1 bs=1k count=1
# pvcreate /dev/hda1
Physical volume "/dev/sda1" successfully created
# pvcreate /dev/hda2
Physical volume "/dev/hda2" successfully created
# pvcreate /dev/hdb1
Physical volume "/dev/hdb1" successfully created
```

2) colocación en un VG creada de los diferentes PV:

```
# vgcreate grupo_discos /dev/hda1 /dev/hda2 /dev/hdb1
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
```

Y finalmente, creamos un sistema de ficheros (un *reiser* en este caso):

```
# mkfs.reiserfs /dev/grupo_discos/volumen_logico
```

El cual, por ejemplo, podríamos colocar de espacio de *backup*

```
mkdir /mnt/backup
mount -t reiserfs /dev/grupo_discos/volumen_logico /mnt/backup
```

Disponiendo finalmente del dispositivo como un volumen lógico que implementa un sistema de ficheros del nuestra máquina.

8. Software: actualización

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 a través del comando *rpm*. Contienen información de dependencias del software con otros. A alto nivel mediante *Yum* (o *up2date* en algunas distribuciones derivadas de Red Hat).

- DEB: paquetes de Debian, se suelen manejar con un conjunto de herramientas que trabajan a diferentes niveles con paquetes individuales o grupos. Entre éstas, cabe mencionar: dselect, tasksel, dpkg, y apt-get.

- 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 éstos para la descompresión). Estos paquetes no contienen información de dependencias y normalmente 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, normalmente, 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.

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 en Fe-

dora, 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 sólo compresión sin ningún tipo de dependencias.

6) Sólo código fuente, en forma de paquete o bien fichero comprimido.

9. Trabajos no interactivos

En las tareas de administración, suele ser necesaria la ejecución a intervalos 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 diversos 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. Normalmente, al salir de la cuenta, el usuario pierde sus procesos; nohup permite dejarlos en ejecución, a pesar de que el usuario se desconecte.

- at nos permite 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*). Ejemplos:

  ```
  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). Por ejemplo, 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 7.º día de la semana), o mensualmente (el 1.º de cada mes). Normalmente, los trabajos serían ejecutados por el comando crontab, pero el sistema cron supone que la máquina está siempre en-

cendida, 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 habitualmente 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 fueran una extensión del fichero /etc/crontab.

10. Taller: prácticas combinadas de los diferentes apartados

Comenzaremos por examinar el estado general de nuestro sistema. Vamos a hacer los diferentes pasos en un sistema Debian. Se trata de un sistema Debian *unstable* (la versión inestable, pero más actualizada); sin embargo, 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 un Pentium 4 a 2.66 Ghz con 768 MB y varios discos, DVD y grabador de CD, además de otros periféricos, pero ya iremos obteniendo esta información paso a paso.

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 "levantado" desde que se arrancó la última vez, 2 horas 46 minutos, en nuestro caso tenemos cinco usuarios. Éstos no tienen por qué ser cinco usuarios diferentes, sino que normalmente 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 2007
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 De-

bian, y para máquinas 686 (arquitectura Intel 32bits). Indica también que estamos arrancando un sistema Debian, con este *kernel* que fue compilado con un compilador GNU gcc versión 4.1.2 y la fecha. 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 con cuántos MHz funciona la CPU, y por eso se utiliza esta medida de velocidad. Después nos da más datos de la memoria principal, y 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 ésta 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 detección de 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. 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 UTC 2007
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 sis-

tema... 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 ismod, que nos permitirá saber qué módulos se han cargado en 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 básicamente tenemos los controladores para el hardware que hemos detectado, y otros relacionados o necesarios por dependencias.

Ya tenemos, pues, una idea de cómo se ha 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 (sólo se han sacado 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. Manejo de *buffers* del sistema (bdflush). Manejo de *journal* de *filesystem* (kjournald), manejo 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, esto indica que esta máquina es cliente de un servidor DHCP, para obtener su IP. Syslogd, un *daemon* que envía mensajes al log. El *daemon* de cups, como vimos, 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
```

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+Fx (o Ctrl+Alt+Fx 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... (ver man de ps). Otro comando muy interesante es top, que hace lo mismo que ps pero de forma dinámica, o sea, se actualiza con cierto intervalo, podemos clasificar los procesos por uso de CPU, de memoria, y también nos da información del estado de la memoria global.

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:

Nota

Ver man de los comandos para interpretar las salidas.

```
# 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. Esto puede parecer mucho, pero dependerá de las aplicaciones que se estén ejecutando.

Actividades

1) 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?

2) Supongamos que tenemos un sistema con dos particiones Linux, una / y la otra de swap. ¿Cómo solucionaríamos 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 solucionaríamos?

3) Instalad el sistema de impresión CUPS, definir nuestra impresora para que funcione con CUPS y probar 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?

4) Examinad la configuración por defecto que traiga el sistema GNU/Linux de trabajos no interactivos por cron. ¿Qué trabajos y cuándo se están realizando? ¿Alguna idea para nuevos trabajos que haya que añadir?

5) Reproducid el análisis del taller (más los otros apartados de la unidad) sobre la máquina que se disponga, ¿se observan en el sistema algunos errores o situaciones anómalas? En tal caso, ¿cómo las corregimos?

Otras fuentes de referencia e información

[Wm02] [Fri02] [Smi02] Libros de administración de GNU/Linux y UNIX, donde se comentan de forma amplia aspectos de administración local y gestión de sistemas de impresión.

[Gt] Se puede encontrar 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, podemos dirigirnos a http://www.linuxprinting.org/.

[Hin][Koe] Encontramos información sobre los diferentes sistemas de ficheros disponibles y los esquemas de creación de particiones para la instalación del sistema.

Recurso: Administración avanzada de GNU/Linux Descripción: Manual de Administración avanzada de GNU/Linux Idioma: ES Categoría: Cine Fecha de alta: 2011-10-19 11:10:30.0