



Curso de Linux Básico

ACM Capítulo de Estudiantes
Facultad de Informática - UPM

2, 3 y 4 de noviembre de 2004

Curso de Linux Básico

©2001-2004 ACM Capítulo de Estudiantes - Facultad de Informática UPM
ACM Capítulo de Estudiantes
Facultad de Informática - Universidad Politécnica de Madrid
Campus de Montegancedo s/n
28660 Boadilla del Monte
MADRID (SPAIN)

Esta obra puede ser distribuida únicamente bajo los términos y condiciones expuestos en **Creative Commons Reconocimiento-CompartirIgual 2.0** o superior (puede consultarla en <http://creativecommons.org/licenses/by-sa/2.0/es/>).

Índice general

1. Introducción	1
1.1. ¿Qué es Linux? Reseña histórica	1
1.1.1. Historia de Unix	1
1.1.2. La aparición de Minix	1
1.1.3. Linus Torvalds empieza a desarrollar Linux	1
1.1.4. Linux en la actualidad	2
1.2. Ventajas de GNU/Linux	2
1.3. Características de GNU/Linux	3
1.3.1. La organización de los directorios en GNU/Linux	3
2. Shell básica	5
2.1. Unas palabras sobre la notación	5
2.2. Introducción a la shell	5
2.2.1. Definiciones básicas	5
2.3. La shell bash	6
2.3.1. Introducción	6
2.3.2. Variables de entorno	7
2.3.3. Redirección de entrada y salida estándar	7
2.3.4. Pipes	8
2.3.5. Ejecución de comandos en segundo plano	8
2.3.6. ¿Cómo conseguir más información?	8
2.4. ¿Cómo obtener más ayuda?	8
2.4.1. Páginas de manual (man)	8
2.4.2. HOWTOs y guías	8
2.5. Moviéndose en el sistema de archivos	9
2.5.1. Tabla de referencia	9
2.5.2. Buscando archivos	9
2.6. Trabajar con archivos de texto	9
2.6.1. Mostrar contenido de archivos con cat y less	9
2.6.2. Encontrar cadenas de texto con grep	9
2.6.3. Edición básica con vi y pico	10
2.7. Juntando y comprimiendo archivos	11
2.7.1. tar	11
2.7.2. gzip	11
2.7.3. zip y unzip	11
2.7.4. Otros compresores/descompresores	11
2.8. Algunas utilidades de sistema	11
2.8.1. ps	11
2.8.2. kill	12
2.9. df	12
2.10. mount y umount	12
2.11. Utilidades de red/Internet	12
2.11.1. lynx	12
2.11.2. pine	12
2.11.3. ftp, ncftp	12

3. Kernel: el núcleo de Linux	13
3.1. Introducción	13
3.1.1. Definición de kernel	13
3.1.2. Tipos de kernel	14
3.2. El kernel de Linux	14
3.2.1. Origen y filosofía	14
3.2.2. Dónde conseguirlo	14
3.2.3. Numeración de versiones	15
3.2.4. Módulos. Estructura del kernel de Linux	15
3.2.5. Otros núcleos de GNU	15
3.3. Compilación del kernel	15
3.3.1. Introducción teórica.	15
3.3.2. Configuración previa a la compilación	16
3.3.3. El proceso de compilación	18
3.3.4. Instalación tras la compilación	19
3.4. Los módulos en detalle	24
3.4.1. Introducción a los módulos de un núcleo	24
3.4.2. Gestión de módulos	24
4. Gestión de redes con Linux	27
4.1. Introducción a las redes de computadores	27
4.1.1. Ventajas de la puesta en común de recursos	27
4.1.2. Resumen de una sesión FTP	27
4.1.3. Instalación de un servidor FTP	27
4.2. Terminal remota: Telnet	27
4.2.1. Cliente de telenet	27
4.2.2. Servidor de Telnet	29
4.3. Secure Shell: ssh	30
4.4. Login Gráfico	30
4.5. Sistema remoto de ficheros: nfs	30
4.5.1. Configuración de un sistema NFS	31
4.6. Servidor de impresión: apsfilter	32
4.7. Compatibilidad con Windows: Samba	32
4.8. Arranque del sistema vía NFS	32
4.8.1. Configuración del servidor	32
4.8.2. Configuración del cliente	33
4.9. Configuración de la red:	35
4.9.1. ifconfig	35
4.10. Enlace con la red de redes, Internet	37
4.10.1. A través de un módem: pppconfig	37
4.10.2. A través de un router: route	37
5. Prácticas en Linux	41
5.1. Introducción	41
5.2. Instalación del gnat	41
5.2.1. Instalación con apt-get en debian	41
5.2.2. Instalación con dpkg	41
5.2.3. Instalación a partir del paquete tar.gz	43
5.2.4. Uso del comando gnatmake	44
5.3. Instalación de GTK-Ada	45
5.3.1. Instalación compilado las fuentes (tar.gz)	45
5.4. Instalación del interprete para Haskell (Hugs)	46
5.4.1. Instalación con apt-get en debian	46
5.4.2. Instalación con rpm otras distribuciones	47
5.4.3. Manejo básico del Hugs	47
5.4.4. Instalación del Haskell-Mode para Emacs	47
5.5. Creación de Makefiles para la compilación	48

6. Trabajando y disfrutando	49
6.1. Introducción	49
6.2. El escritorio de KDE y sus aplicaciones	49
6.2.1. Konqueror, la navaja suiza	52
6.2.2. Correo con Kmail y Kontact	54
6.2.3. Escuchando música con JuK y amaroK	55
6.2.4. Mensajería instantánea con Kopete	56
6.2.5. Al tostadero con K3b	58
6.2.6. Más allá de lo básico: Opciones y configuraciones	60
6.3. OpenOffice.org	62
6.3.1. Introducción	62
6.3.2. Introducción a Writer (procesador de textos de OpenOffice).	62

Índice de figuras

3.1. Ejemplo de configuración del kernel en modo texto. <code>make config</code>	17
3.2. Ejemplo de configuración del kernel con ventanas en modo texto. <code>make menuconfig</code>	17
3.3. Ejemplo de configuración del kernel utilizando la versión para QT. <code>make xconfig</code>	18
4.1. Resumen de los comandos ftp	28
4.2. Configurando la conexión con pppconfig	38
6.1. Escritorio inicial con Kde	50
6.2. Menu de kicker	51
6.3. Konqueror como gestor de ficheros	52
6.4. Konqueror como navegador web	53
6.5. Kmail en plena acción, como parte de Kontact	54
6.6. El resumen de Kontact	55
6.7. amaroK en acción	56
6.8. La ventana de charla de Kopete	57
6.9. La ventana de contactos de Kopete	57
6.10. La ventana principal de K3b	58
6.11. Añadiendo ficheros a un CD de datos	59
6.12. Grabando un CD	60
6.13. Konqueror con aspecto KDE-XP	61
6.14. Cambiando las combinaciones de teclas	61
6.15. Open Office Writer	63
6.16. Open Office Estilo	63
6.17. Open Office Navegador	64

Capítulo 1

Introducción

1.1. ¿Qué es Linux? Reseña histórica

La historia de Linux está muy ligada a UNIX, por tanto vamos a empezar viendo un poco de la historia de este último.

1.1.1. Historia de Unix

A mediados de los de la década de los 60 varias grandes empresas norteamericanas se juntaron para tratar de hacer un sistema operativo de gran potencia al que denominaron MULTICS. El proyecto fue un fracaso pero uno de los programadores del MIT que había trabajado en el proyecto, Ken Thompson, y un grupo de colaboradores decidieron escribir una versión miniatura de MULTICS. Unos de los compañeros de Ken, Brian Kernigham, en una reunión de equipo, bromeando llamó al sistema de Ken Thompson UNICS.

UNICS fue un gran éxito y Ken decidió que UNIX era un nombre más atractivo que UNICS. Había nacido UNIX.

Un famoso artículo del año 1974 que describía UNIX atrajo la atención de las universidades que solicitaron el código fuente para estudiarlo y explicarlo en las aulas. Muy pronto, UNIX logró una gran aceptación en la comunidad científica y el interés por este sistema operativo comenzó a extenderse. A partir de este momento comienza una verdadera avalancha de versiones del sistema, lo que en un principio empezó como un proyecto de investigación se convirtió más tarde en un gran negocio.

Las más importantes de todas las versiones de UNIX fueron la BSD (Berkeley Software Distribution), de la Universidad de California en Berkeley, que contenía una serie de mejoras que hicieron a UNIX un sistema operativo más amigable, y la System V. Actualmente el System V es considerado el estándar de UNIX, ya que toda la industria se ha agrupado entorno a él.

1.1.2. La aparición de Minix

A pesar del éxito comercial de UNIX y de su aceptación como sistema operativo, el código fuente de UNIX no podía ser explicado en aulas universitarias, de modo que el desarrollo de sistemas operativos volvía a ser una ciencia restringida a un reducido grupo de empresas y personas.

Ante esta situación, el conocido profesor Andrew Tanenbaum decidió imitar a Ken Thompson cuando escribió el código de UNIX basándose en MULTICS, e inspirándose en UNIX llevó a cabo un nuevo sistema operativo mucho más reducido, al que llamó MINIX (de Mini-UNIX). MINIX había sido desarrollado en una IBM PC y ofrecía las mismas llamadas al sistema que UNIX V7. Tanenbaum hizo público el código de MINIX, y su texto aún se usa en la mayoría de las universidades del planeta para enseñar las bases del diseño de sistemas operativos.

1.1.3. Linus Torvalds empieza a desarrollar Linux

A principios de los 90 Linus Torvalds, un estudiante universitario, decidió crear su propia versión de Unix, que superara las limitaciones del sistema minix. Este sistema lo llamó Linux (Contracción de Linus y Unix). Pocos meses después publicó su primera versión, numerada como versión 0.01. Esta versión solamente contenía un kernel (núcleo del sistema operativo) muy rudimentario. El 5 de Octubre de 1991 fue creada y publicada la versión 0.02 cuando Torvalds logró ejecutar programas como el Bash y el Gcc (básicos para un sistema unix). Por aquel entonces decidió difundirlo pública y gratuitamente (bajo la licencia GNU, de la que hablaremos mas adelante) e invitó a todo aquel que pudiera a aportar ideas nuevas y a mejorar el código via Internet. Gracias a estos aportes Linux comenzó a evolucionar rápidamente.

A partir de ese momento, y a pesar de que Linus sigue teniendo la última palabra sobre el contenido del kernel o núcleo del sistema operativo, el proyecto se a ido abriendo a la comunidad con la participación de programadores de todo el mundo, que reproduciendo a una escala mucho mayor el esquema seguido por gran parte de las aplicaciones GNU.

Con el paso del tiempo se han ido sucediendo las versiones, y Linux ha ido tomando cuerpo como una muy seria alternativa a los demás sistemas operativos, tanto Unix tradicionales como Windows. Así, los sistemas Linux actualmente representan una gran amenaza competitiva para los desarrolladores de sistemas operativos propietarios. Microsoft ha comenzado a advertir la creciente popularidad de Linux y el riesgo que esto representa para su dominio del mercado. Linux se encuentra actualmente en el segundo lugar de las mayores amenazas para Microsoft, según lo anunció la empresa, siendo las *condiciones económicas* su amenaza número uno.

1.1.4. Linux en la actualidad

La creciente popularidad de Linux se debe -entre otras razones- a su extraordinaria estabilidad, al acceso libre al código fuente (lo que permite personalizar el funcionamiento) y a la abundancia de documentación relativa a los procedimientos. Día a día, más y más programas están disponibles para este sistema, y la calidad de los mismos aumenta de versión a versión. La gran mayoría de los mismos vienen acompañados del código fuente y se distribuyen gratuitamente bajo terminos similares a los del núcleo.

Hoy en día Linux es distribuido a través de múltiples “distribuciones”. Existen numerosas distribuciones de Linux (también conocidas como “distros”), ensambladas por empresas (caso de SuSE, RedHat), como fundaciones (caso de Debian o Gentoo) e incluso administraciones públicas (caso de Linex y Guadalinux en España). Cada distribución suele incluir software adicional, incluyendo software que facilita la instalación del sistema.

La base del sistema de cada distribución es muy parecida, incluyendo varios paquetes básicos del proyecto GNU, incluyendo un intérprete de comandos y utilidades como bibliotecas, compiladores y editores de texto. Estos componentes (esenciales para un sistema funcional) provienen del proyecto GNU, anterior a Linux. Es por esto que Richard Stallman (fundador del proyecto) pide a los usuarios que se refieran a dicho sistema como GNU/Linux. A pesar de esto, la mayoría de los usuarios continúan llamando al sistema simplemente “Linux” las razones expuestas por Richard Stallman son eterno motivo de discusión¹.

1.2. Ventajas de GNU/Linux

- **Precio:** Debido a que su licencia es GNU, podemos descargarlo gratuitamente desde Internet o comprarlo a un precio muy asequible.
- **Requerimientos:** Actualmente los sistemas operativos necesitan mucha máquina y recursos del sistema para ejecutarse con fluidez, Linux, al poder funcionar exclusivamente en modo texto sin la necesidad de cargar un entorno gráfico puede ejecutarse en cualquier máquina a partir de un i386. Evidentemente si queremos sacarle todo el partido necesitaremos una máquina mucho más potente.
- **Estabilidad:** Al tener su núcleo basado en UNIX, hereda esa estabilidad que siempre ha caracterizado a los sistemas UNIX.
- **Seguridad:** A nivel de servidor podemos encontrar que la seguridad de GNU/Linux frente a otros servidores de código cerrado del mercado es mucho mayor.
- **Multitarea real:** Es posible ejecutar varios procesos simultáneamente.
- **Velocidad:** Debido a la multitarea real que incorpora, y que no es necesario cargar su entorno gráfico para ejecutar servicios o aplicaciones, hacen que su velocidad sea muy superior a otros sistemas operativos que necesitan imperiosamente cargar un montón de aplicaciones gráficas para su funcionamiento.
- **Código Fuente:** El paquete incluye el código fuente, lo que es posible modificarlo y adaptarlo a nuestras necesidades libremente. Sin olvidar todo lo que se puede aprender de su funcionamiento.
- **Entorno de Programación:** Es ideal para la programación, ya que se puede programar para otros sistemas operativos. Además está extensamente documentado, y existen infinidad de herramientas para el programador.
- **Crecimiento:** Su sistema de crecimiento, gracias a la licencia GNU, el código abierto, y la gran comunidad de miles de programadores, es de los más rápidos que existen en la actualidad.

¹En el mundo del software libre hay un montón de “eternas discusiones”, síntoma para muchos de la libertad de elección que proporciona.

1.3. Características de GNU/Linux

Todo el mundo ha oído hablar de la potencia y fiabilidad que hacen de GNU/Linux un maravilloso sistema operativo. Pero en realidad, *¿qué características hacen que GNU/Linux sea lo que es?*

En la historia de los sistemas operativos ha habido diversas soluciones a los diferentes problemas que se han ido encontrando. Éstos pueden ser, el de la concurrencia de procesos, la memoria compartida, la comunicación de procesos, el soporte a múltiples usuarios, etc . . .

Muchos conocemos las limitadas características de MS-DOS como sistema operativo monotarea y monousuario, que a algunos nos sirvieron en nuestras primeras andaduras en esto de la informática. Pero como todo, los sistemas operativos cambian y evolucionan con respecto a los requerimientos de sus usuarios. En los sistemas operativos actuales, sean de código libre o comerciales, hay algunas palabras que están en boca de todos. Multitarea, multiusuario, portabilidad, etc . . . *¿Qué quieren decir cada uno de estos términos?*

- **Multitarea.** La palabra multitarea describe la capacidad de ejecutar varios programas al mismo tiempo sin detener la ejecución de cada aplicación. Se le denomina multitarea prioritaria porque cada programa tiene garantizada la oportunidad de ejecutarse, y se ejecuta hasta que el sistema operativo da prioridad a otro programa, por medio del módulo planificador para que se ejecute. Linux y otros sistemas operativos multitarea consiguen el proceso de prioridad supervisando los procesos que esperan para ejecutarse, así como los que se están ejecutando. El sistema programa cada proceso para que disponga de las mismas oportunidades de acceso al microprocesador. El resultado es que las aplicaciones parecen estar ejecutándose al mismo tiempo (en realidad hay una demora de billonésimas de segundo entre una ejecución y otra). Linux también es capaz de gestionar máquinas con varios procesadores, obteniendo la capacidad de procesar paralelamente (a la vez) varios procesos (tantos como microprocesadores tenga la máquina). Esto es lo que se llama concurrencia real.
- **Multiusuario.** La idea de que varios usuarios pudieran acceder a las aplicaciones o a la capacidad de proceso de un único PC era una utopía hace relativamente pocos años. La capacidad de Linux para asignar el tiempo de microprocesador simultáneamente a varias aplicaciones ha derivado en la posibilidad de ofrecer servicio a diversos usuarios a la vez, ejecutando cada uno de ellos una o más aplicaciones. La característica más relevante de Linux y sus funciones de multiusuario y multitarea, es que más de una persona puede trabajar con la misma máquina simultáneamente desde el mismo terminal o desde terminales distintos. No se debe confundir esto con el hecho de que muchos usuarios puedan actualizar el mismo archivo simultáneamente, característica que es potencialmente confusa, ya que es otro tipo de problema de concurrencia.
- **Shell programable.** El shell programable es otra característica que hace que UNIX, y por tanto GNU/Linux, sea lo que es: el sistema operativo más flexible de los existentes. Dentro del marco del shell hay un nuevo mundo accesible a todos aquellos que sean lo suficientemente aventureros como para dominar los entresijos de la sintaxis de los comandos de GNU/Linux. La programación del shell de GNU/Linux ofrece tantas funciones como personas deseen utilizarlas. Muchos utilizan esta característica para personalizar su sistema y hacerlo más amigable. Otros lo encuentran muy útil para simplificar muchas de las aplicaciones que ejecutan, permitiéndoles realizar una serie de procesos en segundo plano (background) para poder trabajar en otros.
- **Independencia de dispositivos.** Las características que Linux ha heredado de UNIX hacen que éste se aproveche de las mejores características de UNIX. Una de ellas es la adaptabilidad del kernel en cuanto a la visualización de dispositivos nuevos en el sistema, ya que el kernel de UNIX/Linux ve a estos dispositivos como enlaces, o simples controladores. No obstante, si existiese algún problema con algún dispositivo, y al tener el código libre, siempre se puede programar un controlador para un dispositivo que no exista, cosa que no puedes hacer en ningún otro sistema operativo comercial.
- **Comunicaciones y redes.** La superioridad de UNIX sobre otros sistemas operativos es igualmente evidente en sus utilidades y conexión en red. Linux no es ninguna excepción. Ningún otro sistema operativo incluye unas posibilidades de conexión en red tan ajustadamente acopladas y ningún otro sistema operativo posee la flexibilidad incorporada de estas mismas características. Internet nació y se creó en un mundo UNIX.
- **Portabilidad.** La portabilidad es simplemente la posibilidad de transportar un sistema operativo de una plataforma a otra sin que se vea alterado su comportamiento. En la actualidad, Linux es capaz de correr en múltiples plataformas, como pueden ser el i386, ALPHA, MAC, etc...

1.3.1. La organización de los directorios en GNU/Linux

Debido a la gran cantidad de distribuciones existentes se ha promovido el uso de la Filesystem Hierarchy Standard (www.pathname.com/fhs/). Esta guía permite al usuario encontrar los ficheros que buscan sin importar la distribución que tenga instalada.

El objetivo de esta guía es definir el propósito y función de cada directorio del sistema. Aunque la primera impresión resulta un poco caótica (cada nuevo programa instalado distribuye sus archivos por varios directorios) a la larga es una organización realmente ingeniosa.

La estructura de directorios más comunes y sus funciones son:

- **/**: es el directorio raíz², y a partir de él cuelgan todos los demás.
- **/bin**: contiene los ejecutables básicos del sistema.
- **/sbin**: contiene los ejecutables del administrador del sistema.
- **/usr**: contiene todo lo demás: programas, juegos, etc ...
- **/home**: aquí residen las “casas” de los usuarios del sistema en las cuáles guardan sus archivos y configuraciones personales. Por ejemplo el usuario “paquito” tendrá su casa en `/home/paquito`.
- **/root**: es el directorio de trabajo del administrador.
- **/etc**: contiene las configuraciones específicas del sistema, entre ellas las necesarias para arrancar. También contiene las configuraciones globales del software instalado.
- **/tmp**: se utiliza para almacenamiento temporal de información.
- **/var**: contiene archivos variables, como pueden ser las colas de impresión, etc ...
- **/boot**: es donde se encuentran los archivos de arranque, así como el kernel que se cargará en el inicio.
- **/dev**: contiene todos los dispositivos³ (devices).
- **/proc**: guarda la información sobre los procesos que hay en ejecución y sobre el sistema en general (CPU, memoria, etc ...).
- **/lib**: contiene la biblioteca de funciones necesarias para el compilador de C. Contiene las bibliotecas compartidas, esenciales en el sistema.
- **/media**: es donde se suelen encontrar los dispositivos externos como las unidades de CD o DVD.

²Recordemos que el sistema se organiza como un árbol invertido de directorios, por tanto “el raíz” es la base del árbol.

³En UNIX todo es un fichero. Esto significa que todos los dispositivos de hardware instalados (teclados, ratones, tarjetas de audio...) son ficheros que se pueden encontrar en `/dev`.

Capítulo 2

Shell básica

2.1. Unas palabras sobre la notación

En esta sección vamos a ver bastantes ejemplos y sintaxis de comandos. Es conveniente especificar una notación que permita describir la sintaxis de un comando de manera fácil.

Por ejemplo:

```
comando1 [arg1] arg2 {arg3|arg4} [arg5|arg6]
```

Significa que `comando1` acepta el argumento `arg1`, que el argumento `arg2` es obligatorio, se puede elegir entre `arg3` o `arg4` pero no los dos o ninguno. Por último, se puede elegir entre `arg5` o `arg6`, pero no los dos

2.2. Introducción a la shell

2.2.1. Definiciones básicas

La shell

La shell o intérprete de comandos es el interfaz básico que nos permite comunicarnos con el ordenador y arrancar los programas que guardamos en el disco duro.

Internamente no es más que un programa que básicamente espera a que nosotros le introduzcamos una orden o comando para ejecutarlo, tras lo cual vuelve a quedarse a la espera del siguiente comando. Esta funcionalidad básica es un punto común que tienen todos los intérpretes de comandos de cualquier sistema operativo. A partir de aquí, dependiendo de la shell y de los recursos que ponga a su disposición el sistema operativo, aparecen diversas extensiones que pueden cubrir desde una interfaz intuitiva y totalmente gráfica (escritorio de la familia de S.O de Microsoft) hasta el auténtico lenguaje de programación que encontramos en casi todas las shells de los S.O del entorno Unix. En esto último es en lo que nos vamos a centrar en este capítulo.

El prompt

El prompt de la shell es un indicativo de que la shell está lista para recibir nuevos comandos.

Variables de entorno

Prácticamente todas las shells ofrecen esta característica mediante la cual el usuario puede asignar o acceder a cierta información. Esta información es válida sólo mientras la shell que el usuario está usando continúe cargada en memoria. Cuando el usuario sale del sistema (logout) esta información se pierde.

El uso de estas no difiere de las variables encontradas en los lenguajes de programación aunque está en cierto modo simplificado. No hace falta declararlas y todas las variables almacenan valores del tipo string (cadena de caracteres). Al igual que con las variables de los lenguajes de programación encontramos operadores de comparación y asignación. La shell también ofrece otros operadores relacionados con el uso de variables de estado. Veremos esto más detalladamente en próximos capítulos.

Entrada y salida estándar. Redirección

La entrada y salida estándar son unas abstracciones que proporcionan al programador una forma unificada de leer y escribir datos desde su programa.

Supón por un momento que eres un programador que tienes que diseñar una aplicación que lea unos ciertos datos del usuario y muestre unos resultados. Uno podría utilizar las rutinas que el S.O. le ofrece y leer estos datos desde el teclado, para luego mostrar los resultados por pantalla. Sin embargo, como buen programador que eres, quieres darle a tu programa suficiente flexibilidad para que además de leer del teclado, también pueda leer los datos de un archivo. Y además, también quieres que sea capaz de imprimir los resultados por la impresora Esto requeriría la adición de nuevas rutinas para leer desde un archivo y para imprimir a una impresora. Y esto es sólo el principio, hay mil formas de leer datos y otras mil de mostrarlos.

Precisamente lo que se pretende con la entrada y salida estándar es que el programador no tenga que hacer nada de lo indicado arriba. Proporcionan dos canales únicos de entrada y salida que el programador puede utilizar en cualquier momento. El programador no sabe y no le preocupa como se han introducido los datos que vienen del canal de entrada estándar, y tampoco le interesa a donde van a parar los datos que vuelca sobre el canal de salida estándar. En lo que a él concierne, sólo está leyendo datos del usuario y mostrándole los resultados.

Todo esto se resume básicamente en lo siguiente: La entrada estándar es un canal por el cual un usuario le introduce datos a un programa. Este canal inicialmente apunta al teclado, pero (y aquí está lo importante) no tiene porque ser siempre así. El usuario puede hacer que la entrada estándar apunte a un archivo de datos o a un micrófono, por ejemplo. Lo mismo ocurre con la salida estándar. Esta apunta por defecto a la pantalla, pero un usuario puede manipularla de tal manera que apunte a un archivo, la impresora o los altavoces.

Al hecho de hacer que el canal de entrada o salida estándar apunte a otro sitio que no sea el dispositivo por defecto se llama redirección de la entrada y salida estándar. La redirección de la entrada y salida estándar es algo que se hace desde la shell, al mismo tiempo que se llama al programa. La shell proporciona una sintaxis adecuada para poder ejecutar un programa con la entrada/salida estándar redireccionada. Lo veremos con ejemplos en el próximo capítulo.

Tuberías (pipes)

Una tubería no es más que una forma de conectar la salida estándar de un programa con la entrada estándar de otro programa automáticamente. Es un concepto muy importante, ya que permite que varios programas puedan encadenarse y hacer que los resultados de un comando puedan tratarse como datos a procesar por el siguiente.

Ejecución en primer y segundo plano

Un hecho conocido de los sistemas Unix es que son S.O. multitarea y multiusuario. La característica principal de un S.O. multitarea es que puede aparentar que está ejecutando varios procesos (programas) a la vez (y digo aparentar porque un procesador convencional sólo puede ejecutar una instrucción a la vez).

Las shell de los sistemas Unix se aprovechan de esta capacidad ofreciendo al usuario dos formas de ejecutar sus comandos:

1. Ejecución síncrona (en primer plano): Es la forma más usual de ejecutar un comando. El usuario escribe el comando en el prompt de la shell, pulsa INTRO y espera a que se ejecute el comando. Sólo (importante) después de que el comando se halla ejecutado la shell muestra el prompt indicando que está preparada para el siguiente comando.
2. Ejecución asíncrona (segundo plano): Una vez que el usuario indica a la shell el comando que quiere ejecutar y pulsa INTRO, esta le devuelve el control (es decir, muestra el prompt) de inmediato. El comando introducido se está ejecutando en segundo plano, es decir, que el usuario puede seguir introduciendo comandos (que a su vez también pueden estar en segundo plano) mientras el primero se ejecuta.

2.3. La shell bash

2.3.1. Introducción

Esta shell seguramente es la de uso más extendido. Proporciona una interfaz al usuario muy configurable y potente.

El prompt de bash suele variar de un sistema a otro, dependiendo de como este configurado. El prompt por defecto suele ser este: `nombre-sistema:dir-actual $` para los usuarios normales y `nombre-sistema:dir-actual #` para el administrador del sistema.

En mi ordenador por ejemplo es así:

```
tajo:~$
```

(yo suelo dar nombres de ríos a mis ordenadores ;-)

A partir de ahora, para notar que en un ejemplo hay que teclear el comando comando1 pondra de esta forma:

```
tajo:~$ comando1
```

Lo que significa que hay que teclear `comando1` en la shell y pulsar INTRO. Por supuesto, no hay que teclear `tajo:~$`, eso es solo para indicar que me encuentro en la shell.

2.3.2. Variables de entorno

En bash, una variable de entorno se asigna de la siguiente manera:

```
nombreVar=valor
```

En primer lugar hay que notar que no hay ningún espacio en la asignación. Estas asignaciones son erróneas y provocarán un error:

```
nombreVar =valor
nombreVar= valor
nombreVar = valor
```

También hay que tener en cuenta que las variables de entorno son sensibles a las mayúsculas y minúsculas: Si el valor que asignamos tiene espacios en blanco, la asignación se realiza de las siguientes maneras:

```
nombreVar='valor con espacios en blanco'
nombreVar="valor con espacios en blanco"
```

(nótese que los caracteres `'` y `"` son sólo delimitadores, no forman parte del valor).

Supongamos por ejemplo:

```
tajo:~$ inicioQuijote='En un lugar de la mancha....'
```

En bash hay una distinción entre la variable y el valor que contiene. Es una distinción muy importante que hay que saber. Podemos pensar en la variable como una cierta localización de memoria X en la que podemos guardar valores. Así, cuando escribimos:

```
tajo:~$ inicioQuijote='En un lugar de la mancha....'
```

estamos diciendo a bash que guarde en la localización de memoria X (la variable `inicioQuijote`) la cadena (el valor) `'En un lugar de la mancha....'`.

¿Cómo podemos acceder al valor de la variable `inicioQuijote`? Hay dos formas:

```
$inicioQuijote \'
```

o

```
${inicioQuijote}
```

2.3.3. Redirección de entrada y salida estándar

Supongamos que queremos redirigir la salida del comando: `echo "Hola caracola"` al archivo `saludos.txt`. Esto se hace de la siguiente manera:

```
tajo:~/pruebas$ echo "hola caracola" > saludos.txt
```

Análogamente, supongamos que tenemos un comando `comando1`, el cual tiene que leer una serie de datos del usuario. Para hacer que lea los datos de un archivo (`datos.txt`) en vez de por teclado:

```
tajo:~/pruebas$ comando1 < datos.txt
```

Supongamos que además tenemos un comando `comando2` que hace lo mismo que `comando1` pero que además muestra unos resultados. Haremos lo siguiente para que lea de `datos.txt` y muestre en `res.txt`:

```
tajo:~/pruebas$ comando2 < datos.txt > res.txt
```

2.3.4. Pipes

La sintaxis adecuada para encadenar la salida del comando `comando1` con la del comando `comando2` es:

```
comando1 | comando2
```

Se pueden hacer pipes entre varios procesos, de la siguiente forma:

```
comando1 | comando2 | comando3 . . . . .
```

Esto conecta la salida estándar de `comando1` a la entrada estándar de `comando2`, la salida estándar de `comando2` con la entrada estándar de `comando3`, y así ad infinitum (:D).

2.3.5. Ejecución de comandos en segundo plano

Imaginemos que queremos ejecutar el comando `comando1`, pero mientras este se ejecuta queremos seguir haciendo otras cosas, ya que `comando1` tarda mucho en ejecutarse. Esto se hace de la siguiente manera:

```
tajo:~/pruebas$ comando1 &
[1] 32933
tajo:~/pruebas$
```

Con lo cual `comando1` seguirá ejecutándose mientras nosotros trabajamos en otras cosas.

El número a la derecha de los corchetes es el PID (ver más adelante) del proceso. Una vez que el proceso termine de ejecutarse en segundo plano, la shell nos lo notificará con un mensaje.

2.3.6. ¿Cómo conseguir más información?

Lo que hemos visto hasta ahora no son más que conceptos básicos sobre bash. Más información se puede conseguir en la página de manual de bash (comando: `man bash`) o en su manual de referencia (<http://www.gnu.org/software/bash>)

La página de manual de bash también nos puede ser útil cuando queramos alguna referencia rápida (comando: `man bash`)

2.4. ¿Cómo obtener más ayuda?

2.4.1. Páginas de manual (man)

Cuando tengamos alguna duda sobre algún comando, generalmente podemos consultar su página de manual. Esto se hace así:

```
man [seccion] comando
```

Las páginas `man` se agrupan en secciones que contienen ayuda de diversos temas. Por ejemplo, la sección 1 es para programas de usuario. La sección 2 es para rutinas del kernel, etc

Para ver más información detallada siempre podemos escribir:

```
man man
```

Hay veces en las que un tema aparece en más de una sección. Podemos ver en que secciones aparece el comando `comando1` con:

```
man -k comando1
```

2.4.2. HOWTOs y guías

Los Howto's son textos que documentan los pasos necesarios para realizar con éxito alguna tarea (compilar el kernel, configurar las X, etc...). Son textos de orientación principalmente prácticos.

Las guías son textos más largos que los Howto's y que cubren casi totalmente un tema determinado (administración del sistema, red, etc . . .). La mayoría son mini-libros.

Tanto las guías como los howto's se pueden coger gratis y sin ningún cargo de <http://www.linuxdoc.org>, aunque lo más normal es que estén también en tu distribución (prueba en `/usr/doc/HOWTO`, o `/usr/doc/Linux-HOWTO`).

2.5. Moviéndose en el sistema de archivos

2.5.1. Tabla de referencia

MANDATO	DESCRIPCIÓN	NOTAS
ls	Lista los contenidos de un directorio	Una opción útil es <code>-l</code> (formato largo)
cd [dir]	Entrar en el directorio <code>dir</code>	Si no se especifica el directorio se vuelve al directorio de inicio
mkdir dir	Crea el directorio <code>dir</code>	Para poner espacios, englobar el nombre entre comillas
cp arch dir	Copia el archivo <code>arch</code> al directorio <code>dir</code>	Si el directorio <code>dir</code> es <code>.</code> se lleva al directorio actual
mv arch dir	Mueve el archivo <code>arch</code> al directorio <code>dir</code>	Si el directorio <code>dir</code> es <code>.</code> se lleva al directorio actual
rm arch	Borra definitivamente el archivo <code>arch</code>	¡¡Cuidado con lo que borras!!
rmdir dir	Borra el directorio definitivamente <code>dir</code>	El directorio tiene que estar vacío

2.5.2. Buscando archivos

Para buscar archivos utilizaremos el comando `find`:

La forma básica de buscar un archivo:

- Sin tener en cuenta mayúsculas y minúsculas: `find ruta_inicio -iname nombre_archivo_y_comodines`
- Teniendo en cuenta mayúsculas y minúsculas: `find ruta_inicio -name nombre_archivo_y_comodines`

En ambos casos, `find` se sitúa en `ruta_inicio` y va descendiendo recursivamente por todos sus subdirectorios.

2.6. Trabajar con archivos de texto

2.6.1. Mostrar contenido de archivos con `cat` y `less`

`cat` y `less` muestran por salida estándar el contenido de los archivos que les especifiquemos. La diferencia principal entre ambos es que `less` proporciona una salida paginada. Esto es, si tenemos un documento con cientos de líneas lo más probable es que no quepa entero en la terminal. `cat` pasará por pantalla todo el documento, pero sólo la última parte será legible en la terminal. `less` proporciona al usuario el texto del archivo página por página (entendiendo por página la cantidad de información que cabe en la pantalla de la terminal). Las teclas básicas para utilizar `less` son:

```
q          Salir
Espacio   Avanzar una p\`agina
Intro     Avanzar una l\`inea
```

`cat` y `less` se invocan de la siguiente manera:

```
cat archivo1 [archivo2] [archivo3] ...
less archivo1 [archivo2] [archivo3] ...
```

2.6.2. Encontrar cadenas de texto con `grep`

`grep` es una herramienta que nos permite encontrar ciertas cadenas de texto o patrones en un archivo de texto. Su uso más básico es:

```
grep cadenatexto archivotexto1 [archivotexto2] ...
```

El resultado de este comando son las líneas en las que aparece la cadena de texto `cadenatexto` en los archivos `archivotexto1`, `archivotexto2` ... Si `cadenatexto` contiene caracteres de espacio o tabuladores, tendremos que limitarla con comillas simples o dobles para evitar que la shell piense que son nombres distintos.

`grep` es una herramienta muy versátil y flexible. Es conveniente echarle un vistazo a la página de manual (comando: `man grep`).

2.6.3. Edición básica con vi y pico

Pico y vi son editores muy conocidos en el mundo UNIX. Vi es el editor estándar de UNIX (dejando a un lado ed) y es uno de los más potentes que existen (con permiso de GNU emacs).

Sin embargo, es difícil de aprender (sólo en principio) y hay personas que no se sienten cómodas con él. De esta necesidad nació pico, que es un editor muy cómodo e intuitivo. No es tan potente como vi.

Edición básica con vi

Para leer un archivo o editar uno nuevo: `vi archivo`.

Es importante saber que vi posee dos modos de edición: modo de comandos y modo de edición: El modo de comandos sirve para “hablar” con el editor, esto es, abrir archivos, salvar, buscar, etc.

El modo de inserción es el que se utiliza para introducir texto.

Nada más cargar vi este se pone en modo de comandos. Como lo primero que queremos es empezar a introducir texto o modificar el existente, pasaremos a modo de inserción; esto lo haremos pulsando una de las siguientes teclas: a, i ó tecla INSERT si nuestra terminal dispone de ella.

Una vez que hemos terminado de insertar/modificar/borrar texto, lo siguiente que queremos hacer es salvar los cambios; para ello deberemos regresar primero al modo de comandos pulsando ESC.

Una vez en modo de comandos, salvaremos con el comando `:w [ENTER]` (notar que ':' es parte del comando, no del texto).

- Para salir del editor utilizaremos el comando `:q [ENTER]`
- Si queremos salir sin salvar cambios utilizaremos el comando `:q! [ENTER]`
- El comando para buscar una cadena de texto es `/cadena [ENTER]`
- Para solicitar ayuda en pantalla el comando es `:help [ENTER]`
- Si queremos movernos a una línea en concreto utilizaremos `:num_lin [ENTER]`
- Si queremos borrar la línea actual pulsaremos `dd`
- Si queremos borrar las n líneas siguientes (incluida la actual): `n dd`

Aunque a primera vista pueda parecer críptico y poco intuitivo, vi es un editor que merece la pena. Es muy potente y no es de extrañar que sea el editor preferido de mucha, mucha gente.

Edición básica con pico

El editor pico no debería entrañar ningún tipo de problema al principiante. Parte de la pantalla esta ocupada con los shortcuts a los comandos más usuales.

En pico no hay distinción de modos como en vi. Todo lo que tecleas aparece por pantalla. Si quieres “hablar” con el editor hay que utilizar una combinación de teclas del tipo:

`control-letra (^ letra abreviado):`

- `^X` sale del editor
- `^O` guarda los cambios
- `^W` busca una cadena
- `^K` corta texto
- `^U` pega texto cortado previamente
- `^G` obtienes ayuda

2.7. Juntando y comprimiendo archivos

2.7.1. tar

tar es la forma estándar de hacer un volumen de archivos (un archivo que contiene varios archivos). Hay que notar que tar no comprime el volumen resultante. Somos nosotros los que elegimos el algoritmo de compresión mediante otro programa (normalmente gzip).

Sintaxis básica:

```
tar [OPERACIONES Y OPCIONES] archivos_involucrados
```

Operaciones básicas:

- Creación de un volumen: `tar cf archivo.tar archivo_o_dir1 [archivo_o_dir2] ...`
- Añadir archivos a un volumen: `tar rf archivo.tar archivo_o_dir1 [archivo_o_dir2] ...`
- Extraer archivos de un volumen: `tar xf archivo.tar [archivo_o_dir1] [archivo_o_dir2] ...`
- Listar los archivos contenidos: `tar tf archivo.tar`

2.7.2. gzip

gzip es el compresor por excelencia en cualquier sistema Unix (dejando a un lado compress).

Operaciones básicas:

- Comprimir un archivo: `gzip [-n] arch` (siendo n un número del 1 al 9), 1 más rápido, 9 más comprimido
- Descomprimir un archivo: `gzip -d archivo.gz`

También existe otro compresor, bzip2, que a pesar de ser más lento es bastante más eficiente. La sintaxis es prácticamente la misma, solo que el sufijo del archivo es .bz2.

En cualquier caso, la forma estándar de juntar y comprimir varios archivos consiste en hacer un tar con todos ellos y luego comprimir el archivo con gzip.

2.7.3. zip y unzip

Estas utilidades nos permiten trabajar con los archivos .zip, como era de esperar. Además de hacer un volumen de archivos, zip también los comprime (al contrario que tar).

Operaciones básicas:

- Comprimir un archivo: `zip archivo.zip archivo1 [archivo2] [archivo3] ...`
- Descomprimir un archivo: `unzip archivo.zip`
- Listar los contenidos de un archivo zip: `unzip -l archivo.zip`

Nota: también podemos descomprimir los archivos autodescomprimibles que crea Winzip para windows con este comando: `unzip archivo.exe`

2.7.4. Otros compresores/descompresores

Existen gran cantidad de programas encargados de comprimir y descomprimir los múltiples formatos que existen: unarj, rar, unrar, etc.

En la mayoría de los casos estos programas tendrán página de manual o aceptarán la opción -h, para informarnos de como se pueden utilizar.

2.8. Algunas utilidades de sistema

2.8.1. ps

ps muestra información sobre los procesos que están corriendo en el sistema. El modo por defecto es mostrar todos los procesos que un usuario tiene en la consola a la que esta conectado. También podemos modificar este comportamiento usando alguna de las opciones que ps acepta.

2.8.2. kill

El programa kill nos permite matar (terminar) un proceso que este corriendo en el sistema, siempre y cuando seamos nosotros quienes hemos ejecutado tal proceso.

```
kill PID
```

Podemos averiguar el PID de un programa mediante ps.

Si no sabemos el PID, o si queremos matar un grupo de procesos con el mismo nombre, podemos utilizar:

```
killall nombre_proceso
```

2.9. df

df sirve para averiguar el estado de las particiones:

```
tajo:~$ df
Filesystem          1k-blocks      Used Available Use% Mounted on
/dev/hda1            488006        341867   120941   74% /
/dev/hda2           1982544       1367280   512852   73% /usr
/dev/hda3            550616        324024   198620   62% /almacen
/dev/hdc             664900        664900         0 100% /cdrom
/dev/hdd            447202        447202         0 100% /cdrom2
tajo:~$
```

Una opción útil es -m, que muestra las capacidades en bloques de un mega, en vez de en bloques de un kb.

2.10. mount y umount

mount sirve para montar unidades (cdrom, dvd, particiones de discos duros) e incluirlas en el sistema de archivos. De esta forma, podemos navegar por ellas sin ningún problema.

Por ejemplo, vamos a montar nuestro cdrom:

```
tajo:~$ mount /cdrom
```

De esta forma, si ahora tecleamos:

```
tajo:~$ ls /cdrom
```

Obtendríamos un listado de los directorios del cdrom.

Una vez que hemos terminado de usar la unidad, es necesario desmontarla. Para ello, usamos umount.

```
tajo:~$ umount /cdrom
```

2.11. Utilidades de red/Internet

2.11.1. lynx

lynx es un navegador en modo texto. Puede sernos útil cuando no dispongamos de un entorno gráfico o cuando naveguemos en busca de información y las imágenes estén de más.

Su uso es bastante fácil, apareciendo una pequeña leyenda en la parte de abajo de la pantalla indicándonos las teclas de control.

2.11.2. pine

pine es un cliente de correo bastante intuitivo y fácil de usar. pine lee el correo de la máquina local, así que si queremos utilizar un servidor POP3 tendremos que utilizar una utilidad aparte (por ejemplo, fetchmail).

2.11.3. ftp, ncftp

- ftp es el cliente estándar FTP. Podemos ver una lista de comandos tecleando help en el prompt de este programa.
- ncftp es un cliente en modo texto bastante más elaborado que ftp. Entre sus características destacan: capacidad para mantener un historia de conexiones/servidores, autologin anónimo y soporte fácil para **resumes**.

Capítulo 3

Kernel: el núcleo de Linux

3.1. Introducción

La compilación del kernel constituye uno de los pasos que afianzan a cualquier usuario de Linux en el entorno en que se desenvuelve. Este proceso de refinamiento marca la diferencia fundamental entre los sistemas operativos libres y los propietarios: **“Tenemos el código, podemos verlo, podemos tocarlo, podemos modificarlo y aprender de él. Y si nosotros no lo hacemos ¿ Entonces quién ?”**. A lo largo de este capítulo se introducirá al lector en el proceso de compilación e instalación de un nuevo núcleo de Linux, también se tratarán temas de desarrollo de módulos y gestión de los mismos. Sin embargo el objetivo principal consiste en lograr que el núcleo se comprenda y no constituya un problema a la hora de implantar Linux en un entorno determinado.

Este documento no pretende sustituir a ninguno de los ya existentes acerca de la compilación del kernel. Humildemente, aún tenemos mucho que aprender de los famosos HOWTO ¹ de cualquier distribución y el dedicado a esta tarea es uno de los más antiguos y completos que existen.

No pretendemos proporcionar una sección que trate con absoluto rigor y detalle cada uno de los subprocesos involucrados en la compilación del kernel, el objetivo básico consiste en mostrar la comodidad con que puede ser llevado a cabo y las ventajas que reporta al administrador. Al mismo tiempo se mostrará la necesidad de la compilación del núcleo para obtener funcionalidades que no vienen en los kernels precompilados como el acceso a particiones NTFS o la configuración de grabadoras de CD.

Este capítulo está estructurado del siguiente modo: Esta primera sección proporciona el contexto necesario para la adecuada comprensión del proceso. En la segunda sección se mostrará una descripción del kernel de Linux, proporcionando información sobre dónde conseguirlo y dónde obtener información relevante. En la tercera sección se detalla el proceso de compilación paso a paso y en el detalle necesario. En la cuarta sección se introduce al lector en la gestión de módulos, proporcionando unas pequeñas nociones sobre desarrollo.

3.1.1. Definición de kernel

Según la teoría clásica de sistemas operativos entendemos un S.O. como un intermediario entre los programas de usuario y el hardware, es decir el software encargado de proporcionar el entorno necesario dónde será posible ejecutar otros programas. Un símil adecuado podrían ser los mandos del salpicadero de un coche a través de los cuales podremos manejarlo con la soltura suficiente. Cada aplicación de usuario se ejecuta sobre la máquina utilizando lo que llamamos proceso (que podría entenderse como un programa en ejecución). Pero es necesario ser consciente de que existe mucho bajo del concepto de proceso. No es necesario decir que nuestra máquina, realmente (como norma general) no posee más que un microprocesador, y este no es capaz de ejecutar más de una instrucción por ciclo ². Sin embargo todo el que ha trabajado con cualquier sistema operativo medianamente serio ha sido capaz de simultanear entre varias aplicaciones y tenerlas ejecutando en paralelo. ¿ Cómo es esto posible ? o mejor dicho ¿ Quién es el responsable de que esto ocurra ?.

Imaginemos por un momento una carrera de tortugas con varios carriles por dónde circularán para intentar obtener la victoria. Ahora, sin embargo, sólo existe un carril por dónde, evidentemente, sólo puede circular una tortuga en un momento determinado. En esta extraña competición se nos encarga la ardua tarea de gestionar ese único carril para que las tortugas puedan correr en él y efectuar una competición digna. Podríamos hacer lo siguiente: tomamos una tortuga al azar y la colocamos en la línea de partida, esta comenzará a moverse; tras un período fijado de tiempo anotamos su posición, la sacamos y tomamos de nuevo otra aleatoriamente para colocarla en la línea de partida. Si

¹Los HOWTO son un compendio de documentos que muestran como realizar ciertas tareas específicas dentro de un campo determinado, en este caso nos referimos a los referidos a Linux, que pueden encontrarse en el directorio `/usr/doc/HOWTO`

²Esto no es realmente cierto, véase procesadores superescalares, sin embargo para el objetivo del artículo es más que suficiente

utilizamos el mismo intervalo de tiempo y procuramos que todas las tortugas puedan acceder a la pista por igual (colocándolas haciendo cola por ejemplo) podríamos simular una carrera con un sólo carril.

Esta pequeña metáfora refleja una de las funciones del núcleo de un sistema operativo, el planificador de procesos. Pretendemos mostrar al sistema operativo como el software encargado de proporcionar el soporte necesario para habilitar la ejecución de aplicaciones y compartición de recursos como la memoria o la CPU (el único carril de nuestro circuito).

Podemos definir un kernel entonces como el software que constituye el núcleo del sistema operativo, dónde se realizan las funcionalidades básicas como la gestión de procesos, la gestión de memoria y de entrada salida. Pero entonces, ¿un kernel es en sí un sistema operativo? No realmente, depende de la definición de sistema operativo. Hay quién afirma que el shell también forma parte de él. Desde nuestro punto de vista la pregunta anterior no es cierta, dado que existen otros conceptos que iremos comprendiendo a lo largo del artículo y que constituyen, en sí mismos piezas fundamentales del sistema operativo.

3.1.2. Tipos de kernel

En función del tamaño y de las funcionalidades que posea el kernel podemos clasificarlo. Realmente, y pese a seguidores incondicionales en un modelo u otro, existe una tendencia básica a reducir el tamaño del núcleo proporcionando menos funcionalidades, que son desplazadas a módulos que se cargan en tiempo de ejecución.

En función a esta idea tenemos tres tipos fundamentales de kernel:

Kernel monolítico. Todas las funcionalidades posibles están integradas en el sistema. Se trata de un programa de tamaño considerable que deberemos recompilar al completo cada vez que queramos añadir una nueva posibilidad. Esta es la estructura original de Linux. Por tratarse de una técnica clásica y desfasada el creador de Linux fue muy criticado.

Kernel modular. Se trata de la tendencia actual de desarrollo. En el kernel se centran las funcionalidades esenciales como la administración de memoria, la planificación de procesos, etc. Sin embargo no tiene sentido que el núcleo de un sistema operativo englobe toda la parafernalia para comunicarse con todas las posibles de tarjetas de vídeo o de sonido. En otros sistemas operativos esto se soluciona con unos ficheros proporcionados por el fabricante llamados drivers. En Linux se creó un interfaz adecuado para posibilitar el desarrollo de módulos que cumplieran esas funcionalidades. Esos módulos pueden ser compilados por separado y añadidos al kernel en tiempo de ejecución.

Estructura de microkernel. Esta técnica pretende reducir a su mínima expresión el kernel, dejando a los niveles superiores el resto de las funcionalidades. Existen algunos kernels que lo utilizan, si bien el que centra nuestra atención es **Hurd**. Se trata del último kernel **GNU** llamado a sustituir a Linux como núcleo del sistema operativo. Aunque esta estrategia de diseño es tan antigua como la modular, no ha sido tenida en cuenta hasta ahora debido a las limitaciones de rendimiento que tenía.

3.2. El kernel de Linux

3.2.1. Origen y filosofía

Todos conocemos hoy día Linux, pero quizá lo que, personalmente más me atrae de todo es que fue un estudiante, exactamente con los mismos años que yo tengo ahora, el que tuvo la mayor idea jamás contada, que seguramente estaba basada en las fábulas que le contaron en la guardería: Decidió compartir el código, liberarlo para que el que quisiera lo viera y lo modificara a su antojo. Este hecho junto con la ambiciosa idea de potenciar un sistema operativo Unix bajo la plataforma Intel logro lo que todos hoy conocen como Linux, el sistema operativo de GNU. Pero, ¿qué fue lo que compartió Linus? el kernel, evidentemente. Cuento esto por que cuando uno habla del kernel de Linux no debe hacerlo como la molestia que debe ser tomada sino como la razón de que esté trabajando en este sistema operativo.

3.2.2. Dónde conseguirlo

Para conseguir el kernel no hay que esforzarse demasiado, en cualquier distribución de Linux viene alguna versión, algunas como Debian traen hasta 5 versiones diferentes. De todas formas la mejor fuente de la que nutrirse para obtener el kernel es www.kernel.org. Es importante darse cuenta de cuál es el kernel con el que estamos trabajando, dado que el proceso de compilación, como luego veremos, pese a ser extremadamente sencillo, resulta también crítico y puede tener consecuencias desastrosas.

Una vez hayamos conseguido el paquete será necesario descomprimirlo para obtener las fuentes. Si el paquete que instalemos es `.deb` o `.rpm` utilizaremos el mismo proceso de siempre (`dpkg -i linux-2.6.9.deb` ó `rpm -i`

linux-2.6.9.rpm). Dejarán el código en el directorio `/usr/src/linux` Si, por el contrario tenemos el paquete en modo `.tgz` será necesario usar el mandato `tar`, sería válido por ejemplo:

```
cp linux-2.6.9.tar.gz /usr/src
tar xvzf /usr/src/linux-2.6.9.tar.gz
```

También es bastante común encontrarlo en modo `.tar.bz2`. En este caso:

```
cp linux-2.6.9.tar.bz2 /usr/src
tar xvjf /usr/src/linux-2.6.9.tar.bz2
```

3.2.3. Numeración de versiones

El tipo de desarrollo que afronta el kernel (como todo proyecto software se debería someter a algún proceso de desarrollo) podría denominarse como evolutivo, en cierto modo. Esta en continuo cambio, siempre hay alguien trabajando en él, añadiendo dispositivos, adaptándolo a otras arquitecturas, etc. Sin embargo, cuando el número de usuarios creció considerablemente, fue necesario establecer de algún modo ciertas reglas de convivencia entre los desarrolladores y los usuarios de Linux. Para ello se dotó a cada línea base del kernel de una numeración determinada que lo clasifica. Esta numeración se compone de tres dígitos, por ejemplo, el último kernel liberado mientras se escribía este capítulo era 2.6.9. De todos estos dígitos debemos prestar atención al segundo, dado que si se trata de un número impar (2.3.13, 2.5.72 ...) se tratará de un kernel en desarrollo y, consecuentemente, inestable. Por otro lado, un número par confirma el kernel como estable y apto para su compilación y uso (2.0.36, 2.2.17, 2.4.9, 2.6.7 ...). No hay razón para no utilizar un kernel estable a no ser que estemos experimentando con él, por que seamos desarrolladores del kernel de Linux, por ejemplo.

3.2.4. Módulos. Estructura del kernel de Linux

Originalmente Linux era monolítico, es decir, todas las funcionalidades estaban incluidas en el código del núcleo y era necesario recompilarlo para soportar un nuevo dispositivo, etc. Sin embargo, esta idea no encaja con la enorme diversidad de componentes hardware que existen. Raro es que todo el mundo posea los mismos componentes en su ordenador y Linux, como buen sistema operativo Unix pretende obtener todo el partido de la máquina en la que se está ejecutando. Debido a todo esto, el diseño fue migrando paulatinamente a un modelo basado en módulos. Se procura así que el núcleo sea lo más ligero posible y cuando sea necesario añadir una nueva funcionalidad como soportar una nueva tarjeta de sonido, sólo haya que compilar el módulo y añadirlo al núcleo.

3.2.5. Otros núcleos de GNU

Actualmente está en desarrollo otro núcleo, llamado a ser el verdadero kernel de GNU. Esto es por que ha sido desarrollado y arropado desde su concepción por la FSF³. Se trata de HURD, un microkernel, descendiente del clásico Mach creado a gusto de los puristas, a conciencia y bien hecho. No olvidemos que el kernel de Linux, ahí dónde está, ha sido muy criticado por su falta de documentación y recursos para lograr que más gente colabore. Lo mejor de todo es que los desarrolladores de HURD están logrando que gran parte del software que funciona bajo Linux lo haga también con su núcleo.

3.3. Compilación del kernel

Una vez que sabemos qué es el kernel y cuál es su importancia, vamos a entrar en detalle en el proceso de compilación. Es necesario evitar dos grandes errores a la hora de realizar este proceso, tener miedo de las consecuencias y tomárselo a la ligera (para aquel que esté acostumbrado a pulsar `intro` antes de leer, entre los que me incluyo, será mejor que pierda el hábito).

3.3.1. Introducción teórica.

Puede que aún alguien dude de porqué es necesario compilar el kernel, propongo este ejemplo: comparemos el kernel de Linux con un coche. Podemos ir al concesionario y comprar un modelo estándar (con complementos de serie) proporcionado por el fabricante. Sin embargo, nosotros somos muy selectivos y tenemos absolutamente claro qué es lo que queremos y qué no. Para ello, nos acercamos al concesionario y solicitamos, de una lista de propiedades, cuales debería tener nuestro futuro vehículo. Tras un tiempo tendremos un coche a la medida en casa. Pues bien, este proceso es exactamente el mismo que realizamos cuando compilamos el kernel. Debemos ser consciente de que el

³Free Software Foundation

mundo es muy grande y la gente posee máquinas muy dispares, como ingenieros preferimos tomar una idea general y adaptarla a nuestras necesidades que tomar algo muy particular que luego no nos sirva. Este proceso es conocido como ajuste o **tunning** y consiste en adaptar el software al hardware sobre el que se ejecuta.

3.3.2. Configuración previa a la compilación

Realmente, este es el paso decisivo a la hora de compilar el kernel. Esencialmente se trata de un programa como otro cualquiera y debe ser compilado. Supongamos que estamos desarrollando una maravillosa práctica que, por cierto, nos trae de cabeza. Para una correcta depuración añadimos unas cuantas sentencias que imprimen por pantalla mensajes indicativos del punto por el que va el programa. Una vez logramos que la práctica satisfaga todos los requisitos propuestos en el enunciado decidimos entregarla, pero, pese a que no queremos que en la ejecución final aparezcan los mensajes de depuración tampoco queremos quitarlos (nadie nos ha aprobado aún). Podríamos tener una constante en nuestro programa que distinguiera entre una versión de entrega y una de evaluación (por ejemplo `DEBUG_MODE = 1`). Las funciones que imprimen los mensajes consultarían esa constante antes de soltar el mensaje por pantalla. Para la entrega final colocamos el modo de depuración en modo entrega (`DEBUG_MODE = 0`) y obtenemos una versión oficial de la práctica. Un ejemplo del código necesario para esto sería:

```
#define DEBUG_MODE

#ifdef DEBUG_MODE
printf(“Salgo del bucle”);
#endif
```

Pues bien, el proceso de configuración previo sobre el código del kernel es exactamente el mismo que sobre nuestra hipotética práctica. Podemos ver un fragmento del código del fichero `main.c` situado en el subdirectorio `init` donde se muestra el paralelismo con nuestra práctica:

```
#ifdef CONFIG_ATARIMOUSE
extern void atari_mouse_setup (char *str,int *ints);
#endif
#ifdef CONFIG_DMASOUND
extern void dmasound_setup (char *str, int *ints);
#endif
#ifdef CONFIG_ATARI_SCSI
extern void atari_scsi_setup (char *str, int *ints);
#endif
extern void wd33c93_setup (char *str, int *ints);
extern void gvp11_setup (char *str, int *ints);
```

Sin embargo, los desarrolladores han logrado evitar que todo el mundo deba editar el código del kernel y cambiar las constantes a mano. Para ello han creado una serie de programas (que en entornos más amigables son llamados **wizards**) que nos ayudarán en nuestra tarea, los más conocidos son tres:

Modo texto. Es el más antiguo y rudimentario. Consiste en una serie de preguntas relacionadas con los distintos componentes. En cada punto podremos decidir si incluimos o no la funcionalidad solicitada en el kernel. Aunque originalmente era imprescindible hoy es el último recurso por ser muy tedioso. Además, pese a poseer respuestas por defecto a todas las preguntas no es apto para no iniciados en la materia.

Para ejecutar este programa sólo es necesario ejecutar en el directorio donde hemos descomprimido el kernel (generalmente `/usr/src/linux`) lo siguiente: `make config`.

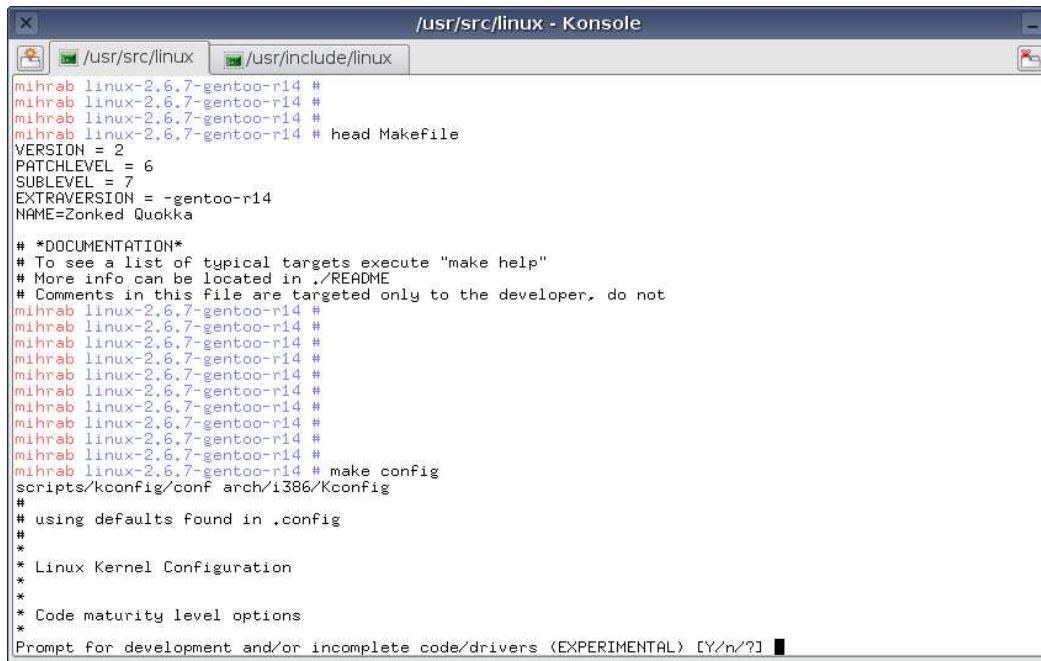


Figura 3.1: Ejemplo de configuración del kernel en modo texto. `make config`

Ventanas en modo texto. Mucho más sofisticado y cómodo de manejar. Consiste en una serie de ventanas que en modo texto nos permiten configurar todas las opciones. El hecho de que se aislen las opciones relacionadas logra que si queremos configurar nuestra tarjeta de sonido o la grabadora no haya que tomar decisiones sobre el driver de la tarjeta de red o el soporte para DMA. Para poder usar esta aplicación es necesario tener las librerías `ncurses` (en Debian por ejemplo los paquetes son: `libncurses5` y `ncurses3`). Una vez tengamos estas librerías instaladas invocaremos a la aplicación mediante `make menuconfig`.

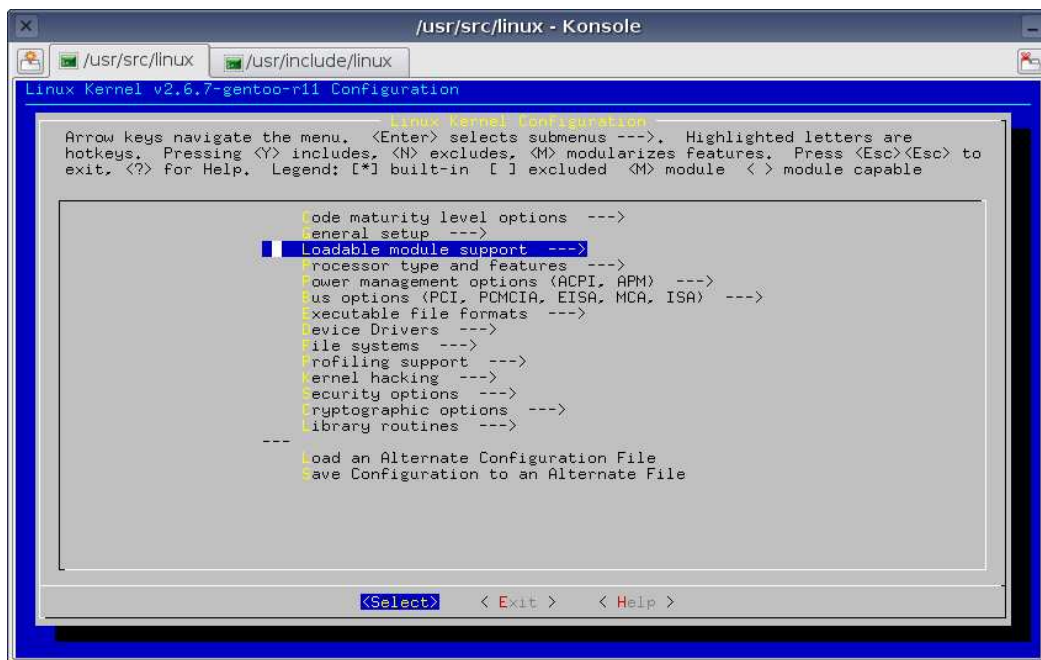


Figura 3.2: Ejemplo de configuración del kernel con ventanas en modo texto. `make menuconfig`

Modo X Window. Se trata de la misma aplicación anterior pero con las posibilidades que ofrece una interfaz gráfica basada en Xwindow. Para su ejecución, en este caso será necesario tener instaladas las librerías `QT` o

GTK dependiendo del front-end que queramos usar (en Gentoo por ejemplo nos bastaría con un `emerge qt` o `emerge gtk+`). Para su ejecución deberemos invocar desde un terminal X (en el directorio origen del código) el mandato: `make xconfig` para el front-end QT y `make gconfig` para su versión GTK.

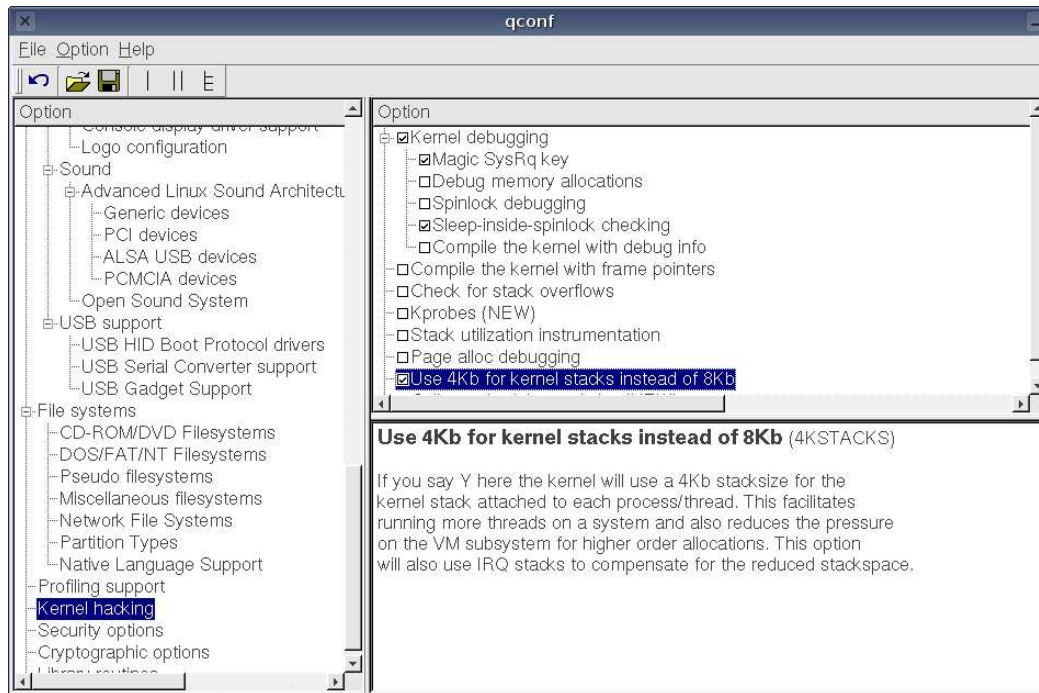


Figura 3.3: Ejemplo de configuración del kernel utilizando la versión para QT. `make xconfig`

Podríamos extendernos páginas completas sobre cada una de las opciones pero lo cierto es que, la mayoría de ellas no nos interesan. Instamos al lector a que se de un paseo por todas ellas comparando las posibilidades ofrecidas con la configuración de su máquina. La mayoría de las veces será necesario cambiar alguna opción o añadir un módulo concreto y el documento dónde se solicite expondrá sus razones coherentemente. De este modo hemos decidido no extendernos con las opciones. Desde la versión 2.6 de Linux no es necesario crear las dependencias necesarias con la orden `make dep`. Si por algún motivo el kernel que vamos a compilar es un Linux 2.4 o anterior debemos ejecutar `make dep` en este momento.

3.3.3. El proceso de compilación

El siguiente paso consiste en la compilación propiamente dicha del código. Es necesario aclarar algunos detalles en este punto. Como todos sabemos los procesadores Intel se han visto obligados, para mantener la compatibilidad con versiones anteriores a iniciar la carga de los sistemas operativos en modo real⁴. A causa de esto, el kernel de Linux se encuentra, al arrancar la máquina con sólo 640 Kb de memoria disponible donde cargarse para comenzar a ejecutar. Los desarrolladores tomaron la decisión de comprimir el kernel al cargarlo en la memoria para, más tarde, una vez se ha saltado a modo protegido y se dispone de toda la memoria, descomprimirlo. Sin embargo, al añadir diferentes dispositivos ha llegado un momento en que no es posible comprimir el núcleo para lograr cargarlo en la memoria, se ha utilizado otras estrategias. ¿En que no afecta esto a nosotros?, enseguida lo veremos.

Una vez hemos terminado de configura las diferentes opciones pasamos a compilar mediante la orden:

```
make zImage
```

Esto genera una imagen del kernel en el directorio `/usr/src/linux/arch/i386/boot`. Sin embargo podemos obtener un mensaje de error indicándonos que el kernel que estamos compilando no podrá ser cargado en el arranque mediante el método clásico de comprimirlo. La solución a esto es muy sencilla, tan sólo cambiaremos la orden de compilación, que será `make bzImage`.

Con el kernel ya configurado y compilado y para terminar el proceso de compilación sólo resta compilar los módulos. Es aconsejable que cualquier opción de la que no estemos completamente seguros de añadir o no sea

⁴Consiste en dotar al procesador de dos modos: modo real, utilizado para mantener la compatibilidad con software creado para versiones anteriores, y modo protegido dónde es posible explotar las nuevas capacidades del procesador

añadida como módulo (sí es posible). Esto suele reflejarse en los programas de configuración con una M sobre la opción. Para compilar los módulos utilizaremos la siguiente orden: `make modules`.

Desde Linux 2.6 esto es más cómodo ejecutando directamente `make all` o simplemente `make`. Para ver exactamente que tareas se realizan mediante un `make all` basta con ejecutar un `make help | grep "*"`

3.3.4. Instalación tras la compilación

Se trata del último paso, los detalles finales antes de tener nuestro kernel instalado en el sistema. Primero copiaremos los ficheros binarios producto de la etapa anterior en los lugares adecuados mediante las siguientes órdenes:

```
make install
make modules_install
```

Debemos ahora configurar el gestor de arranque, esto es, el primer programa que se ejecuta al arrancar un ordenador. El se ocupa de cargar y dar el control al núcleo del sistema operativo para que este a su vez arranque el resto del sistema. Los principales gestores de arranque son Lilo y Grub.

Vamos a configurarlo primero para realizar un arranque con Lilo del nuevo kernel y más tarde para Grub. Es importante habilitar una nueva etiqueta para lograr un arranque con el kernel antiguo (y evitar riesgos innecesarios). La orden `make install` ha llevado a cabo gran parte de este proceso, sin embargo es necesario completarlo del siguiente modo. Primero editaremos el fichero `/etc/lilo.conf` dónde podremos distinguir un fragmento similar a este:

```
image=/vmlinuz label=linux read-only root=/dev/hda1
image=/boot/vmlinuz.old labe=linux.old read-only root=/dev/hda1
```

Deberemos anotar el fichero referido en `image=/vmlinuz`. Generalmente este fichero es un enlace que apunta a la verdadera imagen del kernel (situada en el directorio `/boot`). Nos deberemos asegurar de qué el fichero apuntado es el nuevo recién copiado en el directorio. Podemos listar los ficheros con la opción `ls -l /boot` para comprobar cual ha sido la nueva imagen recién copiada. Una vez estemos *absolutamente* seguros de cual es podremos eliminar el enlace y colocarlo apuntando a la nueva imagen o (esto no es aconsejable) configurar la entrada del fichero `/etc/lilo.conf`. En resumen, lo realmente importante de este último paso es que, en el fichero `lilo.conf` se acceda a la imagen correcta del kernel. Un ejemplo podría ser el siguiente:

```
mihrab boot # ls -l
total 4.6M
-rw-r--r-- 1 root root 41K Jul 20 19:22 initrd-1024x768
drwx----- 2 root root 12K Jul 18 02:45 lost+found
lrwxr-xr-x 1 root root 24 Oct 27 03:00 vmlinuz -> vmlinuz-2.6.7
-rw-r--r-- 1 root root 1.4M Aug 11 20:40 vmlinuz-2.6.7
-rw-r--r-- 1 root root 1.5M Oct 20 15:15 vmlinuz-2.6.9 ...
mihrab boot # rm vmlinuz
mihrab boot # ln -s vmlinuz-2.6.9 vmlinuz
mihrab boot # ln -s vmlinuz-2.6.7 vmlinuz.old
mihrab boot # ls -l vm*
lrwxr-xr-x 1 root root 24 Oct 27 03:01 vmlinuz -> vmlinuz-2.6.9
```

De este modo configuramos lilo para que arranque utilizando el nuevo kernel y, no menos importante, ofrecemos la posibilidad de cargar la versión anterior por si hubiera problemas.

Ahora veremos como configurar el arranque usando Grub en lugar de Lilo. Grub es un gestor de arranque muy potente y tremendamente flexible, además entiende sistemas de ficheros y formatos ejecutables del núcleo, así que te permite arrancar un sistema operativo cualquiera, de la manera que quieras, sin necesidad de saber la posición física del núcleo en el disco.

Debemos editar el fichero `/boot/grub/grub.conf` dónde podremos ver una estructura similar a esta:

```
mihrab grub # cat grub.conf
timeout 10
default 0

title GNU/Linux 2.6.7
root (hd0,0) kernel /vmlinuz-2.6.7 root=/dev/hda3
```

Añadiremos ahora una nueva entrada para nuestro nuevo kernel, justo encima de la que tenemos actualmente para el antiguo. Esto no es totalmente necesario con Grub ya que si surge algún problema con el nuevo kernel y no podemos arrancarlo, podemos modificar los parámetros del kernel a arrancar desde el propio Grub al iniciar el sistema y hacerle usar la imagen del kernel antiguo.

Nuestra nueva entrada podría quedar más o menos así:

```
title GNU/Linux 2.6.9
root (hd0,0) kernel /vmlinuz-2.6.9 root=/dev/hda3
```

Si observamos la segunda línea: `root (hd0,0)`, podemos apreciar que la sintaxis de dispositivos usada en Grub no nos es familiar. Para empezar, Grub requiere que se encierre el nombre del dispositivo entre paréntesis `'('y ')'`. Lo que `hd` significa es que se trata de un disco duro (hard disk). El primer entero indica el número de la unidad, es decir, que se trata del primer disco duro, mientras que el segundo entero indica el número de la partición. Vamos, lo que se suele conocer como `hda1` en Grub se traduciría a `(hd0,0)`.

Las opciones que vemos al principio del fichero `grub.conf` son para indicar cuanto tiempo se ha de esperar hasta iniciar el sistema por defecto y cual sería este.

Ya sólo queda rearrancar la máquina para probar el nuevo kernel. Una vez ha vuelto a cargar el sistema operativo y estamos sobre la consola, lo primero será cerciorarse de que nuestro nuevo kernel ha sido cargado correctamente, viendo su número de versión. Esto podemos hacerlo de varias formas, entre ellas, por ejemplo, utilizando `dmesg`, `/proc` o `uname` como podemos ver a continuación:

```
jorge@mihrab ~ $ dmesg | head -n
Linux version 2.6.9 (root@mihrab) (gcc version 3.4.2) #12 Wed Oct 20 15:15:26
CEST 2004
jorge@mihrab ~ $ cat /proc/version
Linux version 2.6.9 (root@mihrab) (gcc version 3.4.2) #12 Wed Oct 20 15:15:26
CEST 2004
jorge@mihrab~ $ uname -a
Linux mihrab 2.6.9 #12 Wed Oct 20 15:15:26 CEST 2004 i686
Intel(R) Pentium(R) M processor 1500MHz GenuineIntel GNU/Linux
```

El primer método utiliza el mandato `dmesg` cuya misión es reproducir toda la salida que el kernel produjo durante el arranque (utilizamos `head -n 1` por que lo primero que hace todo kernel al arrancar es mostrar un mensaje con su número de versión y el compilador que utilizado y la fecha en que fue compilado). Esta aplicación resulta interesante para observar en detalle la evolución de la carga del kernel en memoria dado que se detalla todo el hardware hallado o posibles problemas. Un ejemplo de la salida típica de `dmesg` sería:

```
Linux version 2.6.7 (root@mihrab) (gcc version 3.4.2))
#12 Wed Aug 11 20:40:26 CEST 2004
BIOS-provided physical RAM map:
 BIOS-e820: 0000000000000000 - 000000000009f800 (usable)
 BIOS-e820: 000000000009f800 - 00000000000a0000 (reserved)
 BIOS-e820: 00000000000d0000 - 00000000000d4000 (reserved)
 BIOS-e820: 00000000000e0000 - 0000000000100000 (reserved)
 BIOS-e820: 0000000000100000 - 000000001ff60000 (usable)
 BIOS-e820: 000000001ff60000 - 000000001ff6a000 (ACPI data)
 BIOS-e820: 000000001ff6a000 - 000000001ff6b000 (reserved)
 BIOS-e820: 000000001ff6b000 - 000000001ff70000 (ACPI NVS)
 BIOS-e820: 000000001ff70000 - 0000000020000000 (reserved)
 BIOS-e820: 00000000ffb80000 - 00000000ffc00000 (reserved)
 BIOS-e820: 00000000fff80000 - 0000000100000000 (reserved)
511MB LOWMEM available.
On node 0 totalpages: 130912
  DMA zone: 4096 pages, LIFO batch:1
  Normal zone: 126816 pages, LIFO batch:16
  HighMem zone: 0 pages, LIFO batch:1
DMI present.
ACPI: RSDP (v000 TOSHIB                               ) @ 0x000f7a00
ACPI: RSDT (v001 TOSHIB 750          0x00970814 MASM 0x06110000) @ 0x1ff63fd8
ACPI: FADT (v002 TOSHIB 750          0x20030101 MASM 0x61100000) @ 0x1ff69d03
ACPI: SSDT (v001 TOSHIB A0007        0x00970814 MSFT 0x0100000e) @ 0x1ff69d87
ACPI: DBGP (v001 TOSHIB 750          0x00970814 MASM 0x61100000) @ 0x1ff69fa4
ACPI: BOOT (v001 TOSHIB 750          0x00970814 MASM 0x06110000) @ 0x1ff69fd8
ACPI: DSDT (v001 TOSHIB A0007        0x20030806 MSFT 0x0100000e) @ 0x00000000
```

```

Built 1 zonelists
Kernel command line: root=/dev/hda3 video=mtrr,vesa:1024x768 vga=0x317
splash=silent
bootsplash: silent mode.
Local APIC disabled by BIOS -- reenabling.
Found and enabled local APIC!
Initializing CPU#0
CPU 0 irqstacks, hard=c03dc000 soft=c03db000
PID hash table entries: 2048 (order 11: 16384 bytes)
Detected 1496.685 MHz processor.
Using tsc for high-res timesource
Console: colour dummy device 80x25
Memory: 515544k/523648k available (1994k kernel code, 7316k reserved,
761k data, 152k init, 0k highmem)
Checking if this processor honours the WP bit even in supervisor mode...
Ok.
Calibrating delay loop... 2957.31 BogoMIPS
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
CPU: After generic identify, caps: a7e9fbbf 00000000 00000000 00000000
CPU: After vendor identify, caps: a7e9fbbf 00000000 00000000 00000000
CPU: L1 I cache: 32K, L1 D cache: 32K
CPU: L2 cache: 1024K
CPU: After all inits, caps: a7e9fbbf 00000000 00000000 00000040
Intel machine check architecture supported.
Intel machine check reporting enabled on CPU#0.
CPU: Intel(R) Pentium(R) M processor 1500MHz stepping 05
Enabling fast FPU save and restore... done.
Enabling unmasked SIMD FPU exception support... done.
Checking 'hlt' instruction... OK.
enabled ExtINT on CPU#0
ESR value before enabling vector: 00000000
ESR value after enabling vector: 00000000
Using local APIC timer interrupts.
calibrating APIC timer ...
.... CPU clock speed is 1495.0998 MHz.
.... host bus clock speed is 99.0733 MHz.
checking if image is initramfs...it isn't (ungzip failed);
looks like an initrd
Freeing initrd memory: 40k freed
NET: Registered protocol family 16
PCI: PCI BIOS revision 2.10 entry at 0xfd981, last bus=4
PCI: Using configuration type 1
mtrr: v2.0 (20020519)
ACPI: Subsystem revision 20040326
ACPI: IRQ9 SCI: Edge set to Level Trigger.
ACPI: Interpreter enabled
ACPI: Using PIC for interrupt routing
ACPI: PCI Interrupt Link [LNKA] (IRQs *3 4 5 7 11)
ACPI: PCI Interrupt Link [LNKB] (IRQs 3 *4 5 7 11)
ACPI: PCI Interrupt Link [LNKC] (IRQs 3 4 5 7 11) *0, disabled.
ACPI: PCI Interrupt Link [LNKD] (IRQs 3 4 5 7 *11)
ACPI: PCI Interrupt Link [LNKE] (IRQs 3 4 5 7 *11)
ACPI: PCI Interrupt Link [LNKF] (IRQs *3 4 5 7 11)
ACPI: PCI Interrupt Link [LNKG] (IRQs *10)
ACPI: PCI Interrupt Link [LNKH] (IRQs 3 4 5 *7 11)
ACPI: PCI Root Bridge [PCIO] (00:00)
PCI: Probing PCI hardware (bus 00)
PCI: Ignoring BAR0-3 of IDE controller 0000:00:1f.1
PCI: Transparent bridge - 0000:00:1e.0
ACPI: PCI Interrupt Routing Table [_SB_.PCIO._PRT]
ACPI: PCI Interrupt Routing Table [_SB_.PCIO.PCI1._PRT]
ACPI: PCI Interrupt Routing Table [_SB_.PCIO.PCIB._PRT]
ACPI: Power Resource [PFAN] (off)

```

```

Toshiba System Managment Mode driver v1.11 26/9/2001
SCSI subsystem initialized
ACPI: PCI Interrupt Link [LNKA] enabled at IRQ 3
ACPI: PCI Interrupt Link [LNKD] enabled at IRQ 11
ACPI: PCI Interrupt Link [LNKH] enabled at IRQ 7
ACPI: PCI Interrupt Link [LNKC] enabled at IRQ 11
ACPI: PCI Interrupt Link [LNKB] enabled at IRQ 4
ACPI: PCI Interrupt Link [LNKG] enabled at IRQ 10
ACPI: PCI Interrupt Link [LNKF] enabled at IRQ 3
ACPI: PCI Interrupt Link [LNKE] enabled at IRQ 11
PCI: Using ACPI for IRQ routing
vesafb: framebuffer at 0xe0000000, mapped to 0xe080e000, size 3072k
vesafb: mode is 1024x768x16, linelength=2048, pages=1
vesafb: protected mode interface info at c000:d0e0
vesafb: scrolling: redraw
vesafb: directcolor: size=0:5:6:5, shift=0:11:5:0
fb0: VESA VGA frame buffer device
Simple Boot Flag at 0x36 set to 0x1
Machine check exception polling timer started.
devfs: 2004-01-31 Richard Gooch (rgooch@atnf.csiro.au)
devfs: boot_options: 0x1
udf: registering filesystem
Initializing Cryptographic API
ACPI: AC Adapter [ADP1] (off-line)
ACPI: Battery Slot [BAT1] (battery present)
ACPI: Power Button (FF) [PWRF]
ACPI: Lid Switch [LID]
ACPI: Fan [FAN] (off)
ACPI: Processor [CPU0] (supports C1 C2 C3)
ACPI: Thermal Zone [THRM] (28 C)
toshiba_acpi: Toshiba Laptop ACPI Extras version 0.18
toshiba_acpi: HCI method: \_SB_.VALZ.GHCI
mice: PS/2 mouse device common for all mice
serio: i8042 AUX port at 0x60,0x64 irq 12
input: PS/2 Generic Mouse on isa0060/serio1
serio: i8042 KBD port at 0x60,0x64 irq 1
input: AT Translated Set 2 keyboard on isa0060/serio0
bootsplash 3.1.4-2004/02/19-spock-0.1: looking for picture....
silentjpeg size 17594 bytes, found (1024x768, 23422 bytes, v3).
Console: switching to colour frame buffer device 122x40
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
loop: loaded (max 8 devices)
Uniform Multi-Platform E-IDE driver Revision: 7.00alpha2
ide: Assuming 33MHz system bus speed for PIO modes;
override with idebus=xx
ICH4: IDE controller at PCI slot 0000:00:1f.1
PCI: Enabling device 0000:00:1f.1 (0005 -> 0007)
ICH4: chipset revision 3
ICH4: not 100% native mode: will probe irqs later
    ide0: BM-DMA at 0x1840-0x1847, BIOS settings: hda:DMA, hdb:pio
    ide1: BM-DMA at 0x1848-0x184f, BIOS settings: hdc:DMA, hdd:pio
hda: TOSHIBA MK4021GAS, ATA DISK drive
Using anticipatory io scheduler
ide0 at 0x1f0-0x1f7,0x3f6 on irq 14
hdc: UJDA750 DVD/CDRW, ATAPI CD/DVD-ROM drive
ide1 at 0x170-0x177,0x376 on irq 15
hda: max request size: 128KiB
hda: 78140160 sectors (40007 MB), CHS=65535/16/63, UDMA(100)
    /dev/ide/host0/bus0/target0/lun0: p1 p2 p3
hdc: ATAPI 24X DVD-ROM CD-R/RW drive, 2048kB Cache, UDMA(33)
Uniform CD-ROM driver Revision: 3.20
oprofile: using NMI interrupt.
NET: Registered protocol family 2
IP: routing cache hash table of 4096 buckets, 32Kbytes
TCP: Hash tables configured (established 32768 bind 65536)

```

```

ip_contrack version 2.1 (4091 buckets, 32728 max) -
296 bytes per contrack
ip_tables: (C) 2000-2002 Netfilter core team
ipt_recent v0.3.1: Stephen Frost <sfrost@snowman.net>.
http://snowman.net/projects/ipt_recent/
arp_tables: (C) 2002 David S. Miller
NET: Registered protocol family 1
NET: Registered protocol family 17
speedstep-centrino: found "Intel(R) Pentium(R) M processor 1500MHz":
max frequency: 1500000kHz
ACPI: (supports S0 S3 S4 S5)
RAMDISK: Couldn't find valid RAM disk image starting at 0.
ReiserFS: hda3: found reiserfs format "3.6" with standard journal
ReiserFS: hda3: using ordered data mode
ReiserFS: hda3: journal params: device hda3, size 8192,
journal first block 18, max trans len 1024, max batch 900,
max commit age 30, max trans age 30
ReiserFS: hda3: checking transaction log (hda3)
ReiserFS: hda3: Using r5 hash to sort names
VFS: Mounted root (reiserfs filesystem) readonly.
Mounted devfs on /dev
Freeing unused kernel memory: 152k freed
Adding 1004052k swap on /dev/hda2. Priority:-1 extents:1
nvidia: module license 'NVIDIA' taints kernel.
NVRM: loading NVIDIA Linux x86 NVIDIA Kernel Module 1.0-6111
Tue Jul 27 07:55:38 PDT 2004
PCI: Setting latency timer of device 0000:00:1f.5 to 64
intel8x0_measure_ac97_clock: measured 49395 usecs
intel8x0: clocking to 48000
eepro100.c:v1.09j-t 9/29/99 Donald Becker
http://www.scyld.com/network/eepro100.html
eepro100.c: $Revision: 1.36 $ 2000/11/17
Modified by Andrey V. Savochkin <saw@saw.sw.com.sg> and others
eth0: OEM i82557/i82558 10/100 Ethernet, 00:08:0D:1C:00:F3, IRQ 11.
Board assembly 000000-000, Physical connectors present: RJ45
Primary interface chip i82555 PHY #1.
General self-test: passed.
Serial sub-system self-test: passed.
Internal registers self-test: passed.
ROM checksum self-test: passed (0x04f4518b).
usbcore: registered new driver usbfs
usbcore: registered new driver hub
USB Universal Host Controller Interface driver v2.2
uhci_hcd 0000:00:1d.0: Intel Corp. 82801DB (ICH4) USB UHCI #1
PCI: Setting latency timer of device 0000:00:1d.0 to 64
uhci_hcd 0000:00:1d.0: irq 3, io base 00001800
uhci_hcd 0000:00:1d.0: new USB bus registered, assigned bus number 1
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
uhci_hcd 0000:00:1d.1: Intel Corp. 82801DB (ICH4) USB UHCI #2
PCI: Setting latency timer of device 0000:00:1d.1 to 64
uhci_hcd 0000:00:1d.1: irq 11, io base 00001820
uhci_hcd 0000:00:1d.1: new USB bus registered, assigned bus number 2
hub 2-0:1.0: USB hub found
hub 2-0:1.0: 2 ports detected
ehci_hcd 0000:00:1d.7: Intel Corp. 82801DB (ICH4) USB2 EHCI Controller
PCI: Setting latency timer of device 0000:00:1d.7 to 64
ehci_hcd 0000:00:1d.7: irq 7, pci mem e0c85000
ehci_hcd 0000:00:1d.7: new USB bus registered, assigned bus number 3
PCI: cache line size of 32 is not supported by device 0000:00:1d.7
ehci_hcd 0000:00:1d.7: USB 2.0 enabled, EHCI 1.00, driver 2004-May-10
hub 3-0:1.0: USB hub found
hub 3-0:1.0: 4 ports detected
Initializing USB Mass Storage driver...
usbcore: registered new driver usb-storage

```

```

USB Mass Storage support registered.
usbcore: registered new driver usbhid
drivers/usb/input/hid-core.c: v2.0:USB HID core driver
Real Time Clock Driver v1.12
boot splash 3.1.4-2004/02/19-spock-0.1: looking for picture....
found (1024x768, 23422 bytes, v3).
boot splash: status on console 0 changed to on
boot splash 3.1.4-2004/02/19-spock-0.1: looking for picture....
found (1024x768, 23422 bytes, v3).
boot splash: status on console 1 changed to on
boot splash 3.1.4-2004/02/19-spock-0.1: looking for picture....
found (1024x768, 23422 bytes, v3).
boot splash: status on console 2 changed to on
boot splash 3.1.4-2004/02/19-spock-0.1: looking for picture....
found (1024x768, 23422 bytes, v3).
boot splash: status on console 3 changed to on
boot splash 3.1.4-2004/02/19-spock-0.1: looking for picture....
found (1024x768, 23422 bytes, v3).
boot splash: status on console 4 changed to on
boot splash 3.1.4-2004/02/19-spock-0.1: looking for picture....
found (1024x768, 23422 bytes, v3).
boot splash: status on console 5 changed to on
ehci_hcd 0000:00:1d.7: remove, state 1

```

El segundo método es quizá más rudimentario. Sin embargo resulta mucho más útil de cara a la obtención de información. Es conveniente, de hecho, que todo usuario se habitúe a buscar los datos que precisa sobre los parámetros del sistema operativo entre las entradas del directorio `/proc`. Este directorio es un tanto especial dado que sus entradas no son ficheros como tales y se crean cuando arranca el kernel. Al listar un fichero, por ejemplo `/proc/ioports`, no se muestra su contenido sino que se realizan las oportunas llamadas al sistema operativo consultando la información necesaria para luego mostrarla por pantalla. Si no se lo cree intente buscar el tamaño del fichero `/proc/vmstat` o cualquier otro fichero situado en `/proc`.

3.4. Los módulos en detalle

Pese a tener nuestro kernel ya compilado y funcionando, en más de una ocasión será necesario añadirle un módulo. Por ejemplo, como veremos más adelante, para lograr utilizar un dispositivo de almacenamiento externo usb deberemos añadir el módulo `usb_storage` entre otros. En esta sección proporcionaremos la información básica para la gestión de módulos.

3.4.1. Introducción a los módulos de un núcleo

Un módulo no es más que un programa C común que incluye ciertos ficheros de cabecera específicos del núcleo. Si nos paramos a ver el código en detalle, no encontramos una función `main` sino dos, llamadas `init_module` y `cleanup_module` que son ejecutadas al cargar y al descargar el módulo de memoria respectivamente, aunque desde Linux 2.6 hay algo más de libertad en este aspecto y podemos llamarlas como queramos e indicarlo mediante dos macros: `module_init` y `module_exit`. El proceso de compilación de un módulo es muy sencillo, tan sólo lo compilaremos como cualquier otro programa, aunque no lo enlazaremos (con la opción `-c` del compilador `gcc` obtenemos el fichero compilado, no el ejecutable final tras enlazarlo con las librerías pertinentes). Una vez esto ha sido efectuado con éxito lo podremos cargar en el núcleo mediante el mandato `modprobe` y descargarlo con `rmmmod` como veremos más adelante.

El fin último del uso de módulos es poder añadir/eliminar nuevas funcionalidades a un kernel sin necesidad de recompilarlo y rearrancar la máquina. Es conveniente que la mayoría de los dispositivos no esenciales como la gestión de sistemas de ficheros foráneos como `fat32`, el dispositivo encargado de gestionar la tarjeta de sonido, etc. sean compilados como módulos. Además, existe un demonio llamado `kernelld` encargado de cargar el módulo automáticamente cuando este es requerido (aunque este método no es eficaz dado que en ocasiones nos interesará cargarlos con condiciones específicas como ahora veremos).

3.4.2. Gestión de módulos

Trataremos en este apartado la inserción y borrado de módulos en el kernel. Detallaremos el uso de cuatro mandatos: `insmod`, `rmmmod`, `lsmod` y `modprobe`. No consiste en proporcionar una lista completa de los parámetros que

acepta cada uno, dado que conocer esto resulta muy sencillo consultando el manual (`man modprobe`), dónde además será posible conocer otras herramientas relacionadas.

insmod Carga un módulo en el kernel.

rmmmod Borra un módulo del kernel.

lsmod Lista los módulos cargados actualmente. El resultado es similar a listar el contenido del fichero `/proc/modules`.

modprobe Carga un módulo aunque previamente resuelve todas las dependencias cargando otros módulos si fuera necesario. Es la forma aconsejada de cargar módulos.

Capítulo 4

Gestión de redes con Linux

4.1. Introducción a las redes de computadores

4.1.1. Ventajas de la puesta en común de recursos

Uno de los protocolos más usados en internet para la transferencia de archivos es el ftp (File Transfer Protocol), es un método sencillo y efectivo de transferir archivos de una forma rápida y fiable entre equipos conectados a una red TCP/IP. El protocolo de transferencia de archivos tiene dos modos de transferencia: ascii y binario. En general el tipo de transferencia ascii solo nos permite transferir archivos de texto plano, y el binario todo lo demás. La sesión ftp nos permite conectarnos a otro equipo, y una vez conectados, podremos navegar por el árbol de directorios, listar sus contenidos y transferir archivos desde y hacia la computadora remota. Para abrir una sesión ftp hará falta un nombre de usuario y una contraseña válidas en ese equipo, muchos servidores de ftp disponen de cuentas públicas, en las que, introduciendo los datos:

```
login      : anonymous
password  : tucuentade@correo.electronico
```

Te permiten acceder al sistema de archivos, normalmente para ofrecerte la descarga de sus archivos. El protocolo ftp tiene una gran cantidad de órdenes, que pueden variar de un servidor a otro

4.1.2. Resumen de una sesión FTP

- Inicio de una sesión, mediante el comando `open nombre_de_servidor`.
- Autenticación del usuario mediante un nombre de usuario y una contraseña
- Cambio de modo de transferencia, binary/ascii.
- Búsqueda y carga/descarga de archivos, con las órdenes: `cd`, `lcd`, `pwd`, `ls`, `dir`, `get`, `mget`, `put`, `mput`.
- Finalización de la sesión, órdenes `bye`, `quit`.

4.1.3. Instalación de un servidor FTP

La instalación del servidor en debian es tan sencilla como teclear el comando:

```
# apt-get install proftpd
```

En la instalación del demonio de ftp (*PROfessional FTP Daemon*), preguntará si deseamos tener una cuenta de ftp anónimo. Si lo deseamos, el programa de configuración la creará automáticamente. Si en algún momento deseamos quitar el servidor de ftp, podemos matar el demonio (que se reiniciará en el próximo arranque).

4.2. Terminal remota: Telnet

4.2.1. Cliente de telenet

La orden telnet es la herramienta básica para las conexiones remotas bajo linux. Con telnet podrás mantener sesiones con un terminal de la computadora remota, ejecutando órdenes como si estuvieras conectado localmente. La

!	Prefijo que se usa para dar una orden a la maquina local
account	Envia la orden de cuenta al servidor remoto
append	Añade un archivo
ascii	Establece el tipo de transferencia de archivos en la modalidad ascii
bell	Emite una señal acustica cuando se completa una orden
binary	Establece el tipo de transferencia de archivos en la modalidad binaria
bye	Finaliza la sesión ftp y sale
cd	Cambia el directorio en la computadora remota
cdup	Equivalente al comando cd ..
chmod	Modifica los permisos del archivo remoto
close	Finaliza una sesion ftp
cr	Conmuta el filtrado de retornos de carro cuando se recibe un archivo ascii
delete	Borra archivos remotos
dir	Lista el contenido del directorio
disconnect	Equivale a close
exit	Finaliza la sesion ftp y sale
get	Obtiene un archivo de la computadora remota
hash	Conmuta la impresión del carácter # para cada memoria intermedia transferida
help	Imprime información de ayuda local
idle	Obtiene o fija el temporizador en la computadora remota
lcd	Cambia el directorio local de trabajo
ls	Lista el contenido del directorio remoto
mdelete	Borra multiples archivos en la maquina remota
mdir	Lista el contenido de directorios remotos multiples
mget	Obtiene archivos multiples de la computadora remota
mkdir	Crea un directorio en la computadora remota
mode	Establece la modalidad de transferencia de archivos
mput	Envia archivos multiples a la computadora remota
open	Establece una conexión con el sitio ftp remoto
passive	Establece el modo de transferencia pasivo
prompt	Fuerza indicaciones interactivas de órdenes múltiples
put	Envía un archivo a la computadora local
pwd	Imprime el directorio de trabajo en la maquina remota
quit	Finaliza la sesión ftp y sale
reget	Obtiene un archivo empezando desde el final del archivo local
rename	Renombra un archivo
rmdir	Borra un directorio en la maquina remota
send	Envía un archivo a la máquina remota
size	Muestra el tamaño del archivo remoto
user	Envía nueva información de usuario
verbose	Conmuta la modalidad detal

Figura 4.1: Resumen de los comandos ftp

conexión via telnet a un sistema es casi análoga a abrir una sesión en uno de los terminales en modo texto. Telnet no permite la conexión a un sistema en un entorno grafico, para ello existe otra herramienta que se estudiará más adelante. Sintaxis de la orden telnet:

```
telnet host [port]
```

Realmente telnet tiene muchas más versatilidad, pero éste es el formato más usual de la orden. Por defecto telnet se conecta al puerto 23 de la máquina destino, que es usualmente donde se activa el servidor de telnet, pero es posible abrir una sesión en puertos diferentes. Es posible, a través de una sesión remota, manejar el correo de un servidor pop3 (puerto 110) o enviar un correo a través de una sesión smtp (puerto 25).

Ejemplo de una sesión telnet:

Se inicia escribiendo el comando telnet seguido del nombre del host al que deseamos conectarnos. Telnet responde con el siguiente mensaje:

```
Trying (dir. IP)
```

Si la computadora acepta la conexión, responderá con un mensaje diciendo que se ha conectado:

```
Connected to manwe.ainulindale.es.
```

Si no lo logra, entonces contestará lo siguiente:

```
[Wed 7-20:27] carlos@Iluvatar:~ > telnet acm.asoc.fi.upm.es
Trying 138.100.13.202...
telnet: Unable to connect to remote host: Connection refused
```

Una vez abierta la sesión linux emite el mensaje

```
Connected to (nombre del host)
```

Y un mensaje que indica una secuencia determinada de caracteres, la secuencia de escape. Ésta sirve para salir de la sesión de terminal y pasar al interprete de ordenes de telnet. Se utiliza cuando se desea pasar ordenes directamente al interprete y no a la sesion en la maquina remota. Una vez conectado, se solicitara un nombre de usuario y una contraseña, si la informacion de inicio de sesión es correcta, se iniciará una sesión en el sistema remoto. A continuación un ejemplo de una sesión telnet desde una computadora linux, conectada a otra computadora linux.

```
[Wed 7-19:35] carlos@Iluvatar:~ > telnet manwe
Trying 192.168.0.12...
Connected to manwe.ainulindale.es.
Escape character is '^'.
Debian GNU/Linux 2.2 manwe.ainulindale.es
login: carlos
Password:
Last login: Wed Nov 7 18:20:19 2001 from iluvatar on pts/0
Linux Manwe 2.4.4 #14 Fri Oct 19 14:09:15 CEST 2001 i586 unknown
```

No mail.

```
[Wed 7-19:35] carlos@Manwe:~ > logout
Connection closed by foreign host.
[Wed 7-19:36] carlos@Iluvatar:~ >
```

4.2.2. Servidor de Telnet

En cuanto al servidor de telnet solo dos palabras, como instalarlo en debian y cómo abrir y cerrar el puerto. En debian instalar el servidor de telnet es algo tan simple como teclear el comando:

```
# apt-get install telnetd
```

Cuando finalice ya tendremos instalado y funcionando a la perfección el demonio de telnet, telnetd (telnet daemon). Si en algún momento quisieramos cerrar el puerto de telnet, lo que tendríamos que hacer sería editar el archivo /etc/inetd.conf, comentar la linea:

```
telnet stream tcp nowait telnetd.telnetd /usr/sbin/tcpd /usr/sbin/in.telnetd
```

Es necesario reiniciar el demonio de red, lo haremos con: /etc/init.d/inetd restart. En general, en la distribución debian los puertos de la máquina se gestionan desde este archivo.

4.3. Secure Shell: ssh

Telnet es una herramienta muy poderosa, pero es vulnerable a ataques de usuarios maliciosos por que la transferencia de información a través de la red va sin codificar, y cualquiera observando el tráfico de la red podría descubrir nuestras contraseñas. Para dificultar el descubrimiento de la información privada que viaja a través de la red en este tipo de sesiones existe una herramienta de shell seguro ssh (secure shell). Su instalación en Debian es tan fácil como todo:

```
# apt-get install ssh
```

Una vez que la tengamos instaladas podremos abrir sesiones remotas de forma segura con el comando:

```
$ ssh -l nombre-de-usuario host
```

La primera vez que nos conectemos con esa ip nos pregunta si estamos seguros de a quién nos estamos conectando, ya que podríamos conectarnos a la maquina de un usuario malicioso, en la que nos autenticaríamos dándole así nuestra contraseña. En principio si conocemos la máquina destino, no tendría por qué haber problemas, aceptamos y nos autenticamos con nuestra contraseña. Una vez finalizado el proceso de autenticación, entramos en un terminal con las mismas posibilidades que el de telnet, pero en modo seguro.

4.4. Login Gráfico

Hemos visto cómo acceder a una máquina remota con un terminal de texto, pero linux nos ofrece muchas más posibilidades. Otra utilidad interesante para el manejo remoto de un ordenador es la posibilidad de abrir una sesión gráfica desde una máquina remota. Para ello tienen que cumplirse una serie de requisitos mínimos:

- En la máquina cliente deberán estar configuradas las X.
- La máquina servidor deberá estar activada la opción de recibir peticiones XDMCP. En GDM (Gnome Display Manager) es fácil de activar. Basta con poner a cierto la línea señalada del archivo `/etc/X11/gdm/gdm.conf` en el fichero de configuración del GDM tiene que estar la opción:

```
[xdmcp]
----> Enable=true
HonorIndirect=false
MaxPending=4
MaxPendingIndirect=4
MaxSessions=16
MaxWait=30
MaxWaitIndirect=30
Port=177
```

Para abrir una sesión remota, sólo tendremos que teclear el comando:

```
X :1 -query (nombre\_de\_la\_máquina)
```

Por ejemplo, mediante el mandato `X :1 -query Iluvatar` se arrancan las X (sesión gráfica) en el display 1, al que podremos acceder pulsando `Control+Alt+F8`, y solicita a la máquina una sesión gráfica por XDMCP.

4.5. Sistema remoto de ficheros: nfs

Otra manera de ver la pueta en común de recursos del ordenador es el Sistema remoto de ficheros. El protocolo ftp nos permitía transferir archivos de una máquina a otra, en cambio nfs nos permite ver los archivos compartidos como parte de nuestra propia máquina. Los archivos compartidos podemos montarlos como parte de nuestro árbol de directorios. El proceso de configuración del nfs es algo más complicado, a continuación se detalla lo básico para que se puedan configurar un cliente y un servidor de nfs, pero para más información recomiendo encarecidamente acudir al NFS-HOWTO.

4.5.1. Configuración de un sistema NFS

1. Los archivos de configuración de los hosts (lista de ip's a las que corresponde un nombre de equipo) */etc/hosts* tienen que ser coherentes en los dos equipos entre los cuales se quiere montar NFS. Por ejemplo: un equipo lo llamaremos Iluvatar y otro Nessa

```
/etc/hosts @ Iluvatar
```

```
127.0.0.1    Iluvatar    localhost
192.168.0.2  Manwe
192.168.0.3  Nessa
```

```
/etc/hosts @ Nessa
```

```
127.0.0.1    Nessa    localhost
192.168.0.1  Iluvatar
192.168.0.2  Manwe
```

2. En el archivo */etc/exports* se tienen que declarar los directorios que se quieren exportar y a dónde se quieren exportar, con el siguiente formato:
directorio_que_se_exporta opciones_exportación Por ejemplo (Utilizaremos * para indicar opciones por defecto):

```
/etc/exports @ Iluvatar
```

```
# /etc/exports: the access control list for filesystems which may be exported
#                to NFS clients.  See exports(5).
#dir_raiz      destino(opciones)
```

```
nessa(rw,no_root_squash)
    rw           : read/write
    ro           (*): read only
    no_root_squash : PERMITE que los clientes accedan a
                   los archivos con owner root
    root_squash (*): EVITA que los clientes accedan a
                   los archivos con owner root.
```

3. Ya esta configurado la exportacion de la informacion. Ahora hay que configurar los demonios que se encargan de ella.

- a) En la maquina servidor (en mi caso Iluvatar), debe estar activado:

portmap *man portmap* para más información. Normalmente este demonio suele venir activado por defecto.

rpc.mountd

rpc.nfsd Si se cambia la configuración en los archivos */etc/hosts* y/o */etc/exports* hace falta matar a *rpc.mountd* y *rpc.nfsd* para que actualicen la configuración.

El paquete que permite esto, en Debian es **nfs-server**

- b) En la maquina cliente:

- Tiene que estar activado *portmap*
- El comando para montar el archivo NFS es:
`mount -t nfs nombre:directorio_que_monto directorio_en_donde_monto`
- Se puede editar el archivo */etc/fstab* para no tener que dar todos los parametros cada vez que queramos montar el directorio. Ejemplo en mi caso:

```
/etc/fstab @ Nessa
```

```
[..]
```

```
Iluvatar:/    /mnt/Iluvatar nfs    defaults,user,noauto 0 0
```

```
Manwe:/      /mnt/Manwe    nfs    defaults,user,noauto 0 0
```

Ya puedes acceder a los archivos del servidor a través de NFS, pero falta una cosa importante: en este archivo no se tienen en cuenta cuestiones de seguridad.

4.6. Servidor de impresión: apsfiler

Una vez que ya tenemos montada la red, seguramene querremos poder imprimir desde todos los ordenadores conectados a la red. Nada más fácil. En general para entrar un poco en detalles, tendremos que configurar la impresora en una máquina local que será la que actúe como servidora de impresión. Y luego configurar en la otra máquina la impresora como remota, este proceso es muy sencillo. Existe actualmente una aplicación muy sencilla para conseguirlo: Cups.

4.7. Compatibilidad con Windows: Samba

Samba es la adaptación del protocolo de Microsoft para compartir recursos, portada a linux. Con samba podemos, por ejemplo conseguir compartir carpetas en una máquina Windows y montarlas en linux, y exportar carpetas de un sistema linux, y verlas en una maquina Windows. Pero no sólo eso, también podemos compartir impresoras viendo desde Windows la impresora conectada a la maquina linux como si fuera servida por otra máquina Windows. Hay otras muchas cosas que se pueden hacer con samba, por ejemplo compartir bases de datos y servicios, para que la maquina cliente ve a linux como si de un servidor NT se tratara.

4.8. Arranque del sistema vía NFS

Este apartado está dedicado a la administración y mantenimiento de sistemas GNU/Linux basados en soluciones diskless (sin disco). Estos sistemas son de gran utilidad en áreas muy diversas como: procesamiento paralelo, terminales gráficas “tontas”, y otras. La configuración de estos sistemas se realiza en dos partes: cliente y servidor. Las máquinas cliente necesitan un sistema de archivos para montar como raíz. Comúnmente éste sistema se haya en un disco “local” de la máquina. Sin embargo, los sistemas diskless no disponen de una partición raíz, y dependen por tanto de otro sistema para arrancar. NFS es una de las posibles alternativas que permiten arrancar un sistema Linux y montar como sistema raíz un directorio remoto. De esta manera, las máquinas cliente no requieren un disco duro local. Imaginemos un clúster de 20 equipos dedicado al procesamiento de datos. Normalmente estos clústeres no requieren de un disco local para trabajar con un rendimiento óptimo. Esto permite eliminar 20 discos de nuestro presupuesto, pero debemos considerar la compra de un disco lo suficientemente grande como para albergar los sistemas de archivos a compartir. Otro uso de esta facilidad consiste en arrancar un sistema Linux desde una máquina cualquiera con acceso a la red del servidor de NFS.

4.8.1. Configuración del servidor

Lo primero es configurar una máquina para que comparta un directorio raíz por cada cliente. Es importante que los clientes tengan directorios distintos. Esto es, el servidor debe compartir distintos directorios para lograr que las máquinas respeten sus ficheros y sean independientes. Esto puede lograrse añadiendo las siguientes líneas a */etc/exports*:

```
192.168.0.70 /unidades/cliente1,rw,no_root_squash
192.168.0.71 /unidades/cliente2,rw,no_root_squash
```

De esta manera compartiremos dos unidades para las máquinas 192.168.0.70 y 192.168.0.71. Estas unidades serán accesibles con todos los privilegios. Este es un gran problema con NFS, ya que cualquier “gracioso” armado con un portátil y una tarjeta de red con cualquiera de estas IP’s podría gastarnos una “broma”. Afortunadamente disponemos de “diversos” métodos para “disuadir” a esta gentecilla. Aún así, expondremos los que no conllevan violencia. En más común consiste en montar un servidor DHCP que asigne IP’s sólo equipos con MAC conocida. De esta forma orientamos la seguridad a usuarios con una tarjeta de red determinada. Ésta es una buena medida. De hecho es la más habitual. Pero también complicada. Otra manera mucho más sencilla, y elegante para pequeñas redes privadas consiste en establecer entradas estáticas en nuestra tabla de ARP. La tabla ARP del kernel relaciona una IP con una MAC, y permite un nivel de seguridad aceptable.

La siguiente línea muestra cómo especificarle al kernel que una IP sólo será poseída por una máquina con dirección física 0004DD706542. Aunque otra máquina tenga esta IP, no podrá montar las unidades NFS de nuestro equipo.

```
# arp
Address      HWtype  HWaddress  Flags Mask  Iface
192.168.0.70 ether    00:04:DD:70:65:42  CM        eth0
```

Una vez configurado el servidor, debemos intentar comprobar si realmente funciona. Para ello intentaremos montar alguno de los directorios compartidos desde otra máquina. Si esto funciona, ya tendremos una tercera parte de nuestro trabajo terminada. Sólo queda almacenar un sistema arrancable en el directorio compartido, y preparar el disquete de arranque del cliente.

Configuración del sistema de archivos a exportar

El directorio a exportar debe contener todos los ficheros necesarios para el arranque de la máquina y su utilización. Lo imprescindible serían los directorios, etc, lib, usr... En el BootDisk.HOWTO.txt se incluyen los ficheros mínimos para un arranque, pero no interesa complicarnos la vida en exceso. Basta con instalar los paquetes del sistema base en dicha partición, o copiar “*cp -a /etc /unidades/cliente1/etc*” los ficheros desde nuestro sistema principal. Una vez instalado el sistema base, procedemos a configurarlo. A continuación exponemos los pasos más comunes en la configuración inicial. Con *cd /unidades/cliente1/ chroot* cambiamos el directorio raíz de nuestra shell, y trabajamos “como” si hubiésemos arrancado con esa partición como raíz.

Una vez hecho esto, configuraremos el hostname del cliente, instalamos los clientes de X, telnet, ssh, o lo que necesitemos. Si el cliente es Debian, usaremos *dselect, apt-get, dpkg, alien...* o *rpm* si usamos RedHat, Suse... Interesa dejarle los parámetros de red convenientemente configurados, y en particular los clientes de ftp, telnet, ssh, etc. Hecho esto, y lo que el usuario considere conveniente, daremos por terminada esta fase, y pasaremos a configurar el cliente.

4.8.2. Configuración del cliente

Esta parte depende en gran medida del tipo de máquina de la que dispongamos y de su sistema de arranque. En principio este documento está pensado para arquitecturas i386 (es decir, PC's basados en micros de AMD o Intel.), si bien las diferencias en el sistema de arranque de un PC, y otras plataformas son salvables.

Arranque en plataformas i386

El arranque de una máquina i386 se realiza en los siguientes pasos.

- POST (*Power On Self Test*)
- Activación de gestor de arranque
- Carga del kernel (SO)
- Montado del sistema ROOT

El elemento encargado de arrancar la máquina es la BIOS. Las funciones de la BIOS son: comprobar los elementos básicos del sistema (POST), inicializarlos y buscar dispositivos ROM con código a ejecutar. Los dispositivos ROM son los encargados de realizar tareas que nuestra BIOS no puede hacer y que son necesarias para el arranque. Un ejemplo concreto es el sistema de arranque de las controladoras SCSI. No podríamos arrancar una máquina con discos SCSI sin haberlos inicializado previamente, y para ello la tarjeta SCSI lleva un pequeño programa (firmware) en una memoria de sólo lectura con el programa de inicialización. Este programa se carga en memoria y se ejecuta. Es la BIOS la que hace la carga del programa, y la CPU la encargada de su ejecución.

DTC 3210 SCSI BIOS

Searching for devices:

```
[ID 0] .    DTC 3210 SCSI Host Controler
[ID 1] .    YAMAHA CD RW Drive
[ID 2] ...  Not Found
[ID 3] ...  Not Found
[ID 4] ...  Not Found
[ID 5] ...  Not Found
[ID 6] ...  Not Found
[ID 7] ...  Not Found
```

Otro dispositivo ROM conocido es la BOOT-ROM de algunas tarjetas de red local. No es algo común. Pero puede adquirirse fácilmente en tiendas especializadas. Una ROM de arranque contiene un programa que inicializa la tarjeta de red, y la emplea para conectarse a un servidor. El servidor nos envía una copia del kernel a través de nuestra tarjeta. Dicha imagen se lleva a memoria principal, y se ejecuta. Una vez hecho esto, sólo queda montar un sistema de archivos como raíz. Pero eso es algo que se explica un poco más adelante.

En caso de no tener ninguna BOOT-ROM, la BIOS busca un dispositivo del que cargar el kernel. Generalmente es una disquetera, un disco, o una unidad de cdrom. El proceso de carga de este kernel requiere su localización dentro de la unidad. (Puede encontrarse más documentación sobre el tema en Bootdisk-HOWTO).

El kernel es el encargado de inicializar y mantener en marcha el sistema. El kernel reside en una unidad de disco, y es la BIOS la que lo busca en los dispositivos de arranque. Si el kernel está en un disquete o en un disco IDE, entonces la BIOS estándar es capaz de encontrarlo y cargarlo. Sin embargo, si el kernel reside en un disco SCSI, entonces la BIOS no puede cargar el kernel. Tiene que recurrir a las rutinas almacenadas en la ROM de la controladora SCSI. De igual manera, si el kernel reside en una máquina remota, entonces hay que utilizar la ROM de la tarjeta de red para el arranque.

Creación del kernel para arranque vía Ethernet

El kernel de la máquina cliente debe ser capaz de configurar la red, y de pedir archivos de un servidor NFS. Para ello no hay más remedio que recompilar el kernel. La compilación debe llevar al menos las siguientes opciones:

```
[NFS file system support.]
[root file system over nfs.]
[IP Kernel level autoconfiguration.]
(Soporte para tu tarjeta de red)
```

Creación del disquete de arranque con SYSLINUX

Hay muchos métodos para generar disquetes de arranque. Los disquetes de arranque tienen: un gestor de arranque, y una o varias imágenes del kernel. El gestor se encarga de arrancar con un kernel u otro y pasarle una serie de parámetros. El kernel es la pieza central de nuestro sistema operativo, y una vez cargado debe completar el arranque del sistema. Para ello debe saber dónde está el resto del sistema, (Directorio raíz), y opcionalmente algunas cosas sobre nuestro hardware. Un gestor permite arrancar un sistema operativo u otro de forma sencilla, y pasarle parámetros para indicarle su forma de trabajo. Los gestores más conocidos y utilizados son: Lilo y Syslinux, Lilo se utiliza principalmente en instalaciones sobre disco duro, y syslinux se utiliza para generar disquetes de arranque. Syslinux utiliza un disquete formateado para msdos (FAT16) con ciertas peculiaridades. Este disquete contiene unos ficheros de texto con la configuración del sistema, y un binario con el gestor.

Para poder arrancar de un disquete de syslinux, tenemos que previamente formatearlo con la utilidad syslinux, y acto seguido, montarlo y personalizarlo. Si no queremos hacer todo esto, basta con sobrescribir con disco de arranque de cualquier distribución de Linux. En nuestro ejemplo hemos utilizado un disquete de una Debian GNU 2.2r3 (Potato)

Nuestro disquete contiene al menos los siguientes ficheros:

```
-rwxr-xr-x  1 sps      misc      3349 Aug 29  2000 config.gz
-rwxr-xr-x  1 sps      misc      1243 Aug 29  2000 debian.txt
-rwxr-xr-x  1 sps      misc       838 Aug 29  2000 f1.txt
-rwxr-xr-x  1 sps      misc       774 Aug 29  2000 f10.txt
-rwxr-xr-x  1 sps      misc       865 Aug 29  2000 f2.txt
-rwxr-xr-x  1 sps      misc      1139 Aug 29  2000 f3.txt
-rwxr-xr-x  1 sps      misc      1239 Aug 29  2000 f4.txt
-rwxr-xr-x  1 sps      misc      1272 Aug 29  2000 f5.txt
-rwxr-xr-x  1 sps      misc      1427 Aug 29  2000 f6.txt
-rwxr-xr-x  1 sps      misc       786 Aug 29  2000 f7.txt
-rwxr-xr-x  1 sps      misc      1397 Aug 29  2000 f8.txt
-rwxr-xr-x  1 sps      misc      1383 Aug 29  2000 f9.txt
-rwxr-xr-x  1 sps      misc      1476 Aug 29  2000 install.sh
-r-xr-xr-x  1 sps      misc      5860 Aug 29  2000 ldlinux.sys
-rwxr-xr-x  1 sps      misc    1042807 Aug 29  2000 linux
-rwxr-xr-x  1 sps      misc       634 Aug 29  2000 rdev.sh
-rwxr-xr-x  1 sps      misc      1120 Aug 29  2000 readme.txt
-rwxr-xr-x  1 sps      misc     79465 Aug 29  2000 sys_map.gz
-rwxr-xr-x  1 sps      misc      1093 Aug 29  2000 syslinux.cfg
-rwxr-xr-x  1 sps      misc         7 Aug 29  2000 type.txt
```

fX.txt → Ficheros txt con la ayuda a mostrar con la tecla F(x)

syslinux.cfg → Configuraciones y parámetros de arranque

linux → Imagen del kernel

Este disquete está formateado en FAT16, con lo cual puede ser modificado bajo Linux y MSDOS. Al arrancar la máquina, SYSLINUX muestra una ayuda antes de arrancar un kernel determinado. Dicha ayuda se mostrará al pulsar las teclas F1 a F12. De ahí los nombres de los ficheros. SYSLINUX.CFG Contiene lo siguiente:

```
-- # see /usr/doc/syslinux/syslinux.doc.gz for file format description
DEFAULT linux APPEND vga=normal noinitrd load_ramdisk=1
prompt_ramdisk=1 ramdisk_size=16384 root=/dev/fd0 disksize=1.44

TIMEOUT 0
DISPLAY debian.txt
PROMPT 1
F1 f1.txt
F2 f2.txt
F3 f3.txt
F4 f4.txt
F5 f5.txt
F6 f6.txt
F7 f7.txt
F8 f8.txt
F9 f9.txt
F0 f10.txt
LABEL initrd
```

Lo que vamos a añadir es:

```
DEFAULT nfs
LABEL nfs
  KERNEL linux
  APPEND vga=normal noinitrd root=nfsroot nfsroot=192.168.0.10:/unidades/cliente1
        ip=192.168.0.70:192.168.0.10::255.255.255.0:CLiente1:eth0:0
```

Los parametros utilizados son:

- root=NFSROOT ← Indica dónde buscar el directorio raíz.
- nfsroot=192.168.0.10:/unidades/cliente1 ← Indica la IP del servidor, y el nombre del directorio compartido.
- ip=192.168.0.70:192.168.0.10:gw-ip:255.255.255.0:Cliente1:eth0:0 ← Indica: IP del cliente, IP del servidor, gateway, máscara de red local, nombre del cliente, nombre del dispositivo que da servicios de red, y modo.
 - IP cliente: Es la IP del ordenador que arranca utilizando recursos de red.
 - IP servidor: Es la IP del servidor de ficheros.
 - gw IP: Es la IP de un posible router que permitiese la conexión entre cliente y servidor.
 - Netmask: Es la máscara de la red local.
 - Nombre del cliente: Es el "hostname" del cliente.
 - Nombre del dispositivo: Es el nombre del dispositivo que da servicios de red. Los dispositivos válidos para red local son: eth0, eth1...
 - Modo: Indica si debe utilizarse el campo IP cliente, o hacer peticiones de DHCP para obtener una IP.

Una vez modificado el fichero, sólo queda copiar el kernel que hemos compilado previamente, y probar.

4.9. Configuración de la red:

4.9.1. ifconfig

Ifconfig es un programa que gestiona los dispositivos de red conectados a tu equipo y los enlaza con interfaces de red. Ejemplos de dispositivos de red posibles son tarjetas de red (interfaz ethX), modems (pppX), puerto paralelo

(plipX), infrarrojos ... Una vez que tengamos la tarjeta de red bien instalada, (es decir, cargados los módulos correspondientes o compilado en el kernel el soporte para la tarjeta) tendremos que crear un interfaz que será por el que se envíe la información a los equipos conectados a este dispositivo de red. Para ello ejecutaremos el comando *ifconfig*.

Ifconfig es una utilidad de línea de comandos que permite obtener y configurar las interfaces de red de un equipo. Si no se proporcionan argumentos, *ifconfig* muestra el estado de las interfaces de red que se encuentran activas. Si se proporciona una interfaz como argumento, *ifconfig* muestra el estado de dicha interfaz. Si se utiliza con la opción *-a*, muestra el estado de todas las interfaces, incluso aquellas que se encuentren desactivadas. Para configurar una interfaz se debe utilizar el formato:

```
ifconfig <interfaz> <familia> <dir_ip> netmask <máscara>
        broadcast <dir_broadcast> up
```

- En *interfaz* debe proporcionarse el nombre de la interfaz de red que se desea configurar. Generalmente el nombre de interfaz se forma a partir de un nombre de manejador seguido de un número de unidad. El nombre de manejador para redes Ethernet es *eth* y las unidades comienzan a numerarse a partir de 0 hasta el número de interfaces existentes del mismo tipo menos uno (*eth0*, *eth1*, etc.).
- En *familia* se debe proporcionar el nombre de una familia de direcciones soportada por el sistema. Este nombre se utilizará para decodificar y mostrar en un formato inteligible todas las direcciones de protocolo. Las familias de direcciones más comúnmente utilizadas son *inet* para TCP/IP, *inet6* para IP versión 6 e *ipx* para Novell IPX.
- *dir_ip* es la dirección IP con que se desea configurar la interfaz de red.
- *máscara* establece la máscara de red que se desea utilizar para la interfaz. Si no se proporciona este valor, se utilizarán las máscaras de red por defecto para direcciones clase A, B o C en función de la dirección IP con que se esté configurando esta interfaz.
- *dir_broadcast* al proporcionar una dirección de broadcast con la opción *broadcast*, se indica a la interfaz de red que se desea que habilite el modo de broadcast dirigido y que contemple dicha dirección. La dirección de broadcast dirigido a una red se determina a partir de la dirección IP de cualquiera de los equipos pertenecientes a dicha red y su máscara de red, ya que se forma a partir la dirección de red poniendo '1's en la parte de dirección correspondiente a equipos.

El comando *ifconfig* presenta muchas otras opciones que pueden ser consultadas mediante la ayuda en línea de Linux (*man ifconfig*). Una de ellas, útil para el desarrollo de la práctica, es la opción *mtu valor*, que permite establecer la unidad máxima de transferencia (MTU, del inglés Maximum Transfer Unit) que se desea utilice la interfaz que está configurándose. Antes de reconfigurar una interfaz de red, es conveniente desactivarla utilizando la opción *down*. Utilice para ello el formato *ifconfig interfaz down*

Interfaces

En la distribución Debian, existe el archivo: */etc/network/interfaces*. Este archivo, contiene la configuración de la red que leerán los comandos *ifup* e *ifdown*. El formato de este archivo para una configuración de un dispositivo ethernet, es semejante al que sigue:

```
[Tue 4-12:01] duende@acm:~ > cat /etc/network/interfaces
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The first network card - this entry was created during the Debian installation
# (network, broadcast and gateway are optional)
iface eth0 inet static
    address 138.100.13.202
    netmask 255.255.248.0
    network 138.100.8.0
    broadcast 138.100.15.255
    gateway 138.100.8.125
```

A continuación comentaremos las entradas del archivo: En este equipo hay una entrada del primer dispositivo ethernet, la tarjeta de red. **address**: Es la dirección ip de la tarjeta en cuestión. Para las redes de área local existen unos rangos de direcciones reservadas:

Máscara de red	Direcciones de red
255.0.0.0	10.0.0.0 - 10.255.255.255
255.255.0.0	172.16.0.0 - 172.31.255.255
255.255.255.0	192.168.0.0 - 192.168.0.255

netmask : es la máscara de subred asociada a la dirección, depende de cual se haya elegido en la tabla anterior, tendremos que colocar una u otra.

network : es la dirección de la red. Es aplicar la operación AND entre la máscara y una dirección ip de cualquier equipo.

broadcast : es la dirección de broadcast de la red, la de broadcast es aquélla que envía un paquete a todos los equipos conectados a la red. Se halla aplicando una OR entre la máscara de red invertida y la dirección de red.

```

138.100.10.201  10001010 01100100 00001010 11001001
255.255.248.0  11111111 11111111 11111000 00000000
AND
Red 138.100.8.0  10001010 01100100 00001000 00000000

Mask 0.0.7.255  00000000 00000000 00000111 11111111
Red 138.100.8.0  10001010 01100100 00001000 00000000
138.100.15.255 10001010 01100100 00001111 11111111
    
```

gateway : En este campo colocaremos la dirección del Gateway que nos da acceso a internet. Esto sólo es de utilidad para aquellos que accedan a otras redes a través de un router o un gateway. Ej: Usuarios de ADSL con el 3COM.

4.10. Enlace con la red de redes, Internet

4.10.1. A través de un módem: pppconfig

Para conectar nuestro equipo a internet podemos usar el programa *pppconfig*:

Es recomendable crear un link en `/dev/` desde el dispositivo donde tengamos el modem (usualmente un `/dev/ttySX`) a `/dev/modem`, `ln -s /dev/ttySX /dev/modem` así es más fácil configurar todas los programas que accedean al módem.

4.10.2. A través de un router: route

Para configurar nuestro equipo para acceder a internet a través de un router, configuración típica de los usuarios de ADSL habrá que usar el comando *route*. Para que dos equipos intercambien datagramas IP, ambos deberán tener una ruta al otro, o utilizar un gateway por omisión que conozca una ruta. Normalmente, los gateways intercambian información entre ellos utilizando un protocolo como RIP (Routing Information Protocol) u OSPF (Open Shortest Path First). Puesto que algunos sistemas operativos como Windows NT no han proporcionado tradicionalmente una implementación para estos protocolos gateway, si se deseaba utilizar un equipo NT como gateway, debía configurarse manualmente su tabla de rutas. En todo caso, existen multitud de situaciones en las que resulta útil visualizar y modificar el contenido de la tabla de rutas de un equipo. El comando *route* se utiliza para visualizar y modificar la tabla de rutas. *Route print* muestra una lista con las rutas actuales conocidas por IP para el equipo. *Route add* se utiliza para añadir rutas a la tabla de rutas, y *route del* se utiliza para borrar rutas de la tabla. Todos los nombres simbólicos usados para el destino se buscan en el archivo de la base de datos de la red `/etc/networks`. Los nombres simbólicos para la puerta de acceso se buscan en el archivo de la base de datos de nombres de equipos `/etc/hosts`. Si no se puede resolver un nombre simbólico en base al contenido de estos ficheros, se recurrirá al servicio de nombres de dominio. *Route*, por omisión, muestra la tabla de rutas intentando determinar nombres simbólicos. Si se desea que muestre direcciones IP en lugar de nombres simbólicos, deberá utilizarse con la opción `-n`. Para añadir a la tabla de rutas una ruta hacia una red se utilizará el siguiente formato:

```
route add -net <dir_ip> netmask <máscara> [gw <dir_ip_gateway>] dev <interfaz>
```

en el que los argumentos tienen el siguiente significado:

- *dir_ip*. Route puede utilizarse para establecer rutas estáticas hacia equipos y redes vía una interfaz de red una vez que se ha configurado esta última mediante la utilidad *ifconfig*. La mayor parte de las veces se utilizará para establecer rutas hacia redes, por lo que debe utilizarse con la opción `-net`, con lo que `dir_ip` se interpretará como una dirección de red para la que se desea establecer una ruta. Si el servicio de nombres de dominio DNS está configurado y funcionando correctamente, puede utilizarse un nombre simbólico en lugar de una dirección IP. Si se desea establecer una ruta a un equipo, deberá utilizarse la opción `-host`.

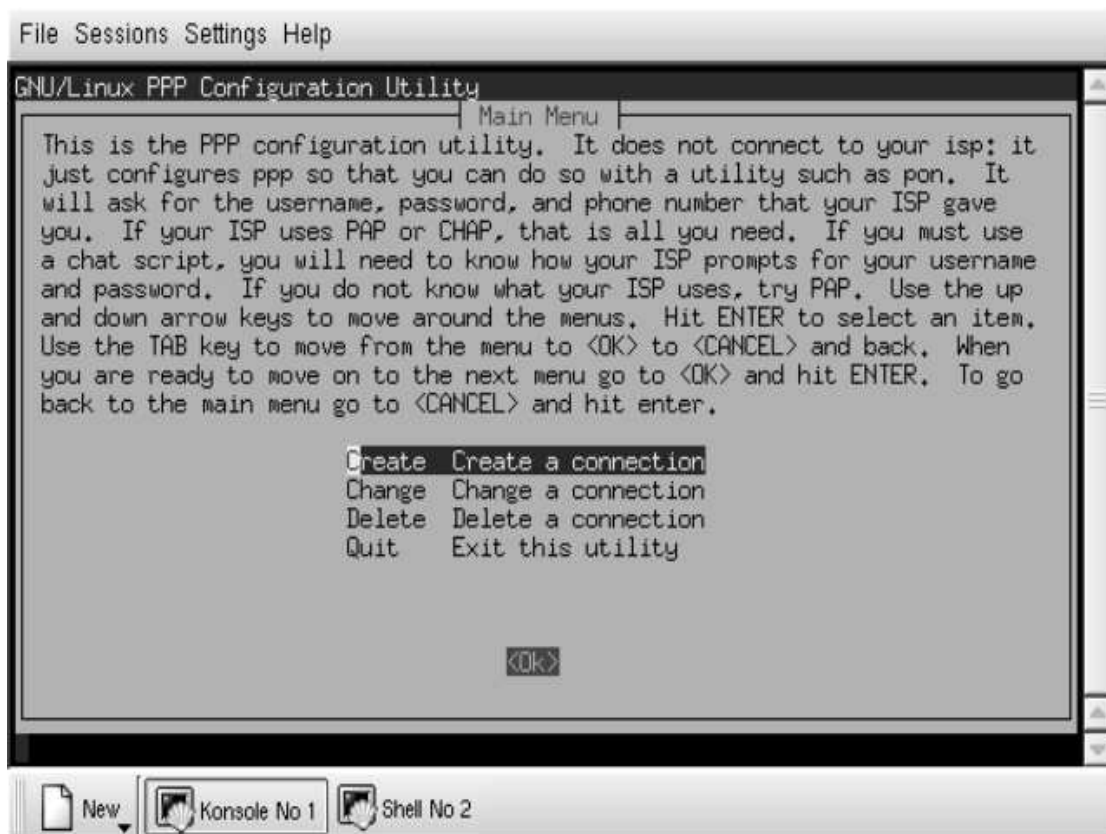


Figura 4.2: Configurando la conexión con pppconfig

- *máscara* especifica la máscara de red para la ruta que se desea añadir. Durante el proceso de encaminamiento de cada paquete IP se realizará una operación lógica AND entre la dirección IP destino contenida en su cabecera y esta máscara. Si el resultado de dicha operación coincide con la *dir_ip*, se utilizará esta entrada de la tabla de rutas para realizar el encaminamiento de dicho paquete IP. Si no se indica una máscara de red, se utilizará una de las máscaras por omisión para las clases de direcciones A, B y C en función de la clase a la que pertenezca *dir_ip*. Si el destino de la ruta es un equipo, deberá utilizarse como máscara 255.255.255.255.
- *dir_ip_gateway*. Cuando se utiliza la entrada que se está creando en la tabla de rutas para enviar un paquete IP, dicho paquete se encaminará a través del *gateway* indicado en esta opción. Dicho *gateway* debe poder ser alcanzado, por lo que previamente deberá haberse establecido una ruta hacia el mismo (por ejemplo utilizando el comando *route add* con la opción *-host*). Si se especifica la dirección IP de una de las interfaces de red locales, se utilizará ésta para decidir la interfaz hacia la que se encaminarán los paquetes. Es opcional, solo aparece cuando es necesario encaminar hacia un gateway.
- En *interfaz* debe proporcionarse el nombre de la interfaz de red que se desea utilizar en esta ruta. Generalmente el nombre de interfaz se forma a partir de un nombre de manejador seguido de un número de unidad. El nombre de manejador para redes Ethernet es *eth*. Las unidades comienzan a numerarse a partir de 0 (*eth0*, *eth1*, etc.).

Para eliminar una ruta debe utilizarse la sintaxis

```
# route del -net <dir_ip> <num_unos_mascara> <Destination> gw <Gateway> netmask <Genmask>
```

Donde:

- *num_unos_mascara* es un entero que indica el número de '1's que tiene la máscara de dicha red. A continuación se describen brevemente algunos de los campos de la tabla de rutas mostrada por el comando *route*. Para obtener una descripción completa de todos los campos, así como del resto de opciones utilizadas por el comando *route*, puede consultarse la ayuda en línea de Linux (*man route*).
- El campo *Destination* muestra el equipo o la red destino de una ruta. Puede visualizarse tanto su dirección IP como su nombre simbólico DNS.

- El campo *Gateway* muestra la dirección IP o el nombre simbólico del router utilizado en una ruta. Un * en este campo significa que no se utiliza ningún router para dicha entrada.
- El campo *Genmask* muestra la máscara para la red destino. Si el destino es un equipo, este campo muestra la máscara 255.255.255.255.
- Los flags más frecuentemente mostrados en el campo *Flags* tienen el siguiente significado: U Indica que la ruta se encuentra activa. H indica que el destino es un equipo, no una red y G Indica que se utiliza un Gateway para encaminar.
- El campo *Iface* indica la interfaz de red a la que se enviarán los paquetes para una ruta determinada.

Capítulo 5

Prácticas en Linux

5.1. Introducción

El soporte para la programación en Linux es muy amplio, tanto es así que en cualquier distribución podemos encontrar compiladores, depuradores o intérpretes de prácticamente la mayoría de los lenguajes de programación existentes. Además de la fiabilidad y robustez propias de Linux, también se nos proporcionan editores con funciones orientadas a la programación, así como potentes herramientas (como es el caso del comando *make*) para la gestión y compilación de nuestros proyectos.

En esta sección explicaremos cómo instalar y utilizar los compiladores de Ada y Haskell bajo linux, así como GtkAda, paquete necesario para compilar aplicaciones gráficas programadas en Ada.

5.2. Instalación del gnat

El compilador de Ada viene en los cds de muchas de las distribuciones mas comunes de linux, en caso de no tenerlo tendremos que buscar las fuentes de internet (www.freshmeat.net p ej), bajárnoslas y compilarlas tal y como se describe más adelante.

5.2.1. Instalación con apt-get en debian

En el caso de debian es muy sencillo instalar el gnat, simplemente ejecutaremos *apt-get* y el se encargará de buscarlo en su base de datos, pedirnos el cd correspondiente e instalarlo correctamente

```
[09:19pm]Andromeda:/instalando # apt-get install gnat
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  gnat
0 packages upgraded, 1 newly installed, 0 to remove and 2 not upgraded.
Need to get 0B/7656kB of archives. After unpacking 29.5MB will be used.
Media Change: Please insert the disc labeled 'Debian GNU/Linux 2.2 r0 _Potato_
- Official i386 Binary-2 (20000814)' in the drive '/cdrom/' and press enter

Selecting previously deselected package gnat.
(Reading database ... 59113 files and directories currently installed.)
Unpacking gnat (from ../devel/gnat_3.12p-5.deb) ...
Setting up gnat (3.12p-5) ...
```

5.2.2. Instalación con dpkg

Si lo que tenemos es el gnat para otras distribuciones siempre lo podemos instalar “a mano” transformándolo previamente con *alien* a un paquete soportado por nuestra distribución y posteriormente instalándolo con dpkg (debian), rpm (redhat) ... En el siguiente ejemplo se transforma un paquete rpm de RedHat a un .deb y posteriormente lo instalamos con dpkg

```
[09:28pm]Andromeda:/instalando # alien gnat_3.13p-8_i386.rpm
```

```

95710 blocks
dh_testdir
# Nothing to do.
dh_testdir
dh_testroot
dh_clean -k
dh_installdirs
cp -a 'ls |grep -v debian' debian/tmp
dh_installdocs
dh_installexamples
dh_installmenu
dh_installcron
dh_installchangelogs
utmp entry ("victor") does not match value of LOGNAME ("root");
using "root" at
/usr/lib/dpkg/controllib.pl line 47.
dh_compress
dh_suidregister
dh_installdeb
dh_shlibdeps
utmp entry ("victor") does not match value of LOGNAME ("root");
using "root" at
/usr/lib/dpkg/controllib.pl line 47. dpkg: not found.
dpkg-shlibdeps: warning: could not find any packages for
(libgnat-3.13p.so.1)
dpkg-shlibdeps: warning: unable to find dependency information for shared
library libgnat-3.13p (soname 1, path , dependency field Depends)
dh_gencontrol
utmp entry ("victor") does not match value of LOGNAME ("root");
using "root" at /usr/lib/dpkg/controllib.pl line 47.
utmp entry ("victor") does not match value of LOGNAME ("root");
using "root" at
/usr/lib/dpkg/controllib.pl line 47.
dh_makeshlibs
dh_md5sums
dh_builddeb
dpkg-deb: building package 'gnat' in './gnat_3.13p-8_i386.deb'.
-- Examining gnat-3.13p-7.i386.rpm
-- Unpacking gnat-3.13p-7.i386.rpm
----/usr
-- Automatic package debianization
-- Building the package gnat_3.13p-8_i386.deb

Generation of gnat_3.13p-8_i386.deb complete.
-- Successfully finished

[09:29pm]Andromeda:/instalando # mount /dev/hdc /cdrom
[09:29pm]Andromeda:/instalando # cd
/cdrom/dists/potato/main/binary-i386/devel/
[09:30pm]Andromeda:/cdrom/dists/potato/main/binary-i386/devel #
dpkg -i gnat_3.12p-5.deb
Selecting previously deselected package gnat.
(Reading database ... 59113 files and directories currently installed.)
Unpacking gnat (from gnat_3.12p-5.deb) ...
Setting up gnat (3.12p-5) ...

[09:30pm]Andromeda:/cdrom/dists/potato/main/binary-i386/devel #

```

5.2.3. Instalación a partir del paquete tar.gz

También podemos instalar el gnat a partir del tar.gz que contiene scripts para configurar e instalar el paquete según las opciones que les especifiquemos en doconfig y doinstall. Para ello lo primero que tenemos que hacer es descomprimir el archivo como sigue:

```
[09:36pm]Andromeda:/instalando # ls
gnat-3.13p-i686-pc-linux-gnu-bin.tar.gz
[09:36pm]Andromeda:/instalando # tar zxvf
gnat-3.13p-i686-pc-linux-gnu-bin.tar.gz
```

Una vez descomprimido nos situamos en el directorio creado y ejecutamos la herramienta de configuración 'doconfig' que preparara la instalación del programa

```
[09:37pm]Andromeda:/instalando/gnat-3.13p-i686-pc-linux-gnu-bin # ls
COPYING          crtendS.o  gnatbind   gnatmem    libpthread-2.0.a
Makefile         doconfig  gnatbl     gnatprep   limitx.h
README          examples  gnatbl     gnatpsta   limity.h
README.TASKING  features  gnatelim   gnatpsys   rts-fsu
README.gnatelim float.h    gnatfind   gnatstub   rts-native
README.gnatstub gcov      gnathtml.pl gnatxref    specs
cc1             gdb       gnatinfo.txt gsyslimits.h xgcc
cpp            gdbtk     gnatkr     libaddr2line.a
crtbegin.o     ginclude  gnatlink   libgcc.a
crtbeginS.o    glimits.h gnatls     libgthreads.a
crtend.o       gnat1     gnatmake   libmalloc.a
[09:37pm]Andromeda:/instalando/gnat-3.13p-i686-pc-linux-gnu-bin #
./doconfig
```

There are 2 options for installation:

- 2) Install GCC C compiler and GNAT files in the standard GNAT locations.
(Note: This includes directories under /usr/gnat. On most systems, this requires root permission).
- 3) Install GCC C compiler and GNAT files in non-standard locations that you will specify.

Options 2 provides simplest and most flexible use of GNAT.

Type 2, or 3 (then RETURN) to choose an option:

2

En este punto nosotros seleccionamos la opción 2 por lo que el gnat se instalará en la ruta por defecto, si quisieramos instalarlo en un directorio en concreto deberiamos elegir la opción 3

The installation in the GCC standard locations will install:

```
In /usr/gnat/bin :
gcc gnatbind gnatbl gnatpsta gnatpsys gnatxref gcov gdb gdbtk
gnatmake      gnatelim gnatstub addr2line objdump
```

```
In /usr/gnat/lib/gcc-lib/i686-pc-linux-gnu/2.8.1 :
gnat1 cc1 cpp ld libgcc.a specs
and libgthreads.a
```

```
In /usr/gnat/lib/gcc-lib/i686-pc-linux-gnu/2.8.1/adainclude :
The source files of the RTL
```

```
In /usr/gnat/lib/gcc-lib/i686-pc-linux-gnu/2.8.1/adalib :
    The object & ali files of the RTL
    and libgnat.a
Configuration complete. Run doinstall to do the installation.
DO NOT FORGET: put /usr/gnat/bin at the front of your PATH
```

Con esto ya deberíamos tener configurado el programa de instalación, algo importante es que no olvidemos añadir en el PATH el directorio /usr/gnat/bin para que podamos ejecutar el compilador desde cualquier directorio, para ello modificaremos el archivo /etc/profile o el ~/.bash_profile y añadiremos a la línea donde se define el PATH='... :/usr/gnat/bin'

Ahora solo queda ejecutar la instalación a través de ./doinstall

```
[09:42pm]Andromeda:/instalando/gnat-3.13p-i686-pc-linux-gnu-bin #
./doinstall
```

Una vez hecho esto ya tenemos disponible el compilador de Ada, pudiendo invocarlo en cualquier momento mediante el comando *gnatmake*

5.2.4. Uso del comando gnatmake

La sintaxis del comando es la siguiente:

```
gnatmake opciones nombre [-cargs opts] [-bargs opts] [-largs opts]
```

en donde *nombre* es el nombre del fichero a compilar
opciones es uno de los siguientes argumentos:

gnatmake switches:

```
-a      Consider all files, even readonly ali files
-c      Compile only, do not bind and link
-f      Force recompilations of non predefined units
-i      In place. Replace existing ali file, or put it with source
-jnum   Use nnn processes to compile
-k      Keep going after compilation errors
-m      Minimal recompilation
-M      List object file dependences for Makefile
-n      Check objects up to date, output next file to compile if not
-o name Choose an alternate executable name
-q      Be quiet/terse
-v      Motivate all (re)compilations
```

```
-z      No main subprogram (zero main)
```

```
--GCC=command      Use this gnatgcc command
--GNATBIND=command Use this gnatbind command
--GNATLINK=command Use this gnatlink command
```

Gnat/Gnatgcc switches such as -g, -O, -gnato, etc. are directly passed to gnatgcc

y *-cargs opts*, *-bargs opts*, *-largs opts* son opciones que se le pueden pasar directamente al compilador, al binder y al linker respectivamente.

Ejemplo

```
[06:26pm]victor@Orion:~/facultad/ed1/Practicas/Practical > ls
cadenas_entrada.adb  colas.adb  inf_postf.adb  pilas.adb
prueba.adb  tokens.ads
cadenas_entrada.ads  colas.ads  inf_postf.ads  pilas.ads
tokens.adb
```

```
[06:26pm]victor@Orion:~/facultad/ed1/Practicas/Practical > gnatmake prueba
gnatgcc -c prueba.adb
gnatgcc -c cadenas_entrada.adb
gnatgcc -c colas.adb
gnatgcc -c inf_postf.adb
gnatgcc -c tokens.adb
gnatgcc -c pilas.adb
gnatbind -x prueba.ali
gnatlink prueba.ali
[06:26pm]victor@Orion:~/facultad/ed1/Practicas/Practical > ls
cadenas_entrada.adb colas.adb inf_postf.adb pilas.adb prueba
tokens.adb
cadenas_entrada.ads colas.ads inf_postf.ads pilas.ads prueba.adb
tokens.ads
cadenas_entrada.ali colas.ali inf_postf.ali pilas.ali prueba.ali
tokens.ali
cadenas_entrada.o colas.o inf_postf.o pilas.o prueba.o
tokens.o
[06:26pm]victor@Orion:~/facultad/ed1/Practicas/Practical >
```

5.3. Instalación de GTK-Ada

5.3.1. Instalación compilado las fuentes (tar.gz)

También podemos bajar las fuentes de los programas que queramos instalar y compilarlas nosotros mismos, éste procedimiento está bastante estandarizado y la mayoría de las veces uno no tiene más que recordar 3 o 4 comandos para instalar cualquier programa que se baje a partir de las fuentes, claro que siempre pueden ocurrir problemas (programas que no compilan por estar todavía en desarrollo, dependencias a librerías que no tenemos instaladas...) por lo que siempre es recomendable instalar los programas a partir de los paquetes diseñados para las distribuciones, aun así hay programas que no se encuentran "empaquetados" para las distribuciones por lo que no nos quedará más remedio que abordar el tar.gz.

Éste es el caso del GtkAda, así que lo primero que haremos, como antes, será descomprimir el archivo con el comando tar:

```
[11:32pm]Andromeda:/instalando # ls
GtkAda-1.2.11.tgz          gnat-3.13p-i686-pc-linux-gnu-bin.tar.gz
gnat-3.13p-i686-pc-linux-gnu-bin
[11:32pm]Andromeda:/instalando # tar zxvf GtkAda-1.2.11.tgz
```

Una vez descomprimidos nos situamos sobre el directorio que se ha creado y ejecutamos *configure* que se encargará de preparar el Makefile necesario para la posterior compilación

```
[11:33pm]Andromeda:/instalando # cd GtkAda-1.2.11
[11:33pm]Andromeda:/instalando/GtkAda-1.2.11 # ls
ANNOUNCE INSTALL      aclocal.m4    configure     docs
known-problems
AUTHORS  Makefile.in  config.guess  configure.in  examples    src
COPYING  README      config.sub    contrib      install-sh  testgtk
[11:33pm]Andromeda:/instalando/GtkAda-1.2.11 # ./configure
```

Llegado a este punto configure realizará una serie de comprobaciones para ver si se puede instalar el programa en cuestión, aquí pueden surgir problemas por no tener instaladas en el sistema librerías necesarias para que GtkAda funcione, a continuación se detalla un ejemplo de ejecución de configure en un sistema que no tiene la versión 1.2.2 de libgtk-dev o superior instalada.

```
checking for gtk-config... no
checking for GTK - version >= 1.2.2... no
*** The gtk-config script installed by GTK could not be found
*** If GTK was installed in PREFIX, make sure PREFIX/bin is in
*** your path, or set the GTK_CONFIG environment variable to the
```

```
*** full path to gtk-config.
configure: error: Test for GTK failed. See the file 'INSTALL' for help.
```

Para arreglar esto tendremos que instalar el paquete libgtk-dev con apt-get, dpkg, rpm ... con versión mayor o igual a la 1.2.2 y volver a ejecutar el configure, si todo ha ido bien debería acabar con unas líneas como éstas:

```
[...]
creating src/gtkada-config
creating src/gate
creating testgtk/Makefile
creating src/gtkextra/Makefile
creating src/opengl/Makefile
creating src/pixbuf/Makefile
creating src/glade/Makefile
creating src/gnome/Makefile
creating docs/gtkada_ug/Makefile
creating docs/gtkada_rm/Makefile
```

Ahora tendremos que compilar, para ello el configure habrá creado un Makefile de acuerdo a las características de nuestra máquina, por lo que simplemente tendremos que ejecutar 'make' para que comience la compilación del gtkada

```
[11:34pm]Andromeda:/instalando/GtkAda-1.2.11 # make
```

Una vez que se ha compilado (siempre que no haya habido errores), procedemos a copiar el programa en los directorios que le corresponda, para ello ejecutaremos 'make install' y con ello finalizara la instalacion.

```
[11:34pm]Andromeda:/instalando/GtkAda-1.2.11 # make install
[...]
```

A partir de ahora ya podremos usarlo, tal y como dice el archivo INSTALL

```
gnatmake <your\_application> <your\_switches> 'gtkada-config'
```

5.4. Instalacion del interprete para Haskell (Hugs)

5.4.1. Instalación con apt-get en debian

Al igual que el gnat, la instalación del hugs a través del apt-get es sumamente sencilla, simplemente tendremos que ejecutar *apt-get install hugs* como root y *apt-get* se encargará de instalar el intérprete de haskell.

```
[11:21pm]Andromeda:/home/victor # apt-get install hugs
Reading Package Lists... Done
Building Dependency Tree... Done
The following NEW packages will be installed:
  hugs
0 packages upgraded, 1 newly installed, 0 to remove and 2 not upgraded.
Need to get 0B/416kB of archives. After unpacking 1561kB will be used.
Media Change: Please insert the disc labeled 'Debian GNU/Linux 2.2 r0 _Potato _
- Official i386 Binary-2 (20000814)' in the drive '/cdrom/' and press enter

Selecting previously deselected package hugs.
(Reading database ... 60297 files and directories currently installed.)
Unpacking hugs (from .../hugs_98.199911-1.deb) ...
Setting up hugs (98.199911-1) ...
```

Una vez instalado, para ejecutarlo sólo tendremos que teclear 'hugs' en cualquier consola

```
[11:23pm]Andromeda:/home/victor # hugs
--  --  --  --  ----  ---
||  || ||  ||  ||  ||  ||__  Hugs 98: Based on the Haskell 98 standard
||__||  ||__||  ||__||  __||  Copyright (c) 1994-1999
```

```

||---||          ___||          World Wide Web: http://haskell.org/hugs
||  ||          Report bugs to: hugs-bugs@haskell.org
||  || Version: November 1999  -----

```

Haskell 98 mode: Restart with command line option -98 to enable extensions

Reading file "/usr/share/hugs98/lib/Prelude.hs":

```

Hugs session for:
/usr/share/hugs98/lib/Prelude.hs
Type :? for help
Prelude>

```

5.4.2. Instalación con rpm otras distribuciones

Para instalar el paquete a mano seguiremos el mismo procedimiento que usamos para instalar el gnat, esto es, transformar el paquete a nuestra distribución con el comando ‘alien’, e instalarlo posteriormente con dpkg, rpm ...

5.4.3. Manejo básico del Hugs

Los comandos más utilizados en el hugs son los siguientes:

:l nombre_archivo Carga el archivo indicado en el intérprete, si se detectan errores serán mostrados por el intérprete.

:r Vuelve a cargar el último archivo cargado (útil tras hacer modificaciones).

:t nombre_var Muestra el tipo al que pertenece la variable nombre_var.

!: comando Ejecuta el comando de shell que se haya indicado.

:q Sale del intérprete.

?: Lista los comandos disponibles.

5.4.4. Instalación del Haskell-Mode para Emacs

El modo haskell para emacs permite trabajar más cómodamente en este editor cuando programamos en Haskell, esto se logra coloreando las palabras clave, gestionando las tabulaciones ... Este modo no viene por defecto con emacs pero afortunadamente podemos encontrar en internet un targz que contiene los archivos necesarios para su funcionamiento, este archivo es el haskell-mode.tar.gz y lo podéis encontrar en “<http://www.haskell.org/haskell-mode/>”

Bien, una vez bajado el archivo a nuestro disco deberemos situarlo en la carpeta ~/lib/emacs de nuestro home y descomprimirlo ahí (si no tenemos dicho directorio lo crearemos con *mkdir*).

```

[07:41pm]victor@Andromeda:~ > mkdir lib
[07:41pm]victor@Andromeda:~ > cd lib
[07:41pm]victor@Andromeda:~/lib > mkdir emacs
[07:41pm]victor@Andromeda:~/lib > cd emacs
[07:41pm]victor@Andromeda:~/lib/emacs > cp
/instalando/haskell-mode.tar.gz .
[07:42pm]victor@Andromeda:~/lib/emacs > tar zxvf haskell-mode.tar.gz
haskell-mode.el
haskell-decl-scan.el
haskell-font-lock.el
haskell-indent.el
haskell-doc.el
haskell-hugs.el
haskell-simple-indent.el
[07:42pm]victor@Andromeda:~/lib/emacs > ls
haskell-decl-scan.el  haskell-hugs.el      haskell-mode.tar.gz
haskell-doc.el       haskell-indent.el    haskell-simple-indent.el
haskell-font-lock.el haskell-mode.el
[07:42pm]victor@Andromeda:~/lib/emacs > rm haskell-mode.tar.gz
[07:42pm]victor@Andromeda:~/lib/emacs >

```

Una vez hecho esto abriremos el fichero `.emacs` situado en nuestro home y le añadiremos las siguientes líneas:

```
(setq load-path (cons "~/lib/emacs" load-path))
(setq auto-mode-alist
  (append auto-mode-alist
    '(("\\. [hg]s$" . haskell-mode)
      ("\\.hi$" . haskell-mode)
      ("\\.l [hg]s$" . literate-haskell-mode))))
(autoload 'haskell-mode "haskell-mode"
  "Major mode for editing Haskell scripts." t)
(autoload 'literate-haskell-mode "haskell-mode"
  "Major mode for editing literate Haskell scripts." t)
(add-hook 'haskell-mode-hook 'turn-on-haskell-font-lock)
(add-hook 'haskell-mode-hook 'turn-on-haskell-decl-scan)
(add-hook 'haskell-mode-hook 'turn-on-haskell-doc-mode)
(add-hook 'haskell-mode-hook 'turn-on-haskell-indent)
(add-hook 'haskell-mode-hook 'turn-on-haskell-simple-indent)
(add-hook 'haskell-mode-hook 'turn-on-haskell-hugs)
```

Con esto simplemente emacs cargara el `haskell-mode` cada vez que abramos un archivo con extension `hs`, tambien podemos cargar este modo a mano con la siguiente combinacion: *M-x haskell-mode*

5.5. Creación de Makefiles para la compilacion

Un Makefile es un archivo de configuración que leerá el comando ‘make’ cuando se le invoque, donde se pueden idicar archivos a compilar, dependencias de compilación, flags a pasarle al compilador ...

Los Makefiles pueden llegar a ser archivos grandes y complejos pero nosotros trataremos de explicar las nociones mas basicas de los mismos.

Proceso de creacion de un arcivo Makefile:

Nos situamos en el directorio donde esten contenidos los ficheros con el codigo a compilar

Creamos el fichero con nombre Makefile que estará estructurado de la siguiente manera:

identificador: lista de archivos a compilar
accion a realizar

Veamos esto con un ejemplo:

```
[07:42pm]victor@Andromeda:~/programacion/ada > cat Makefile
```

```
Interfaz: imagen.adb transformacion.adb nucleo.adb rle.adb qt.adb interfaz\
az.adb
```

```
    gnatmake -g interfaz.adb 'gtkada-config'
```

```
clean:
```

```
    rm *.o *.ali *~ b~* interfaz
```

En este caso tenemos 2 posibles llamadas a make: *make interfaz* y *make clean*. En caso de hacer *make interface* el comando make ejecutara el `gnatmake` sobre cada uno de los archivos a compilar en el orden dado por el *Makefile*, esto es: `imagen.adb`, `transformacion.adb`, `nucleo.adb`... asi hasta `interfaz.adb`. El *make clean* lo que hará será borrar todos los archivos generados tras compilar (`.o`, `.ali` y binario final), esto es útil para volver a dejar el directorio como antes de invocar *make interfaz*.

Capítulo 6

Trabajando y disfrutando

6.1. Introducción

El objetivo de este capítulo es hacer una breve introducción al escritorio KDE y a las aplicaciones del día a día. No entraremos en muchos aspectos técnicos o “trucos”. En cualquier caso, es recomendable visitar <http://www.kde.org/>, <http://www.kde-look.org/>, <http://www.kde-forum.org/>, <http://es.kde.org/> y <http://dot.kde.org/> para ampliar conocimientos y sacar el máximo partido a este entorno de escritorio.

¿Porqué KDE y no otro entorno de escritorio como GNOME, Fluxbox, Windowmaker o IceWM? Básicamente por dos razones: la primera, por simplicidad: esto pretende ser una breve introducción, no una extensa comparativa. Y la segunda, porque KDE es, hoy en día, el entorno de escritorio por defecto en muchas distribuciones.

En este capítulo se cubrirá el uso básico de KDE en general, y de algunas aplicaciones “típicamente KDE”. Hay que tener en cuenta que el usar KDE como entorno de escritorio no evita usar aplicaciones “típicamente GNOME”. En otras palabras, se puede usar aplicaciones del entorno de KDE en GNOME y viceversa.

6.2. El escritorio de KDE y sus aplicaciones

Cuando iniciemos nuestra primera sesión en KDE nos encontraremos con un escritorio muy similar al de la figura:

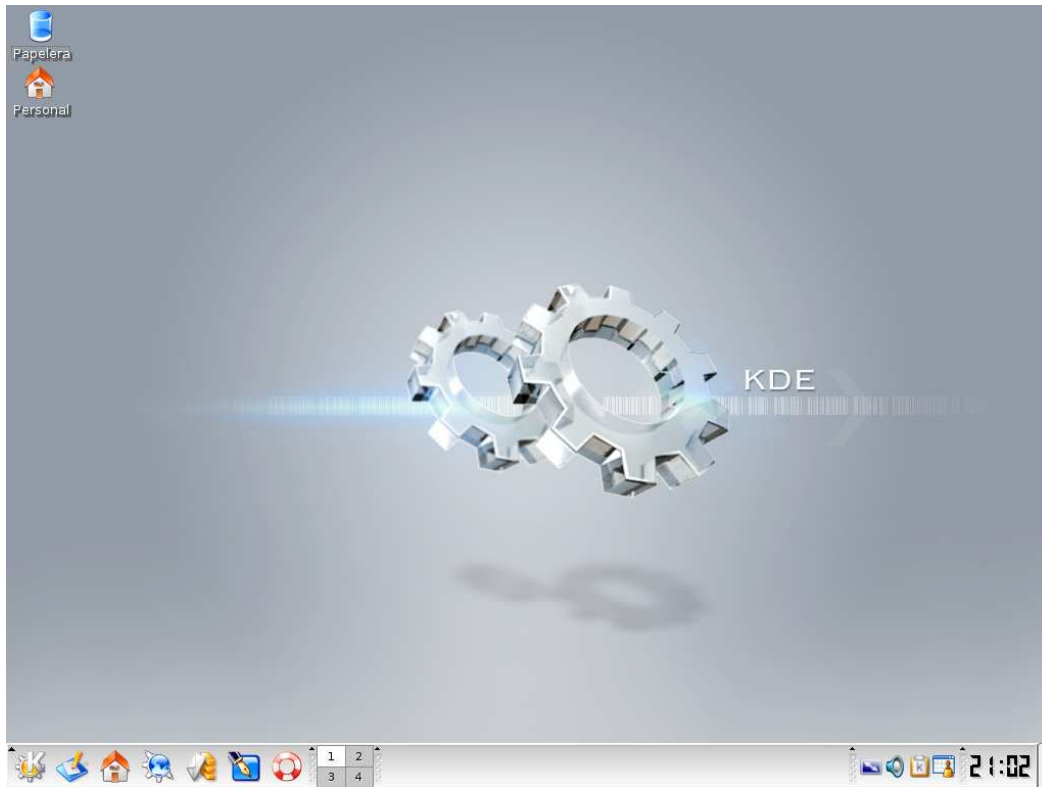


Figura 6.1: Escritorio inicial con Kde

La parte superior de la pantalla es el escritorio en sí; la parte inferior la ocupa el panel de KDE, también llamado kicker. Observando el kicker de izquierda a derecha encontraremos:

- El botón del K-menú, simbolizado normalmente¹ por el logotipo de KDE (una letra “K” dentro de un engranaje). Desde aquí se tiene acceso, mediante menús desplegables, a prácticamente todas las aplicaciones instaladas en el sistema, configuraciones y opciones tales como cerrar sesión, bloquear la pantalla o apagar el ordenador.

¹Diversos distribuidores de GNU/Linux, como por ejemplo SuSE, Fedora o Linspire, cambian el logotipo de KDE por el de su empresa. En muchos casos, el escritorio por defecto en una distribución puede no parecerse al del ejemplo.

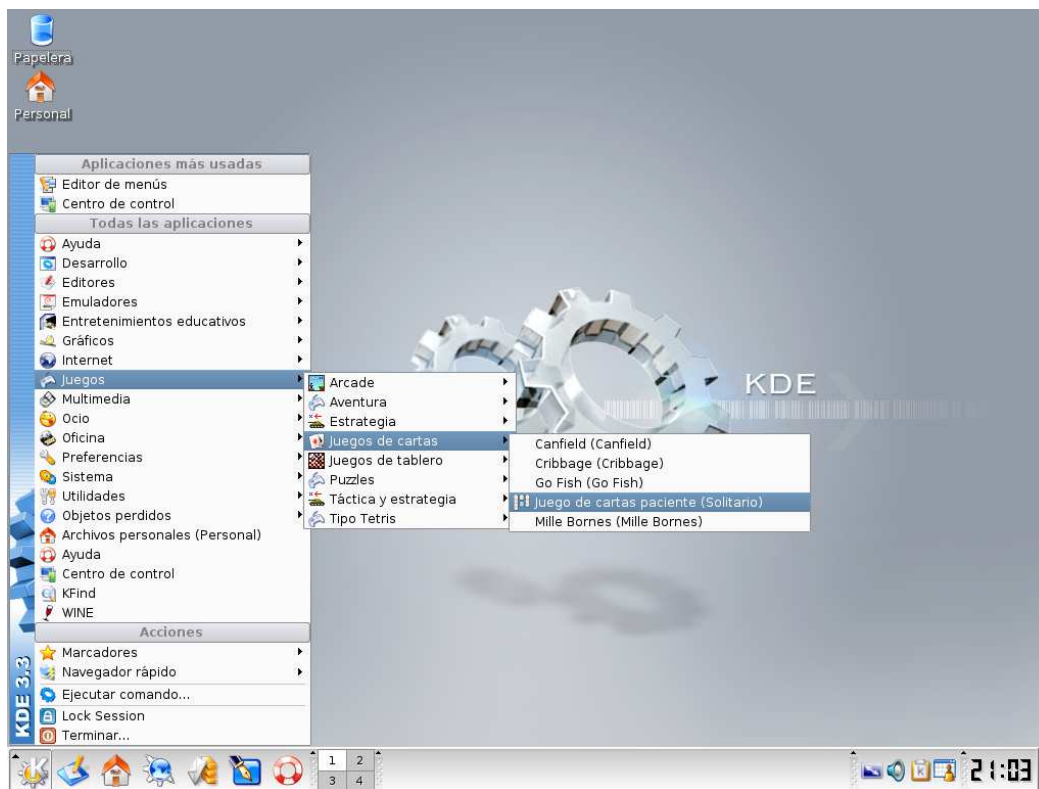


Figura 6.2: Menu de kicker

- El botón para mostrar el escritorio. Al pulsarlo, se minimizan todas las ventanas y el escritorio queda visible.
- Iconos de diversas aplicaciones, que en la figura son: Konqueror como gestor de ficheros, Konqueror como navegador web, Kmail (gestor de correo), Kword (procesador de textos) y Khelpcenter (ayuda y documentación). Al pulsar uno de ellos, se lanza la aplicación correspondiente.
- El paginador. Sirve para cambiar entre escritorios virtuales.
- La barra de tareas. En ella habrá un botón por cada ventana abierta y sirve para pasar a otra aplicación.
- La bandeja del sistema. Determinadas aplicaciones se “anclan” a la bandeja del sistema (aparece un icono en la misma) en vez de mostrar una ventana. Otras aplicaciones (como Juk, Kopete o Amarok) hacen ambas cosas; en esta caso, el estar “anclado” a la bandeja del sistema hace que al cerrar la ventana de la aplicación, ésta no se cierre.
- El reloj. Aparte del uso obvio, se puede configurar para mostrar otra zona horaria o hacer que muestre la hora en texto (“las doce menos cuarto” literalmente). Los más atrevidos pueden cambiarlo por un reloj binario.
- En último lugar, el botón de ocultación manual. Al pulsar sobre él, Kicker se desliza horizontalmente, para dejar visible más escritorio. Para volver a mostrarlo, sólo hay que pulsar otra vez sobre este mismo botón (que permanece visible).

Kicker es (como el resto de KDE) altamente configurable. Los botones de aplicaciones se pueden quitar, poner o cambiar de sitio pulsando con el botón derecho del ratón sobre ellos. Los applets (el reloj, el paginador, la barra de tareas, etc) se configuran/añaden/quitan a través del “asa” del applet (el pequeño separador que aparece a la izquierda de cada uno).

El paginador merece una mención especial. En prácticamente todos los entornos de escritorio en Linux, existe la posibilidad de tener más de un escritorio (en este caso, el paginador nos muestra que tenemos cuatro escritorios virtuales). La potencia radica en que se pueden organizar las aplicaciones por escritorios virtuales, acelerando de esta manera el cambio a otra aplicación distinta (en lugar de buscar la aplicación correspondiente en la barra de tareas, se cambia al escritorio en el que está esa aplicación).

Con respecto al manejo de las ventanas de las aplicaciones, es similar al de muchos otros entornos de escritorios: se cierra una ventana pulsando sobre la “X” en la esquina o pulsando Alt+F4, se mueven arrastrando la barra de

título, se cambia entre ventanas con Alt+Tab, etcétera. Algo que es conveniente saber es que en cualquier entorno de escritorio en Linux se puede mover una ventana dejando pulsado Alt y arrastrando cualquier punto de la misma, y redimensionar dejando pulsado Alt y arrastrando con el botón derecho.

6.2.1. Konqueror, la navaja suiza

Konqueror es una de las aplicaciones “estrella” de KDE. Combina un gestor de ficheros con un navegador web, cliente FTP, y puede integrar prácticamente cualquier otra aplicación en sí (mediante la tecnología de Kparts), como puede ser un editor de texto o un visor de PDFs.

Veamos primero Konqueror como gestor de ficheros. Para lanzarlo, hay que pulsar sobre el icono con forma de casa² que hay en kicker o en el escritorio.

Como se observa en la siguiente figura, la estructura de Konqueror es similar a muchos otros gestores de fichero: un árbol de directorios en la parte izquierda, y en la parte derecha una serie de iconos simbolizando ficheros y directorios. Al pulsar sobre un icono de un fichero, se abrirá con la aplicación correspondiente, y mediante el botón derecho se pueden efectuar diversas acciones (cortar, pegar, abrir con otra aplicación, comprimir, enviar por correo y un largo etcétera).

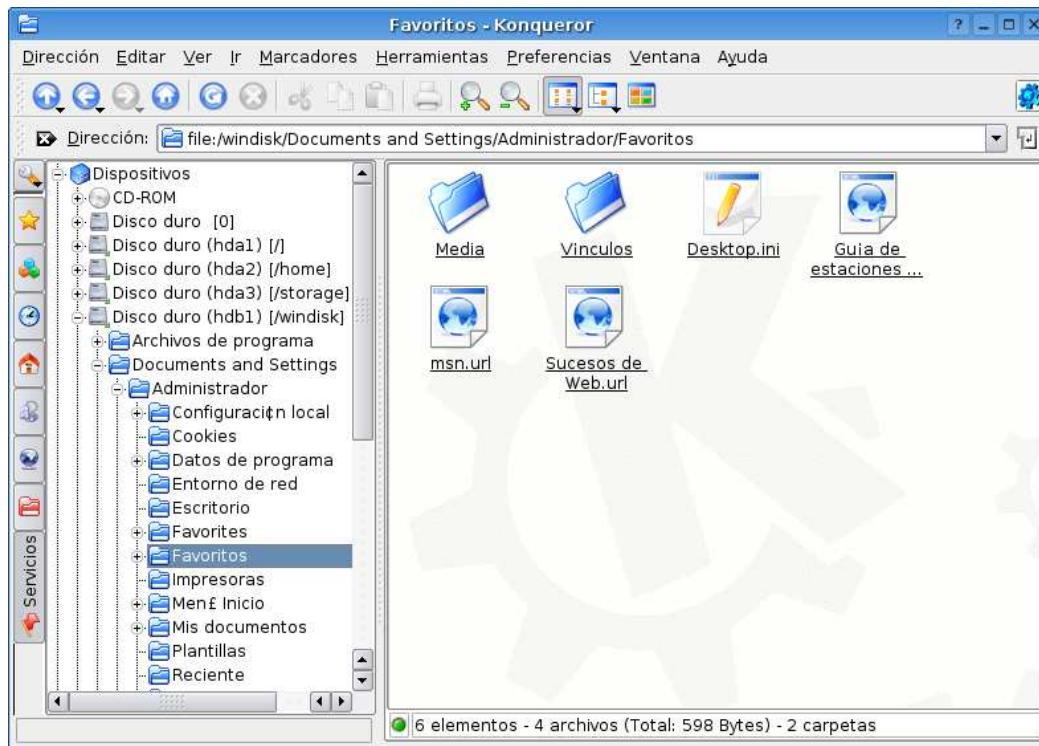


Figura 6.3: Konqueror como gestor de ficheros

La zona del árbol de directorios (llamada técnicamente “panel de navegación”) puede ocultarse mediante la opción “Mostrar/ocultar panel de navegación” en el menú “Ventana”, o pulsando F9³. Además de mostrar un árbol de directorios, el panel de navegación puede mostrar el historial de navegación, los enlaces favoritos, el directorio personal (“home”) del usuario, los dispositivos de disco en el sistema, o (configurándolo debidamente) un resumen de noticias. El cambiar la información mostrada en el panel de navegación se hace pulsando sobre los botones en el lado izquierdo.

Si queremos usar Konqueror como navegador web, sólo hay que introducir una URL, empezando por http:// en la barra de dirección.

²Esto viene del inglés “Home Directory” traducido como “Directorio Personal”, que es el directorio donde un usuario guarda sus documentos y configuraciones.

³Todas, absolutamente todas las combinaciones de teclas de KDE

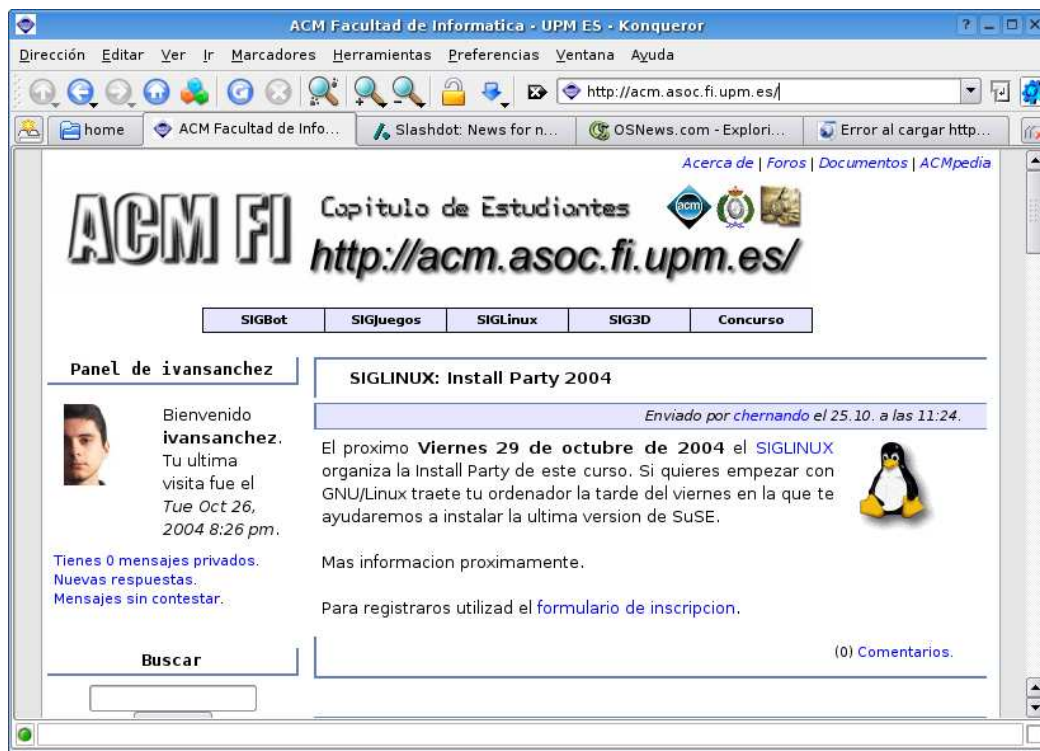


Figura 6.4: Konqueror como navegador web

Una funcionalidad muy útil es la navegación con pestañas (llamada por los ingleses “tabbed browsing”). Mediante el uso de pestañas se puede estar navegando por diversas direcciones desde una misma ventana. Con otros navegadores, al visitar varios sitios web se abren muchas ventanas, lo cual puede no ser fácilmente manejable. Al usar pestañas, la navegación por 10 o incluso 15 webs distintas se hace mucho más fácil.

La lista de pestañas se muestra justo debajo de la barra de herramientas (y no se muestra cuando sólo hay una pestaña). A su izquierda hay un botón para abrir una pestaña nueva (también `Ctrl+Shift+“N”`) y a la derecha otro para cerrar la pestaña actual (también `Ctrl+“W”`).

Además, se pueden usar las pestañas no sólo para navegar por sitios web: podemos tener unas pestañas como gestores de ficheros, y otras para navegar por webs, incluso otra con un visor de PDFs, todo dentro de la misma aplicación.

Los usos que se le pueden dar a Konqueror, dependiendo de la dirección introducida son, entre otros:

- **file:/** para usar Konqueror como gestor de ficheros.
- **http://** para navegar por sitios web.
- **ftp://** como cliente de FTP.
- **smb://** para ver particiones de ficheros e impresoras en máquinas Windows.
- **fish://** para gestión de ficheros en equipos remotos con acceso SSH.
- **audiocd:/** para escuchar y “ripear” CDs de audio.
- **devices:/** para ver los dispositivos de almacenamiento del equipo.
- **system:/** como `devices:/`, pero incluye impresoras, `audiocd:/`, y opciones de configuración.

Konqueror también puede hacer búsquedas en diversos sitios web. Por ejemplo, si ponemos `gg:facultad de informatica` como dirección, buscaremos “facultad de informática” en google. Si ponemos `rae:palabra`, buscaremos “palabra” en el diccionario de la Real Academia de la Lengua Española.⁴

Konqueror tiene muchas otras opciones y funcionalidades, pero si las viéramos todas, ésto no sería una “breve introducción”.

⁴La lista completa de buscadores a los que se puede acceder desde la barra de direcciones se puede ver en la configuración de Konqueror, en la sección “Accesos rápidos de web”

6.2.2. Correo con Kmail y Kontact

Kmail es el gestor de correo de KDE, similar a Evolution en Gnome, a Mozilla Thunderbird o a Outlook Express.

Ofrece las funcionalidades típicas de un gestor de correo: redactar, enviar y recibir correo, organizar correo en carpetas, filtros a la recepción de nuevo correo, etc.

Lo que puede diferenciar a Kmail con respecto de otros gestores de correo es la posibilidad de integrar encriptación GPG fácilmente, o la de filtrar todo el correo mediante spamassassin y bogofilter, dos filtros antispam tremendamente útiles⁵.

Una parte que suele confundir a los usuarios que usan Kmail por primera vez es la configuración de las cuentas de correo, que no es visible a primera vista. Hay que ir a la opción “Configurar Kmail” del menú “preferencias” y una vez ahí, a la sección de “Red”. Aquí hay dos pestañas: “Enviando”, donde hay que configurar el correo saliente (normalmente un servidor SMTP), y la pestaña “Recibiendo”, donde se configuran las cuentas de correo (normalmente mediante acceso a un servidor POP3).⁶

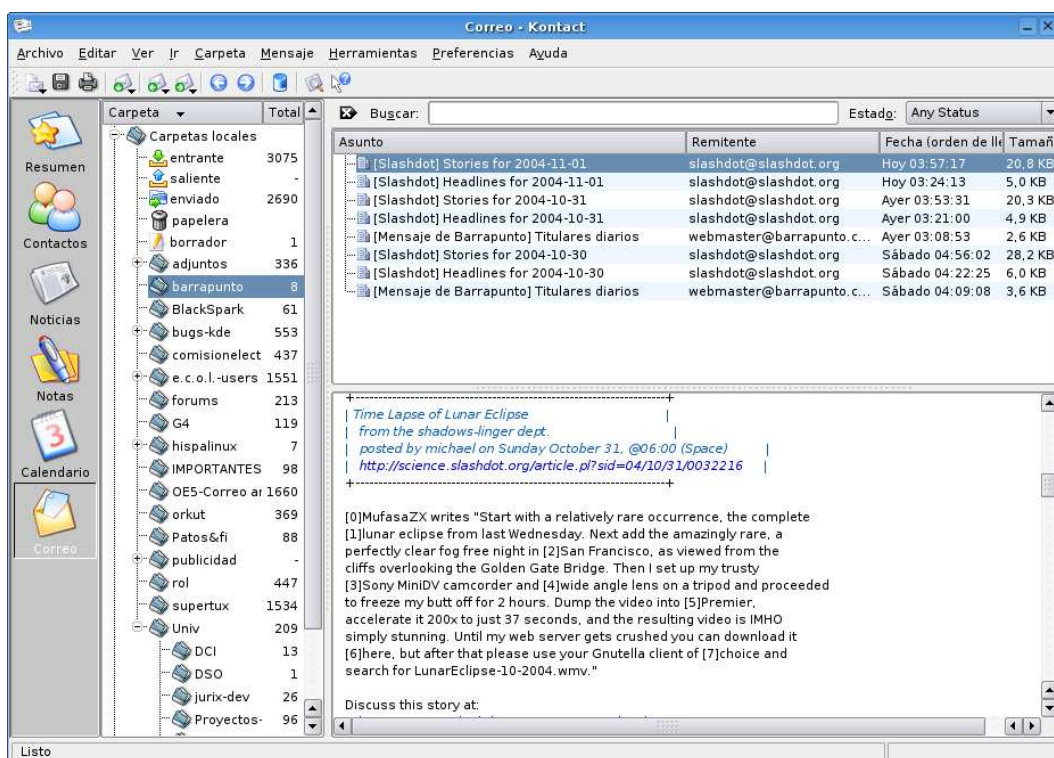


Figura 6.5: Kmail en plena acción, como parte de Kontact

En la figura superior, no se muestra Kmail, sino Kontact. Kontact es una “aplicación contenedora”, cuyo único objetivo es contener otras aplicaciones y, en el caso de Kontact, mostrar a la izquierda una barra con la que poder cambiar entre las aplicaciones “contenidas”. En cualquier caso, el usar Kmail por separado o desde Kontact no supone ninguna diferencia.

A Kontact se le denomina “PIM”⁷. Normalmente agrupa Kmail como gestor de correo, Kaddressbook como lista de contactos, Korganizer como calendario y agenda y un resumen (en la figura siguiente) que incluye información de las distintas aplicaciones, o incluso información meteorológica.

⁵La integración del filtrado anti-spam a través de “wizards” o ayudantes se incluyó en KDE 3.3.1

⁶Los expertos en usabilidad e interfaces de KDE siguen discutiendo sobre este tema.

⁷Personal Information Manager

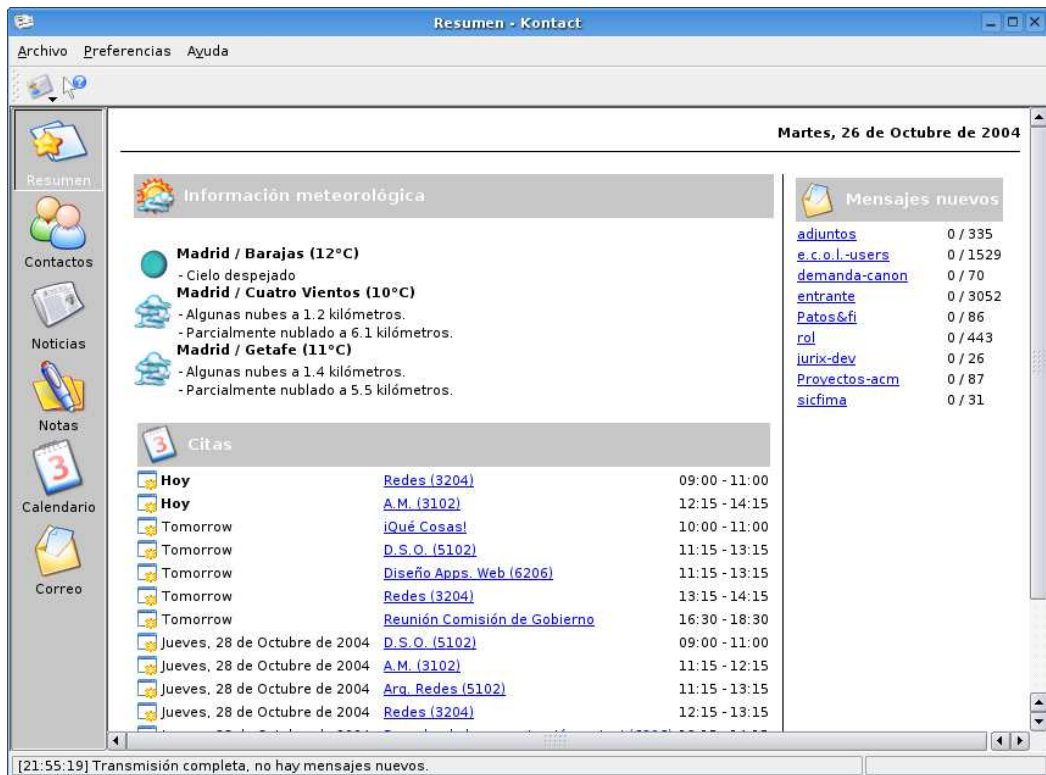


Figura 6.6: El resumen de Kcontact

Adicionalmente, puede integrar un lector de News (Knode), “notas amarillas” (Knotes), un agregador de noticias RSS (aKregator o Knewsticker) o un sincronizador para Palms (Kpilot), entre otros.

6.2.3. Escuchando música con JuK y amaroK

Hoy en día, muchos usuarios de ordenadores tenemos en MP3 u Ogg Vorbis⁸ todas las canciones de nuestros CDs de música. Existe una multitud de reproductores de MP3/Ogg para Linux (así como para otros sistemas operativos), pero en las últimas ediciones de KDE han añadido dos estupendas aplicaciones: JuK y amaroK.

JuK fue creado en el año 2000, y añadido a KDE en el 2002. Fue creado con el propósito principal de organizar listas de reproducción y grandes cantidades de canciones en MP3/Ogg. Lo mejor de esta aplicación es la capacidad de editar las etiquetas ID3 (y buscarlas en MusicBrainz⁹) de manera masiva y rápida. Genera listas por autor, álbum y género, y busca nuevas canciones en los directorios que se le configuren.

Una de las últimas funcionalidades de Juk es el poder cambiar de nombre (y, por consiguiente, mover) los mp3/ogg, muy útil cuando se tiene toda la música en un mismo directorio y se quiere organizar sin demasiado trabajo. Sin embargo, aunque JuK es muy potente en lo que a organización y gestión de las canciones se refiere, es algo pobre visualmente.

⁸Ogg Vorbis es una excelente alternativa libre al formato MP3. Véanse <http://es.wikipedia.org/wiki/Ogg-Vorbis> y <http://en.wikipedia.org/wiki/Ogg>.

⁹<http://www.musicbrainz.org> es una “base de datos de metainformación musical”.

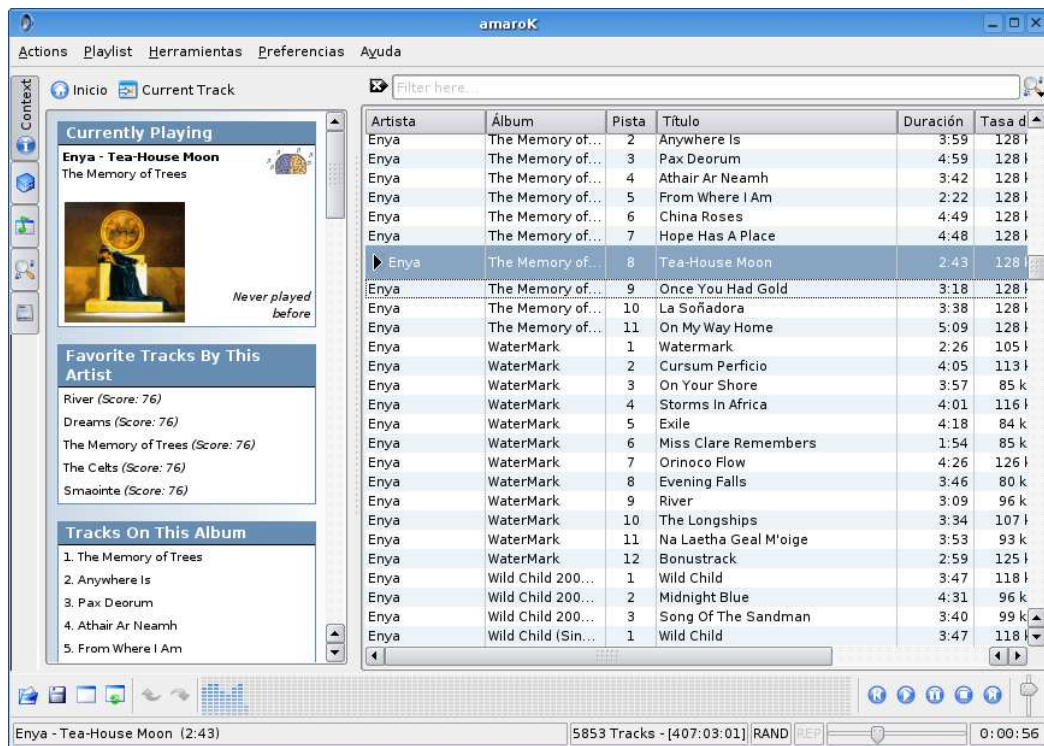


Figura 6.7: amaroK en acción

Como contrapartida, en el 2003-2004 se creó amaroK, un reproductor de música mucho más atractivo visualmente (aunque sin algunas de las funcionalidades más avanzadas de JuK). Incluye un panel lateral en el que se muestra la carátula del álbum que se está escuchando, pone puntuaciones a las canciones automáticamente (dependiendo de si se escucha hasta el final o se pasa a otra), y también es capaz de mostrar listas por autor, álbum y género.

Ambos programas preguntan, la primera vez que son lanzados, el directorio donde buscarán las canciones. A partir de ahí, generarán listas de las mismas y podremos escuchar una canción cualquiera pulsando sobre su entrada en la lista, o usando los controles (play/pause/next/prev). Para editar la metainformación de una canción (las “etiquetas ID3”), en JuK se usa un panel ocultable, y en amaroK se hace mediante el botón derecho del ratón, pulsando sobre una canción.

6.2.4. Mensajería instantánea con Kopete

Durante los últimos años se ha popularizado mucho la mensajería instantánea, pero sobre diversos protocolos: ICQ, MSN, Jabber, IRC, Y!M, AIM o Gadu-Gadu, entre otros. El problema con tanta variedad es que sólo puedes comunicarte con otra persona si usa el mismo protocolo (en otras palabras, el mismo programa de mensajería instantánea). Ante esto, nacieron diversos programas de mensajería multiprotocolo, entre ellos Trillian, Gaim y Kopete.

Kopete se puede usar como cliente “monoprotocolo”, por ejemplo, configurando sólo una cuenta para mensajería mediante MSN (que es la que usa el MSN Messenger). Pero si usamos más de un protocolo (mediante el uso de más de una cuenta de mensajería instantánea), tendremos un abanico más amplio de posibilidades de comunicación.

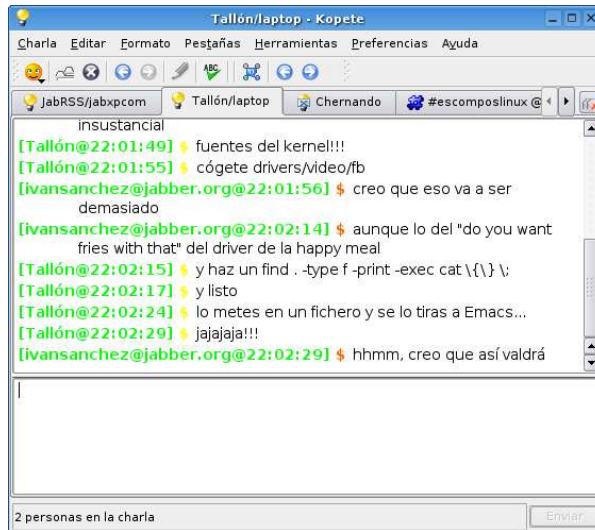


Figura 6.8: La ventana de charla de Kopete

El uso básico de Kopete es el siguiente: se configura una o varias cuentas en “Configurar Kopete” del menú “Preferencias”; se conecta/desconecta mediante los iconos de los enchufes (en la parte superior) o en los iconos de estado (en la parte inferior). Al conectar una cuenta, aparecerán en la ventana de contactos las personas con las que podemos hablar; pulsando sobre el nombre de una de ellas iniciaremos una conversación.

Algo que puede confundir a los nuevos usuarios de Kopete es que, en la ventana de charla, a la hora de enviar un mensaje a la otra persona, no se puede usar la tecla enter y hay que recurrir al botón de “enviar”. Esto es porque, por defecto, Kopete está configurado para enviar los mensajes con Shift+Enter. Como en el resto de KDE, ésta combinación de teclas se puede cambiar (a pulsar Enter únicamente, por ejemplo).



Figura 6.9: La ventana de contactos de Kopete

Una buena característica de Kopete que no tienen otros clientes multiprotocolo es el soporte para IRC, que ha sido mejorado desde las antiguas versiones de Kopete.

Pero la principal ventaja de usar Kopete es la posibilidad de agrupar contactos en un “metacontacto”. Es decir: si yo puedo hablar con una persona mediante MSN, ICQ e IRC, no tiene porqué aparecer por triplicado en mi lista de contactos. Para hacer esto, sólo hay que arrastrar el icono de estado (la bombillita, mariposilla o florecita) hasta otro contacto de la misma persona. De ahí en adelante, Kopete tiene en cuenta las diversas maneras de ponerse en contacto con esa persona, y mostrará a esa persona (el “metacontacto”) si se puede hablar con él a través de al menos uno de los protocolos.

Cualquier uso “no básico”, como unirse a salas de charla o canales en el IRC, o enviar ficheros a otras personas, se hace pulsando el botón derecho del ratón sobre los iconos de estado, o sobre un contacto.

6.2.5. Al tostadero con K3b

K3b es la aplicación por excelencia para “tostar” CDs. Aunque en realidad no sea K3b quien grabe el CD, sino otras aplicaciones (cdrecord, cdrao, mkisofs e growisofs, entre otras), proporciona una interfaz extremadamente fácil de usar. El único inconveniente quizás es que hay que pasar por una detección de las unidades de CD y otras configuraciones antes de usarlo por primera vez. Salvo rarísimas excepciones, K3b funciona (una vez configurado) con cualquier grabadora de CDs o DVDs.

Al lanzar K3b, nos encontraremos con una pantalla que nos permitirá elegir el tipo de CD (o DVD) que queremos grabar, como se muestra en la figura siguiente:

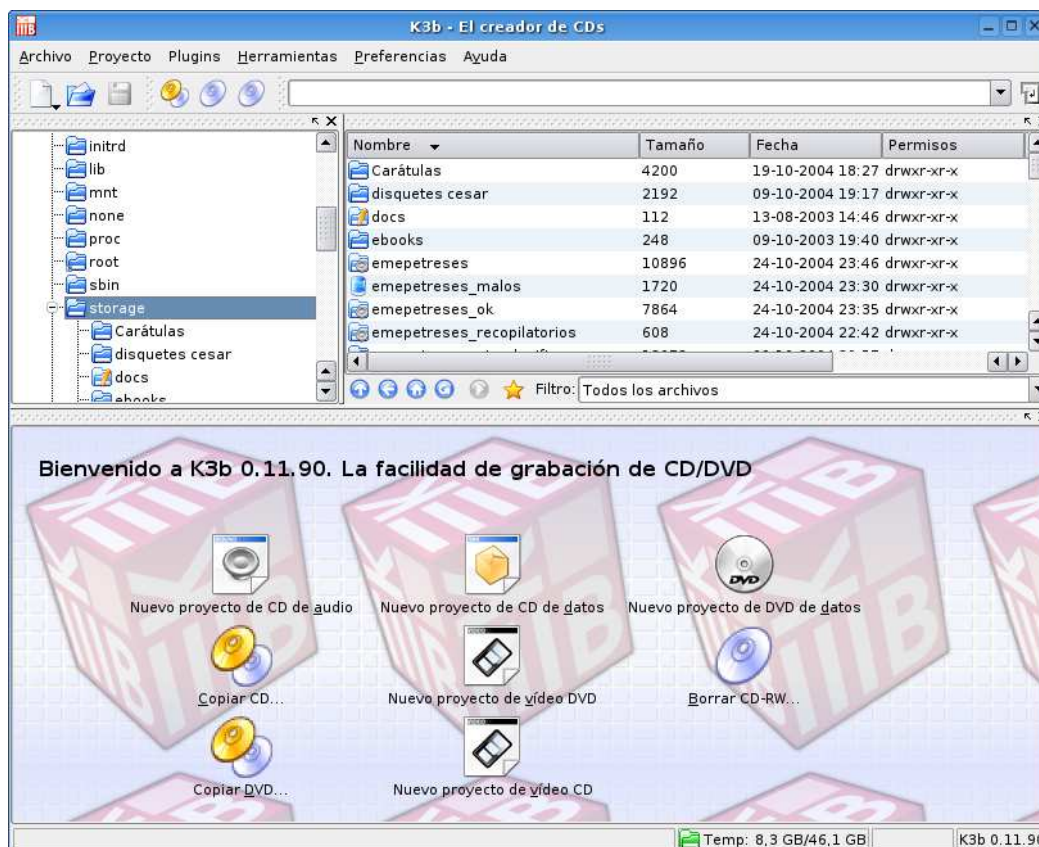


Figura 6.10: La ventana principal de K3b

Si, por ejemplo, pulsamos sobre “Nuevo proyecto de CD de datos” para grabar un CD con ciertos ficheros, tendremos en la parte inferior un lugar donde arrastrar los ficheros que queremos grabar (desde la parte superior de K3b o desde otra aplicación, como Konqueror). Si fuese un CD de audio, sólo podríamos arrastrar ficheros de música (normalmente .wav, .mp3 y .ogg). En cualquier caso, cuánto espacio libre queda en el CD antes de ponernos a grabar.

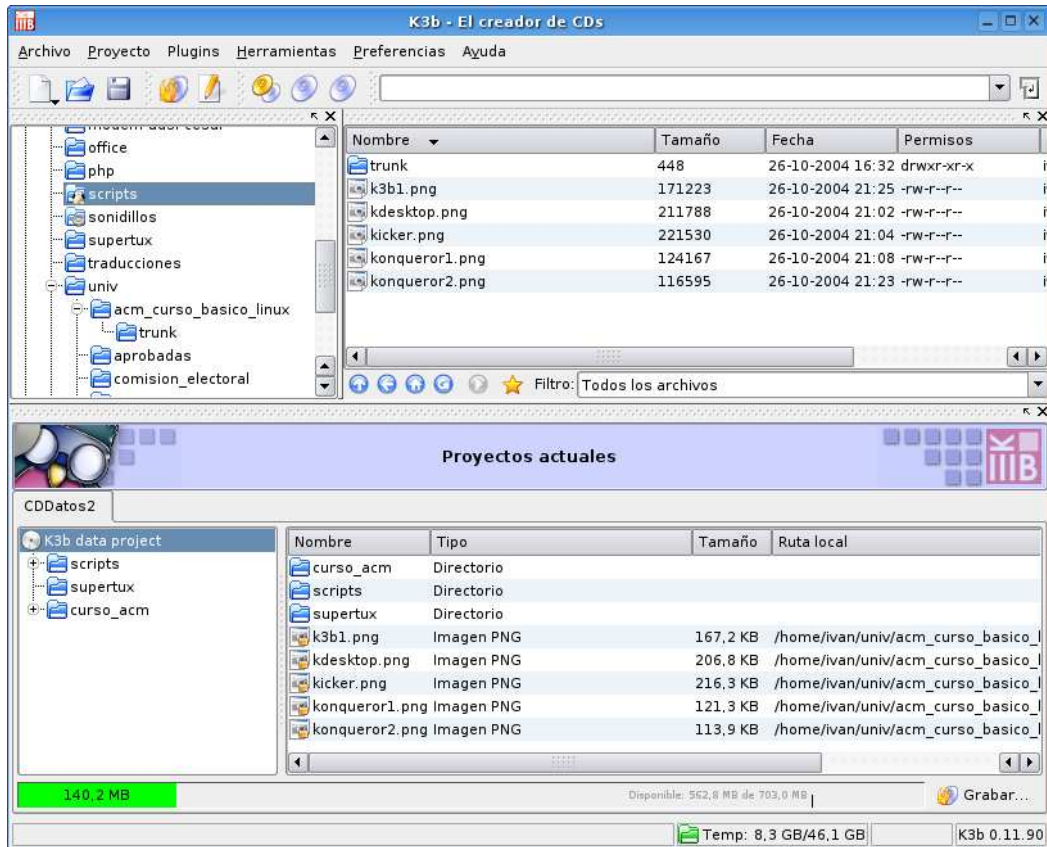


Figura 6.11: Añadiendo ficheros a un CD de datos

Cuando tengamos listo el contenido de nuestro CD, pulsamos sobre el botón de “grabar” que hay sobre la esquina inferior. Después seleccionamos la grabadora, la velocidad de grabación y algunas otras opciones (entre las que hay que activar el “Generar extensiones Joliet” si queremos leer bien los nombres largos de fichero en un ordenador con Windows). Tras aceptar esto, podremos ver el progreso de la grabación:

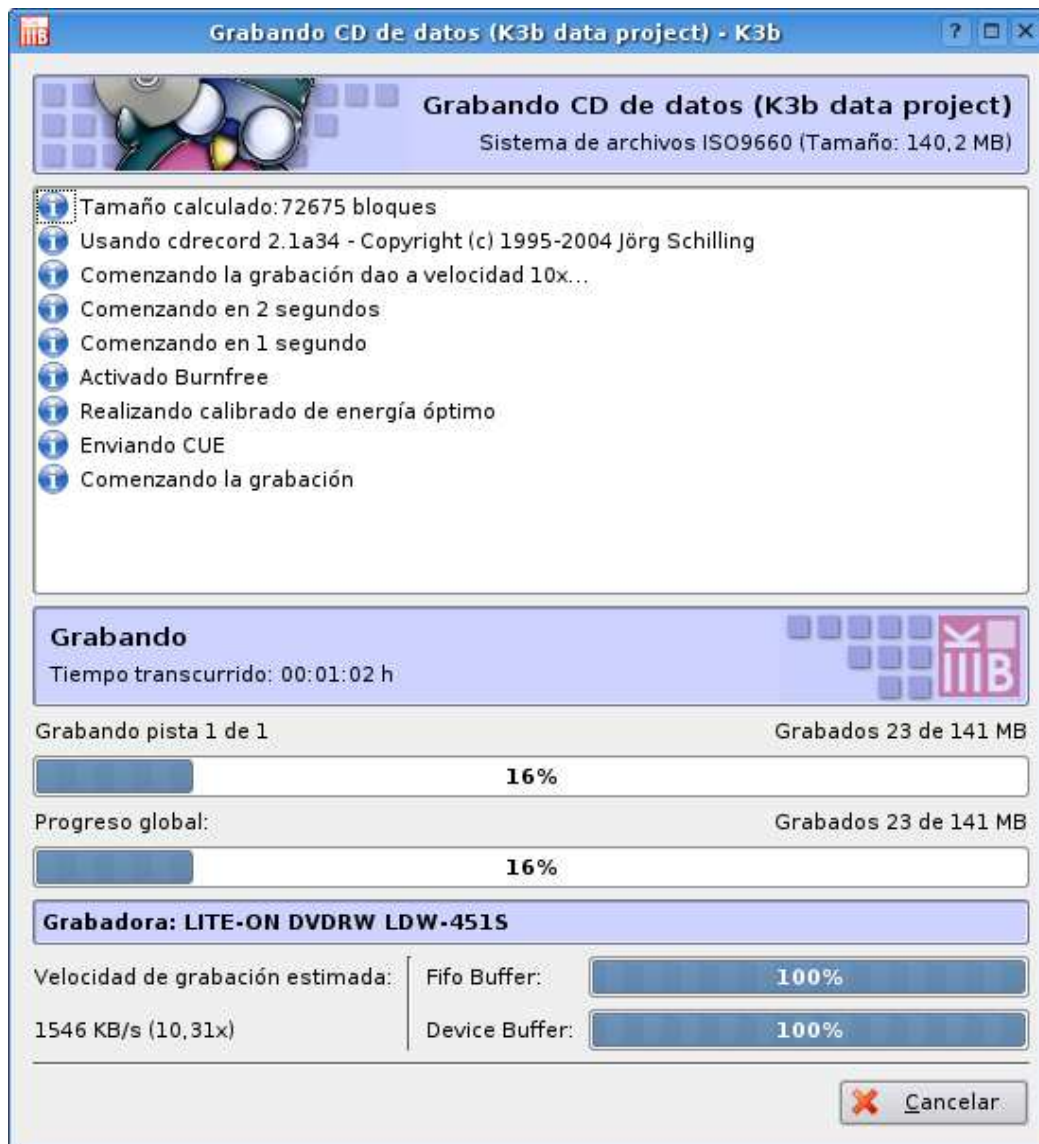


Figura 6.12: Grabando un CD

Si la grabación falla por alguna razón, es conveniente ver la “Salida de depurado”¹⁰, con lo que podremos ver qué es lo que han hecho los programas que realmente graban el CD (como hemos dicho antes, cdrecord y compañía). Recordemos que K3b es sólo una interfaz amigable sobre estos programas, que son los que no pueden grabar en última instancia.

Algunas opciones que no se ven a primera vista, como crear o grabar una ISO, o borrar o formatear un DVD+-RW, son accesibles desde el menú “Herramientas”.

6.2.6. Más allá de lo básico: Opciones y configuraciones

Algo que diferencia a KDE de otros entornos de escritorio es la cantidad de opciones de configuración que tiene, tanto en las aplicaciones más grandes como en el Centro de Control. Esto ha venido creando una cierta controversia, enfrentando por un lado a los “fanáticos de la personalización” contra los “puristas de la sencillez”. Pero si te gusta dejar las cosas a tu gusto o probar cosas nuevas, es probable que pases horas cambiando configuraciones de KDE.

Por una parte, hay que destacar que se puede cambiar todo el aspecto gráfico de las aplicaciones de KDE, mediante las opciones de “Aspecto y temas” del Centro de Control de KDE. Por ejemplo, podemos poner las decoraciones de ventana “Keramic-Based Luna”, junto con estilo, colores e iconos “kde-xp” y tendremos un escritorio y unas aplicaciones con una apariencia totalmente distinta:

¹⁰Del inglés “debug output”

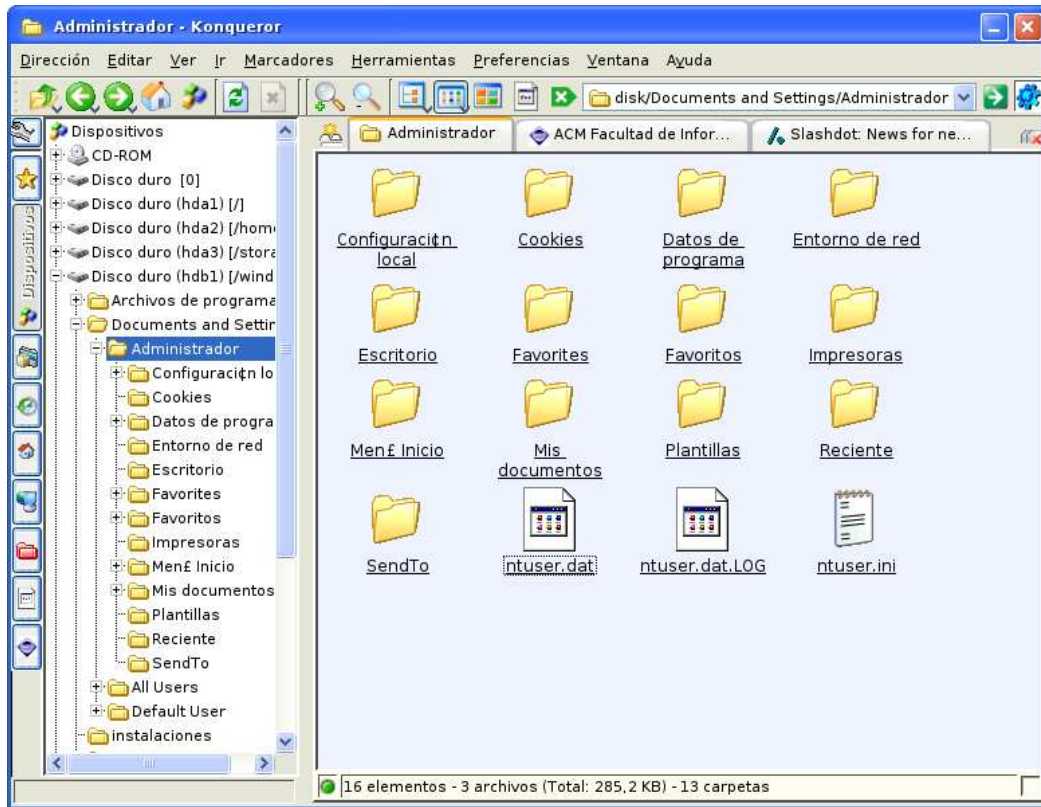


Figura 6.13: Konqueror con aspecto KDE-XP

Otras opciones interesantes son las de las combinaciones de teclas. Por ejemplo, entre otra multitud de opciones, podemos cambiar la manera en la que cerramos una ventana a base de combinaciones de teclas:

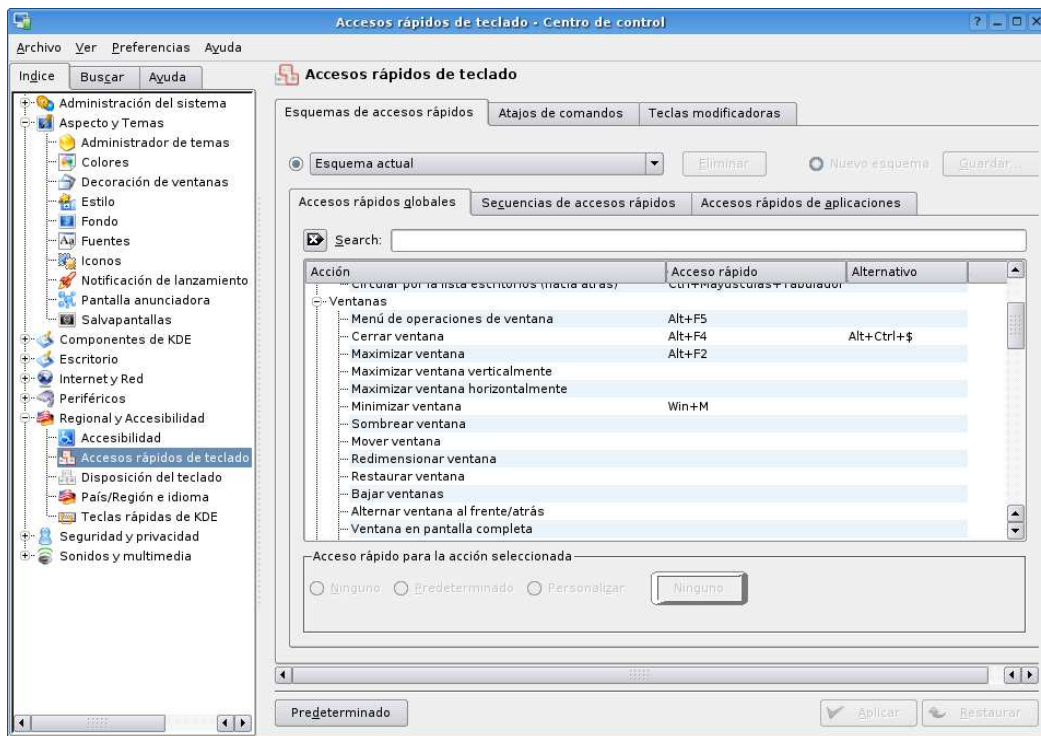


Figura 6.14: Cambiando las combinaciones de teclas

Aparte de las configuraciones globales en el Centro de Control de KDE, están las configuraciones de cada aplicación. Normalmente, una aplicación de KDE tiene (al menos) tres opciones de configuración dentro de su menú “Herramientas” o “Preferencias”: Configuración de las combinaciones de teclado de esa aplicación en concreto; configuración de la barra de herramientas de la aplicación, y configuración general de la aplicación.

En definitiva: no hay mejor manera de ver qué se puede hacer con KDE es experimentar. Experimentar o bien con aplicaciones que nunca has probado, o bien viendo y cambiando las configuraciones de una aplicación en concreto.

6.3. OpenOffice.org

En esta parte vamos a introducir, de una forma muy breve, el manejo del paquete OpenOffice. Una explicación detallada ocuparía un libro entero.

6.3.1. Introducción

OpenOffice es la alternativa libre y multiplataforma a las suites ofimáticas propietarias, en particular al Office de Microsoft. Tiene como gran virtud una compatibilidad muy buena con los documentos generados por esta suite. Por tanto podremos seguir abriendo los documentos que tuvieramos anteriormente así como generar otros nuevos para gente que sólo tenga la suite de Microsoft.

A las ventajas propias de ser un proyecto de software abierto (precio, acceso al código...) se le añaden otras propias de este programa. A diferencia del Office de Microsoft, OpenOffice es multiplataforma, existiendo versiones para, entre otros, Windows y GNU/Linux. Esto nos permite por ejemplo aprender a usarlo en Windows, de manera que al pasarnos a GNU/Linux podamos seguir usándolo de la misma manera.

Por otra parte, a diferencia de otras suites ofimáticas, OpenOffice utiliza formatos de archivos libres (en concreto utiliza XML comprimido con zip). Esto nos dá la seguridad de que siempre podremos acceder a la información. En último recurso podemos leer a mano el fichero XML. Por esto tampoco hay peligro de que el fichero se corrompa, cosa que pasa a veces con otras suites, lo que nos hace perder todo el documento.

Existe, para quien la quiera comprar, una versión de pago, “StarOffice” que comparte funcionalidad, pero añade cosas como plantillas y cliparts predefinidos.

Podemos definir OpenOffice como un conjunto de herramientas para la oficina (suite ofimática) que consta de 5 aplicaciones:

- OpenOffice.org Writer. Herramienta para la edición de textos
- OpenOffice.org Calc. Realización de hojas de cálculo
- OpenOffice.org Impress. Presentaciones y diapositivas
- OpenOffice.org Draw. Diagramas, dibujos y gráficos
- OpenOffice.org Math. Edición de expresiones matemáticas

Si nos fijamos no existe un equivalente al Access propiamente dicho, lo que ocurre es que la funcionalidad de este está incluida dentro de las otras aplicaciones.

Nosotros nos vamos a centrar en el Writer, por ser el más usado.

6.3.2. Introducción a Writer (procesador de textos de OpenOffice).

Cuando arranquemos el programa lo que vamos a encontrar es un entorno que, aunque sea la primera vez que se visite, se nos hará bastante familiar si hemos manejado un procesador de textos en alguna ocasión. Nos recordará mucho a otros programas similares (dígase MS Word, en cualquiera de sus versiones). No obstante, es aconsejable hacer uso de la Ayuda que el programa posee y que aparece en el menú principal si tenemos dudas.

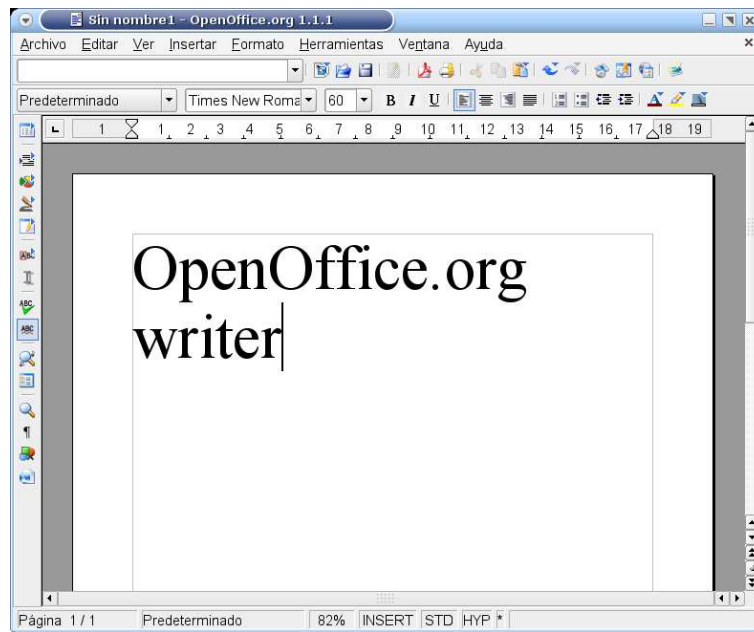


Figura 6.15: Open Office Writer

Por una parte tenemos el “estilista” es una ventana que nos sirve para aplicar formatos al texto.

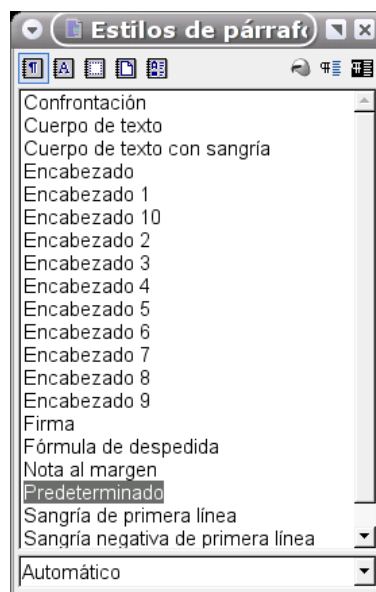


Figura 6.16: Open Office Estilo

La idea es que usemos unos estilos aplicandolos a cada parte del texto. Por ejemplo, si queremos un titulo de apartado, tenemos que escribir el titulo y aplicarle el formato de queramos. Por ejemplo el formato “encabezado1”, aunque no nos guste demasiado. Si hacemos esto con todos los apartados, luego podemos modificar el formato en el estilista, y automáticamente todos los apartados veran su aspecto modificado. Mucho mejor que andar seleccionando fuente, tamaño y alineación para cada encabezado.

Otra cosa que diferencia a Writer de Ms Word es el navegador, que nos permite Navegar por el documento.

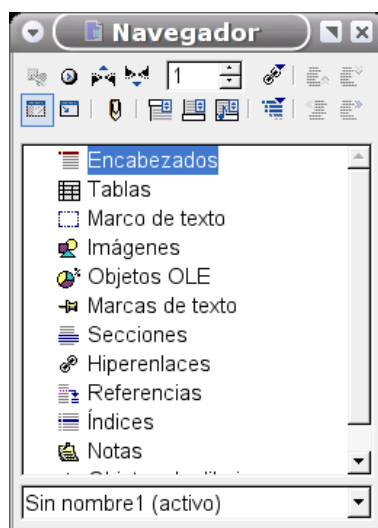


Figura 6.17: Open Office Navegador

Con el podemos ir directamente a una tabla, una seccion, una imagen... Esto hace que escribir textos grandes sea muchísimo mas cómodo.