Guía de aprendizaje para estudiantes

Curso de Introdución a Linux para Alumnos

Guía de aprendizaje para estudiantes

Curso de Introdución a Linux para Alumnos

Miguel Ángel Vilela García
Jesús Miguel Torres Jorge
Carlos Pérez Pérez
Tomás Bautista Delgado
Carlos Alberto Morales Díaz
Félix J. Marcelo Wirnitzer
Sergio García Reus
Carlos de la Cruz Pinto
René Martín Rodríguez
Edín Kozo
Carlos Mestre González

Copyright © 2001 – 2003 Grupo de Usuarios de Linux de Canarias

Se da permiso para copiar, distribuir o modificar este documento en los términos que establece la GNU Free Documentation License, Versión 1.1 o cualquier otra versión posterior publicada por la Free Software Foundation; las secciones que no pueden modificarse son "Presentación" y "Agradecimientos", y no existen textos cubierta ni contracubierta. Se incluye una copia de la licencia en la sección "GNU Free Documentation License". Este libro es © de sus autores.

La forma original de este documento es código fuente en LaTeX. La compilación de este código fuente en LaTeX tiene el efecto de generar una representación del documento independiente del dispositivo, que puede convertirse a otros formatos o imprimirse.

La GNU Free Documentation License está disponible en www.gnu.org o soliciándola por escrito a the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

Este documento lo han maquetado sus autores utilizando LaTeX y The GIMP, que son programas abiertos y libres.

A los novatos y novatas en el mundo de GNU/Linux,
y a sus familiares, amistades y parejas
(ellas y ellos saben por qué)

Presentación

Historia

Desde el curso académico 2001–2002 la Facultad de Matemáticas de la Universidad de La Laguna dispone de sistemas Debian GNU/Linux en las aulas de ordenadores destinadas a impartir clases prácticas. Estas aulas están destinadas a que los alumnos realicen sus prácticas académicas, tales como programación en Pascal, Fortran, C o Java, análisis estadísticos o cálculos simbólicos. Sin embargo, los alumnos no disponían en su mayoría de los conocimientos mínimos necesarios para utilizar un equipo bajo sistema operativo GNU/Linux.

Ante tal panorama, y con el objetivo urgente de proporcionar a los alumnos la destreza mínima que necesitaban para utilizar los ordenadores de la facultad con Debian GNU/Linux, Miguel Ángel Vilela (miembro del Grupo de Usuarios de Linux de Canarias y administrador de los sistemas Debian GNU/Linux de las aulas) propuso paralelamente la realización de un **Curso de Introdución a Linux para Alumnos** a Sergio Alonso (Vicedecano de Estadística) y a otros miembros del el Grupo de Usuarios de Linux de Canarias. En ambas partes la idea tuvo buena aceptación y en sólo 10 días el Grupo de Usuarios de Linux de Canarias y la Facultad de Matemáticas, con el patrocinio de IDE System Canarias S.L., organizaron la primera edición del Curso de Introdución a Linux para Alumnos.

Tras el éxito del Curso de Introdución a Linux para Alumnos (30 inscritos, 27 asistentes y más de 50 alumnos en lista de espera) se organizó la primera edición de la **Party de Instalación de Linux para Alumnos**, a la que no asistieron tantos alumnos debido (suponemos) a que la asistencia implicaba traer el ordenador personal. Sin embargo, el balance final de ambas actividades se consideró muy positivo, por lo que el Grupo de Usuarios de Linux de Canarias se embarcó en el siguente reto: ampliar el Curso de Introdución a Linux para Alumnos para aumentar el temario (incluyendo la Party de Instalación de Linux para Alumnos).

El Curso de Introdución a Linux para Alumnos se ha realizado ya dos años consecutivos en la Facultad de Matemáticas y para los próximos años pretendemos seguir realizándolo, ampliando su radio de acción a otras facultades de la Universidad de La Laguna. Este libro es una puerta abierta para que el Curso de Introdución a Linux para Alumnossea realizado en otras universidades y centros docentes.

Parte de este proyecto ha sido reformar los apuntes del Curso de Introdución a Linux para Alumnos, inicialmente escritos en SGML (con DTD de LinuxDoc), para traducirlos a LaTeX y así generar este libro, que constituye los apuntes del curso. Este libro es una publicación de "contenido abierto", lo que significa que está abierto a las aportaciones que pueda hacer cualquier persona y seguirá evolucionando con el tiempo.

Objetivos

El objetivo del Curso de Introdución a Linux para Alumnos y de este libro es el aprendizaje del alumno (o lector), a ser posible de una forma amena. En adelante trataremos de tú tanto al alumno como al lector. Esperamos que no te sientas ofendido.

Este libro está dirigido principalmente a usuarios, con un enfoque fuertemente académico. Si hace poco que llegaste al maravilloso mundo de GNU/Linux y necesitas una guía práctica para empezar a trabajar, este libro puede resultarte de utilidad. Si además eres estudiante y necesitas hacer tus prácticas en ordenadores con GNU/Linux pero no te orientas aún, este libro puede ser tu salvación. Si ya tienes experiencia con GNU/Linux (no eres un novato) no encontrarás muchas cosas nuevas aquí, aunque tampoco te vendrá mal leerlo.

La finalidad del Curso de Introdución a Linux para Alumnos es que los alumnos adquieran los conocimientos necesarios para utilizar los ordenadores con GNU/Linux en sus prácticas académicas, y tal vez se animen a instalar GNU/Linux en sus propios ordenadores personales. Como se ha mencionado anteriormente, el Curso de Introdución a Linux para Alumnos consta de unas clases teórico-prácticas, en las que los alumnos toman apuntes mientras prueban lo que van aprendiendo con un ordenador delante, y unas jornadas práctico-teóricas en la que cada alumno es invitado a traer su propio ordenador personal para instalar GNU/Linux por sí mismo, contando siempre con la ayuda de los profesores.

Dada la posible variedad de los alumnos, la cantidad de materia que se desea impartir y las limitaciones de disponibilidad de puestos en las aulas de informática en cuanto a número y tiempo, el Curso de Introdución a Linux para Alumnos se ha dividido en varios módulos, todos ellos opcionales.

Las tareas rutinarias de administración y mantenimiento de un equipo bajo sistema operativo GNU/Linux no se tratan en este libro, entre otras razones porque suelen depender mucho de la máquina en la que se practiquen y de las necesitades de su(s) usuario(s). Esta parte del aprendizaje de GNU/Linux es demasiado práctica para un libro como éste. Para esto es mejor comprar algún libro de iniciación a Linux que incluya una distribución de GNU/Linux para instalar. Este tipo de libros se pueden encontrar en casi cualquier librería técnica en variedad para todos los gustos, desde pequeñas guías de 10 minutos con CD de instalación rápida hasta libros gruesos de sabiduría condensada, elige el que más te guste y reserva grandes ratos en tu agenda.

Uno de los motivos por los que publicamos este libro bajo la licencia GNU Free Documentation Licence es porque queremos que tenga toda la divulgación que sus lectores quieran. Este libro no es copyright de ninguna editorial, sólo de los autores, y damos nuestro permiso para que este material sea fotocopiado y redistribuido libremente. Nuestro objetivo es acercar GNU/Linux y sus herramientas a cuantos más alumnos y usuarios sea posible.

Contenidos

Este libro está dividido en módulos atendiendo a la división de contenidos que hacemos a la hora de impartir las clases, los cuales a su vez están divididos en temas, cuyos contenidos resumimos a continuación.

TEMA 1: Introducción a GNU/Linux
Cuando hablamos de GNU/Linux nos referimos a algo que ha sido formado por un sistema operativo llamado Linux y una gran cantidad de software libre proviniente del proyecto GNU. Daremos un repaso histórico desde los orígenes de Linux y GNU hasta la actualidad.

TEMA 2: El intérprete de comandos
El primer mejor entorno de trabajo que puedas encuentrar al entrar en un sistema GNU/Linux es el intérprete de comandos. Aunque cada día es más frecuente entrar a Linux por las ventanas, no debemos dejar de lado la enorme libertad que nos da un intérprete de comandos. Y, aunque no lo parezca, es un entorno sumamente cómodo.

TEMA 3: El entorno X-Window
El entorno favorito de casi todos los usuarios de computadores es el entorno de ventanas, en el que el ratón y el teclado se suman para dar al usuario libertad y comodidad. Verás que existe una amplia variedad de entornos de ventanas, y aprenderás trucos útiles para cada uno de ellos.

TEMA 4: Editores: VIM, GNU Emacs, Joe
Una de las tareas más frecuentes que tendrás que hacer frente a un ordenador es escribir en un fichero. Programas, documentos, correos electrónicos, páginas web, tal vez configuraciones de programas. Este tema te enseñará el manejo básico de los editores de texto más usuales en GNU/Linux.

TEMA 5: Aplicaciones para Internet
Hoy en día un ordenador sin conexión a Internet es algo tristemente aislado. En el caso de GNU/Linux esto se hace más notorio porque es un sistema que ha nacido y crecido a través de Internet, se puede decir que está hecho en y para Internet. Un uso básico de las herramientas para Internet te resultará imprescindible para seguir adelante en tu aprendizaje sobre GNU/Linux, ya que casi todas las respuestas están en Internet.

TEMA 6: Documentación y ayuda

Cuando te sientas perdido con GNU/Linux y sus herramientas tendrás a tu disposición una amplia ayuda y documentación, aprender a utilizarla es la clave del éxito como usuario del software libre.

TEMA 7: Instalación de GNU/Linux

¿Te animas a instalar GNU/Linux en tu ordenador? ¡Felicidades! Ahora bien, antes de atacar la tarea hay unas cuantas cosas que debes saber para que te salga bien.

TEMA 8: Administración básica

Cuando por fín te animes a instalar GNU/Linux en tu propio ordenador tendrás que enfrentarte con la tarea más dura: administrar tu propio sistema, para mantenerlo funcionando. Hay muchos libros dedicados exclusivamente a esta materia, así que en este tema sólo aprenderás lo fundamental para mantener un ordenador personal sin grandes pretensiones. Para aprender más necesitarás dedicarle tiempo y leer bastante.

TEMA 9: Bash

Cuando quieras automatizar algo sencillo puede que sea suficiente un script, que no tendrás que compilar. Bash es el intérprete de comandos más extendido en GNU/Linux, y tiene su propio lenguaje de programación.

TEMA 10: StarOffice

Si piensas que no podrás sobrevivir en GNU/Linux porque no tiene Word o WordPerfect estás equivocado. StarOffice es una suite ofimática de calidad, potente, que responderás a tus necesidades. Además, como es libre, está disponible en varias plataformas además de GNU/Linux.

TEMA 11: El lenguaje HTML

Todos los ficheros que aprenderás a manejar en este curso están en formatos libres, de modo que cualquier otra persona puede utilizarlos (al menos leerlos) sin necesidad de comprar ni piratear programas especiales. Con el aprendizaje del lenguaje HTML estarás en condiciones de escribir documentos que cualquier persona podrá leer directamente en la web con su navegador.

TEMA 12: LyX

Para escribir rápidamente y que quede bien presentado nada mejor que concentrarte en el contenido de lo que escribes y dejar al ordenador que se encarge del trabajo de formato y composición del documento. Lyx está hecho exactamente para eso.

TEMA 13: LaTeX

TeXy LaTeX son los lenguajes más potentes que encontrarás para componer documentos de calidad profesional, aunque como era de esperar requiere un cierto esfuerzo de aprendizaje. Sin duda vale la pena, valga como ejemplo el este libro escrito íntegramente en LaTeX.

TEMA 14: GNU Octave

Para casi cualquier científico se hace muy necesaria la computación rápida de cálculos matemáticos aplicados mediante un lenguaje que permita programar los cálculos. Octave te dará la solución para los cálculos que necesites para física, ingeniería, matemáticas y otras disciplinas.

TEMA 15: GNUplot

La elaboración de gráficas elaboradas es otra tarea fundamental en el campo de las ciencias, y GNUplot tiene amplias posibilidades para esto.

TEMA 16: GNU R

Otro lenguaje de cálculo y representación de datos, pero esta vez con capacidades especiales para cálculos estadísticos.

TEMA 17: Yacas

Este lenguaje es diferente, para cálculo simbólico. Esto es especialmente útil en Matemáticas, donde a veces es necesario manipular las expresiones matemáticas sin evaluarlas.

TEMA 18: Free Pascal

Pascal es un lenguaje de programación ampliamente utilizado en las asignaturas de programación de primer año en ingenierías y otras carretas técnicas como Matemáticas y Física.

TEMA 19: GNU Fortran

Fortran es un lenguaje de programación específico para cálculo numérico. Aún se utiliza en entornos científicos como centros de cálculo o facultades de Matemáticas y Física.

TEMA 20: GNU C/C++
El lenguaje C es sin duda "El" lenguaje de programación por excelencia. GNU/Linux se ha construido mayoritariamente sobre este lenguaje, por lo que conviene saber manejar las herramientas de programación para programar en C. Además es el lenguaje de programación más ampliamente utilizado en prácticas académicas de programación.

TEMA 21: GNU Make
Cuando un proyecto crece, sea del tipo que sea, siempre viene bien una herramienta que automatice las labores de compilación del código. Esto se aplica no sólo a programas, sino también a libros como éste.

TEMA 22: Depuradores
La mayor parte del tiempo que dedicamos a programar lo pasamos buscando y corrigiendo errores en el código. Esta tarea te resultará mucho menos ingrata cuando conozcas el estupendo depurador DDD.

TEMA 23: GNU gprof
Una herramienta muy útil a la hora de optimizar tu código puede ser el profiler de GNU, que te ayudará a encontrar cuellos de botella y otros puntos débiles de tus programas.

TEMA 24: Java
El lenguaje Java es ampliamente utilizado como lenguaje de programación multiplataforma. Es útil para aplicaciones y applets en documentos HTML.

TEMA 25: Expresiones regulares
Una de las mejoras ayudas que encontrarás a la hora de programar son las expresiones regulares. Éstas te permitirán filtrar texto para editar tus programas de forma más eficiente, o convertir ficheros con información mal organizada en tablas de datos de información bien organizada.

Convenciones tipográficas

En la escritura de este libro hemos utilizado algunas convenciones para facilitar su lectura. Usaremos los diferentes tipos de letra para fines determinados:

- sin adornos: nombres de programas, paquetes, protocolos, lenguajes, etc. También para menús y botones.
- `máquina de escribir`: comandos, código fuente, pulsasiones y/o combinaciones de teclas; todo aquello que haya que teclear o leer de la pantalla.
- *itálica*: términos nuevos o que haya que resaltar.
- **negrita**: elementos de una descripción o aquello que necesite especial atención.
- VERSALITA: para nombres de personas, organizaciones y entidades en general.

Cuando hagamos referencia a combinaciones de teclas utilizaremos la siguiente notación:

- "`C-`" indica la tecla `Control`.
- "`S-`" indica la tecla `Shift`.
- "`A-`" indica la tecla `Alt` (a veces llamada `Meta`).
- "`F1`" ... "`F1`" son las teclas de función.
- "`ESC`" indica la tecla de Escape.
- "`TAB`" indica la tecla del tabulardor.
- "`Enter`" indica la tecla del retorno de carro.

Además, cuando tengamos que mostrar comandos y/o la salida que éstos produzcan lo haremos de la siguiente forma:

```
$ comando
salida
del
comando
```

El símbolo del dólar `$` es lo que comúnmente se denomina el *prompt* del sistema, lo que el intérprete muestra a la espera de nuestras órdenes. En las distribuciones de GNU/Linux actuales no es normal que este prompt sea únicamente el símbolo del dólar, sino que suele contener también el nombre de la máquina, el nombre del usuario, o el directorio en el que nos encontramos. Estos detalles serán explicados en el tema "El intérprete de comandos".

En los temas referentes a lenguajes de programación, documentación o cálculo incluiremos algunos ejemplos de código fuente de programas, scripts o documentos. Estos ejemplos están en el directorio `ejemplos` del paquete que contiene el código fuente de este libro, cuya localización en internet te facilitamos en la siguiente sección. Estos ficheros se utilizan en las clase teórico/prácticas para aprender el manejo de compiladores y herramientas de cálculo o documentación.

Fichero `ejemplo.txt`

```
Este fichero contiene el ejemplo de ejemplo

Su contenido se muestra tal como es el fichero original,
    incluyendo ta    bu    la    do    res
    y
    saltos
    de
    línea
```

Ejemplo 0.1: Éste es un ejemplo de cómo te mostraremos los ejemplos de código fuente. Éstos pueden ser programas, scripts o documentos escritos en diferentes lenguajes como C/C++, Fortran, Pascal, HTML, PHP, LaTeX, Octave, GNUplot, R, Yacas, etc. Los ejemplos que aparezcan de este modo estarán disponibles por separado en sendos ficheros cuyo nombre se indica en la cabecera del ejemplo. Así te ahorras tener que teclear los ejemplos a la hora de probar los lenguajes. Cuando necesites buscar uno de estos ejemplos puedes encontrar su página en el índice de ejemplos de la página XXVII.

Sobre este documento

Este libro ha sido escrito gracias a la iniciativa de algunos miembros y amigos del Grupo de Usuarios de Linux de Canarias, para el Curso de Introdución a Linux para Alumnos organizado conjuntamente por el Grupo de Usuarios de Linux de Canarias y la Facultad de Matemáticas de la Universidad de La Laguna, en Tenerife (España). Mira en la página XIII para saber quiénes participaron en la elaboración de este libro.

Como indica la nota de copyright, estos apuntes son libremente distribuibles bajo los términos de la Licencia de Documentación Libre de GNU, lo que a grosso modo significa que se permite su copia, redistribución y modificación siempre que se nombre a los autores originales, no se reclame la autoría de trabajo ajeno ni se pierda la libertad de los contenidos.

En el apéndice de la página 333 encontrarás una traducción **no oficial** de la Licencia de Documentación Libre GNU. Para leer la versión original de la GNU Free Documentation License acude a la página oficial del proyecto GNU, `http://www.gnu.org/licenses/fdl.html`

La última versión de estos apuntes se mantendrá en `http://cila.gulic.org`, tanto su código fuente como las versiones en PostScript y PDF, además de un paquete con los ejemplos por separado.

Este libro es un proyecto abierto y en desarrollo, por eso no está completo y posiblemente tarde en estarlo (siempre hay algo que añadir, mejorar, corregir, actualizar, etc.). Si quieres colaborar en él ponte en contacto con nosostros enviando un correo electrónico a la la dirección `cila@listas.gulic.org`. Nos gustaría recibir tus opiniones, correcciones y sugerencias críticas constructivas. Las críticas destructivas serán redireccionadas, como siempre, a `/dev/null`

Agradecimientos

Este libro lo hemos escrito miembros y amigos del Grupo de Usuarios de Linux de Canarias; informáticos, estudiantes y/o aficionados con entrega, entre todos unimos nuestros esfuerzos para escribir sobre aquello de lo que mejor entendemos.

La Facultad de Matemáticas de la Universidad de La Laguna ha colaborado apasionadamente (y lo sigue haciendo) aportando sus aulas de ordenadores para realizar las sucesivas ediciones del Curso de Introdución a Linux para Alumnos . Desde el Grupo de Usuarios de Linux de Canarias agradecemos además la amabilidad con la que han colaborado en todo momento.

Pero este libro no habría llegado a ser lo que es de no ser por las siguientes personas. A todos ellos; muchas gracias.

- Sergio Alonso, Vicedecano de Estadística en la Facultad de Matemáticasde la Universidad de La Laguna. Acogió amablemente la propuesta del Curso de Introdución a Linux para Alumnosy puso las aulas de ordenadores a disposición del curso. También ha coordinado la publicación de este libro a través del Servicio de Publicaciones de la Universidad de La Laguna.
- Manolo García Román, Secretario de la Facultad de Matemáticasde la Universidad de La Laguna. Brindó una inestimable ayuda en las cuestiones TEXnicas de la elaboración de estos apuntes. También se entregó de modo ejemplar en la organización de las distintas edicioens del curso en el ámbito de la Universidad de La Laguna. Ha coordinado la publicación de este libro a través del Servicio de Publicaciones Universidad de La Laguna.
- Miguel Ángel Vilela García escribió los temas "The GIMP", "GNU Fortran", "Yacas", "R", además de partes de "Aplicaciones para Internet", "Editores" y "Aplicaciones diversas". También retocó la traducción del tutorial de LyX aportada por Sergio García Reus para formar el tema "LyX".
- Jesús Miguel Torres Jorge escribió los temas "El entorno X-Window", "C/C++", "GNU Make", "Bash" y parte de "Aplicaciones para Internet". Posteriormente añadió abudante contenido a raíz de los cursos TIC de la Universidad de La Laguna.
- Carlos Pérez Pérez aportó el tema "StarOffice".
- Tomás Bautista Delgado aportó el tema de "LaTeX", resultado de modificar su estupendo manual "Una descripción de LaTeX"
- Carlos Alberto Morales Díaz escribió los temas de "Octave", "GNUplot" y parte de "Aplicaciones para Internet".
- Félix J. Marcelo Wirnitzer escribió los temas "Introducción a GNU/Linux", "El intérprete de comandos" y "HTML" y parte de "Administración básica" e "Instalación de GNU/Linux".
- Sergio García Reus aportó su traducción al castellano del tutorial de LyX.
- Carlos de la Cruz Pinto escribió con los temas "Emacs", "Java", "KDE" y mejoró el tema de "FreePascal".

- René Martín Rodríguez escribió los temas "Instalación de GNU/Linux" y "Depuradores".
- Edín Kozo colaboró en los temas "Aplicaciones para Internet" e "Instalación de GNU/Linux".
- Carlos Mestre González escribió el tema de "Administración básica" y parte de "Editores".
- Teresa González de la Fé colaboró con el tema "Aplicaciones diversas".
- Luis Cabrera Saúco colaboró con los temas "Introducción a GNU/Linux", "Aplicaciones para Internet" y "Aplicaciones diversas".
- Carlos Alberto Paramio Danta aportó el tema "gprof".
- Pedro A. Gracia Fajardo colaboró con el tema "The GIMP".

También agradecemos a todas las personas que nos han animado a seguir con este proyecto: alumnas y alumnos de la Facultad de Matemáticas y otras facultades de la Universidad de La Laguna, compañeros del Grupo de Usuarios de Linux de Canarias, Maribel C. Salgado, Pedro Reina, gente de HispaLinux y alguien más que se nos queda en el tintero (o en el teclado).

Índice general

Índice de figuras

Índice de tablas

Índice de ejemplos

MÓDULO I

Entorno GNU/Linux

TEMA 1

Introducción a GNU/Linux

1.1. Un poco de historia

1.1.1. Orígenes de UNIX y sus versiones.

UNIX es uno de los sistemas operativos más populares del mundo. Es una marca registrada de **The Open Group**, aunque originalmente fue desarrollado por **AT&T**.

UNIX es un sistema operativo real. Por sistema operativo real se entiende que debe tener como mínimo dos características: más de una persona puede acceder al mismo tiempo al ordenador y, mientras lo hacen, cada una de ellas puede ejecutar múltiples aplicaciones. A esto se le llama ser un sistema operativo *multiusuario* y *multitarea*. UNIX fue diseñado originalmente para ser ese tipo de sistema multitarea, en los años 70, y para que se pudiera ejecutar en *mainframes* y en miniordenadores.

Con UNIX, cada usuario accede al sistema utilizando un nombre de acceso. Opcionalmente, aunque es altamente recomendable, el usuario debe proporcionar una contraseña que asegura que la persona que accede es quien dice ser. Este mecanismo de control permite limitar el acceso de los usuarios a los ordenadores, algo especialmente útil cuando éstos están conectados en red. Dicho proceso de autentificación es conocido coloquialmente como `login`, puesto que ese es el nombre del programa que tradicionalmente se encarga del trabajo de permitir o denegar el acceso a un usuario.

UNIX funciona prácticamente en cualquier plataforma que haya sido construida. Muchos fabricantes han adquirido el código fuente (IBM; Hewlett-Packard, Sun, etc.) y desarrollado sus propias versiones, a las que han incorporado su toque personal a lo largo de los años. Pero no son los únicos que continúan modificando UNIX. Cuando el sistema se desarrolló por primera vez, el código fuente se proporcionó gratuitamente a las universidades y a los institutos. Dos de ellas han estado en primera línea desde el primer momento: la Universidad de California en Berkeley y el Instituto Tecnológico de Massachussetts.

Como podemos imaginar, el desarrollo de UNIX se produjo de forma bastante desordenada. Gente de todo el mundo comenzó a desarrollar herramientas y mejoras para UNIX. Desgraciadamente, no existía ninguna coordinación que guiase todo el desarrollo, lo cual produjo grandes diferencias entre las distintas versiones. Aunque a día de hoy muchas de esas diferencias se mantienen, en la actualidad la mayor parte de las implementaciones de UNIX cumplen con el estándar **IEEE POSIX.1**. Esto simplifica notablemente el desarrollo de aplicaciones, puesto que garantiza la compatibilidad entre las diferentes implementaciones de UNIX.

El mayor inconveniente de UNIX es que es muy grande. También es caro, especialmente en sus versiones para PC. Y aquí es donde aparece Linux, pues, como se explica con más detalle un poco más adelante, se diseñó para ser pequeño, rápido y barato. Hasta ahora los diseñadores han tenido éxito.

Linux fue creado originalmente por *Linus Torvalds* en la Universidad de Helsinki, Finlandia, allá por 1991. Linus basó el Linux en una pequeña implementación de UNIX para PC con fines didácticos llamada *Minix*. A finales de ese año se hizo público Linux con la versión 0.10. Un mes después, en diciembre, apareció la versión 0.11. Linus hizo que el código fuente fuera de libre disposición y animó a otras personas a colaborar en su desarrollo. Lo hicieron. Linux continúa su desarrollo hoy en día gracias a un equipo mundial dirigido por él mismo que trabaja a través de Internet.

Gran parte del *software* desarrollado para Linux se creó a través del proyecto GNU de la **Free Software Fundation**.

1.1.2. El *Software Libre* y la licencia GPL

Llegados a este punto, nos encontramos con un nuevo concepto: el *Software Libre*. Tiene su origen en el nacimiento del *software* en EE.UU., cuando la informática era un feudo reservado a empresas y universidades, y los programadores intercambiaban trucos (*hacks*, en inglés) que hacían brotar chispas en los enormes cerebros electrónicos. Por aquel entonces hacía sus pinitos digitales un joven programador, *Richard M. Stallman* que, al igual que sus compañeros de profesión, fue testigo de la primera gran transformación del mundo de la programación en industria cerrada.

Cuenta este informático que, un buen día, aparecieron por la puerta abogados que prohibieron a estos programadores compartir su código (el código de sus programas) y les obligaron a ocultar celosamente cualquier información que pudiera ser usada por la competencia. Además, decidieron que las empresas guardarían bajo llave el código fuente de sus programas (la secuencia original de instrucciones que los hace funcionar de una determinada manera) y sólo entregarían a sus clientes el código binario (los unos y ceros que el ordenador entiende, pero apenas pueden interpretar las personas). Por último, obligaron a los trabajadores a aceptar la idea de que quien violaba estas normas no sólo cometía un delito, sino también un pecado propio de un loco, o de un *pirata*.

Años más tarde, este dogma informático se extendió hasta convertirse en el actual mercado del *software*, donde comprar un programa significaba adquirir el derecho a usarlo, pero no a abrirlo para saber cómo funciona ni, mucho menos, a copiarlo o modificarlo; una prerrogativa que corresponde en exclusiva a la empresa fabricante.

Stallman, convencido de que a la sociedad se le había robado un debate importante sobre la evolución de la tecnología (la frase "*es como si te vendieran un coche con el capó sellado, para que no puedas ver el motor*" es una de las analogías más usadas para explicar esta realidad mercantil), decidió dejar su trabajo y emprender una tarea mucho más altruista: responder al modelo propietario con un *software* del que nadie pudiera apropiarse, con un *software* libre.

Se trataba, según su promotor, de poner en marcha un nuevo contrato por el que los usuarios recibieran siempre el código fuente y, además, el derecho inalienable a modificarlo a su gusto. A este movimiento se le bautizó con el críptico nombre de **GNU**, y para defenderlo se creó la **Licencia Pública General** (**GPL**, sus siglas en inglés), un peculiar contrato mercantil que, a diferencia de las licencias de *software* tradicionales, no sólo no restringe la posibilidad de copiar y redistribuir los programas, sino que anima a los usuarios a hacerlo.

Este nuevo orden informático fue recibido con entusiasmo en la entonces incipiente comunidad de programadores que pululaba por Internet, pero también con cierta inquietud. De hecho, el movimiento GNU fue visto con recelo desde algunos sectores de la población estadounidense, que lo tacharon de "izquierdoso", por su tendencia a compartir su trabajo y por su aversión al concepto de propiedad que había establecido la industria del *software*.

1.1.3. ¿Qué es Linux y GNU/Linux?

Con ideología o sin ella, el movimiento GNU se extendió por la Red y empezó a dar sus frutos, y así nació Linux. Línea a línea, programa a programa, el sistema operativo del pingüino (la mascota de Linux que, por cierto, responde al nombre de Tux) se convirtió en poco tiempo en el producto más famoso del código abierto y, paradójicamente, en la apuesta de más de una multinacional en el sector informático.

Lo que realmente se entiende por Linux es el *kernel*, el "corazón" de cualquier sistema operativo tipo UNIX. Pero el kernel por sí solo no forma todavía un sistema operativo. Justamente para UNIX existe multitud de *software* libre, lo que significa que también está disponible para Linux. Son estas utilidades las que realmente forman el sistema operativo. Entonces es cuando podemos hablar de **GNU/Linux**.

La gran cantidad de de programas de *software* libre permite la creación de diferentes sistemas Linux, cada uno con un conjunto de programas, entorno gráfico o sistema de instalación diferente. Cada una de estas agrupaciones particulares de software entorno al núcleo de Linux es lo que denominamos distribuciones. En general, cuando hablamos de Linux, nos estamos refiriendo a cualquiera de sus distribuciones. Las más conocidas son *Debian*, *Red Hat*, *SuSE*, *Mandrake*. De todas ellas, la que más se acerca a la filosofía del movimiento GNU es Debian ya que es fruto del trabajo de un gran grupo de voluntarios repartidos por el mundo, mientras que las otras pertenecen a empresas privadas dedicadas al desarrollo de Linux.

1.2. Distribuciones de Linux

Lo que vulgarmente conocemos como Linux, debiera llamarse oficialmente como GNU/Linux. El motivo no es otro sino que el corazón de un sistema Linux está formado por un núcleo (Linux) al que se le han añadido las utilidades desarrolladas por la gente de la *Free Software Foundation (GNU)*, o al menos así era en los primeros momentos.

Hoy en día, la lista de colaboradores en el desarrollo de Linux es inmensa, estando formada tanto por personas como usted, o como yo, como por las más grandes compañías del sector informático actual. Parece que ha pasado una eternidad (5 de Octubre de 1991), desde el momento que Linus anunció la primera versión "oficial" de Linux, la 0.02. Ya podía ejecutar bash (el shell de GNU) y gcc (el compilador de C de GNU), pero no hacía mucho más.

En ese anuncio, puso frases como éstas:

> *"[...]¿Suspiráis al recordar aquellos días de Minix-1.1, cuando los hombres eran hombres y escribían sus propios drivers? ¿Os sentís sin ningún proyecto interesante y os gustaría tener un verdadero S.O. que pudierais modificar a placer? ¿Os resulta frustrante el tener sólo a Minix? Entonces, este artículo es para vosotros.[...]"*

Hoy día, a partir de esas frases (y de lo que implicaban), han surgido distribuciones de Linux para todos los gustos. Algunas de ellas son las que figuran en este listado:

- **Debian**: La distribución de Linux más libre disponible hoy día, mantenida por un gran grupo de voluntarios.
  `http://www.debian.org`
- **e-smith server and gateway**: Software Open-source que convierte un PC en un Servidor de Internet Linux.
  `http://www.e-smith.net`
- **Mandrake**: Una de las distribuciones más recientes, basada en Red Hat y KDE.
  `http://www.linux-mandrake.com`
- **muLinux**: Distribución de Linux, totalmente configurable y minimalista, casi completa y orientada al usuario. muLinux reside en un solo diskette reformateado a 1722K y dispone de varios diskettes adicionales. Requisitos mínimos PC 386-8M
  `http://sunsite.dk/mulinux/`
- **NoMad**: Su principal propósito es ayudar a su creador a ser feliz, dándole algo que hacer en su tiempo libre.
  `http://www.nomadlinux.com`
- **Project Independence**: Project Independence busca facilitar la vida a los recién llegados a Linux.
  `http://independence.seul.org`
- **Red Hat**: La primera distribución Linux en dirigirse seriamente hacia los usuarios finales.
  `http://www.redhat.com`
- **SEUL (Simple End-User Linux)**: Un proyecto basado en Linux que pretende convertirse en una alternativa viable frente a los sistemas operativos comerciales.
  `http://www.seul.org`
- **Slackware**: Una distribución histórica: La Distribución.
  `http://www.slackware.com`
- **Stampede**: Una distribución compilada para Pentium.
  `http://www.stampede.org`
- **S.u.S.E.**: Una distribución alemana, orientada al usuario final. Colaboran activamente en el desarrollo del sistema X-Window.
  `http://www.suse.com`
- **Yellow Dog Linux**: Una distribución para ordenadores Macintosh PPC y G3.
  `http://www.yellowdoglinux.com`

1.3. Paquetes de software en Linux

El gran volumen de software disponible en la mayor parte de estas distribuciones suele hacer muy difícil incluso las tareas más básicas de administración. Para simplificar el proceso se suele recurrir a *empaquetar* el software.

Un *paquete* no es más que un archivo comprimido que contiene todo o parte de los archivos necesarios para ejecutar un determinado programa de software. Además, los paquetes contienen las rutinas necesarias para la correcta instalación,

desinstalación y actualización del software que contienen. El uso de los paquetes es tan potente que se ha extendido a otros elementos, tales como galerías de fondos de escritorio, documentación, scripts, etc.

Los criterios bajo los cuales un determinado grupo de programas se agrupa en un mismo paquete, o un único software monolítico se reparte entre varios, son particulares de cada distribución. Lo que sí es común a todas ellas es que la gestión de los mismos se hace a través de una compleja base de datos. Utilizando las herramientas adecuadas podemos consultar que paquetes están instalados o cuales están disponibles para su instalación. Con la mismas herramientas podemos instalar, desinstalar o actualizar un paquete garantizando la integridad de nuestro sistema. La base de datos mantiene información de las dependencia entre paquetes, por lo que podemos estar informados en todo momento de que paquetes se necesitan para instalar uno dado, o qué paquetes no podemos desinstalar porque son requeridos por otro que si lo está.

Los formatos de paquetes más extendidos son los `rpm` y los `deb`. Los primeros son utilizado mayoritariamente por la distribución *Red Hat* y sus derivadas (como *Mandrake* y *SuSE*). Los segundos son empleados por la distribución *Debian* y sus derivadas.

El empaquetado del software y la gestión de paquetes es una de las grandes ventajas que presentan las distribuciones Linux frente a otros sistemas operativos. Una vez que te acostumbras a sus ventajas se hace muy difícil vivir sin ellas.

Bueno, para terminar este rápido repaso, comentar que en este listado no se encuentran, ni muchísimo menos, todas las opciones de las que puede disfrutar con Linux. Basta dar una vuelta por los principales buscadores de Internet, y se dará cuenta uno de hasta dónde se ha llegado actualmente.

1.4. GNU/Linux: cara y cruz

1.4.1. La cara

Enumeramos las ventajas de Linux a continuación:

- **Multitarea total**. Se pueden ejecutar varias tareas y se puede acceder a varios dispositivos al mismo tiempo.
- **Memoria virtual**. Linux puede usar una porción de su disco duro como memoria virtual, lo que aumenta la eficiencia del sistema al mantener los procesos activos en la memoria física (RAM) y al colocar las partes inactivas o usadas con menos frecuencia en la memoria de disco. La memoria virtual permite utilizar la máxima cantidad de memoria posible del sistema y permite que no se produzca fragmentación de la memoria.
- **Soporte multiusuario**. Linux permite que varios usuarios accedan a su sistema simultáneamente sin que haya conflicto entre ellos, proporcionándole a cada su propio espacio de trabajo.
- **Código fuente no propietario**. El kernel de Linux no utiliza código de **AT&T** ni ninguna otra fuente propietaria. Otras organizaciones, como las compañías comerciales, el proyecto GNU y los programadores de todo el mundo, han desarrollado *software* para Linux.
- **Soporte mediante *software* GNU**. Linux puede ejecutar una amplia variedad de *software*, disponible gracias al proyecto GNU. Este *software* incluye de todo, desde desarrollo de aplicaciones (GNU C y GNU C++) a la administración del sistema (gawk, groff, etc.) y juegos (GNU Chess, GnuGo, y NetHack).
- **Estabilidad**. Linux presenta una gran estabilidad en la gestión de sus procesos internos del sistema. Es muy difícil conseguir que Linux se "cuelgue" y, por supuesto, jamás se verá un "pantallazo azul" de los conocidos por Microsoft Windows.
- **Gran oferta de software**. Aunque Linux no sea tan conocido por el público como lo es Microsoft Windows, eso no quiere decir que no existan aplicaciones para el usuario medio. Por el contrario, cuando uno utiliza Linux, según pasan los días, se tiene la sensación de que no necesita para nada Microsoft Windows porque todo lo que éste ofrece ya lo tiene Linux. Llega un momento en que no se echa de menos a Microsoft Windows.
- **Defensa contra los virus**. Aunque la mayor parte de los virus que rondan por Internet son desarrollados para Microsoft Windows, es cierto que existen algunos para Linux, pero son más difíciles de crear debido a que Linux emplea un sistema de permisos sobre los ficheros previniendo los posibles desastres que se ven todos los días en los entornos de Microsoft Windows. Si a eso añadimos que los virus para Microsoft Windows no se pueden ejecutar en Linux salvo contadísimos casos, nos podemos hacer una idea del grado de seguridad con que cuenta Linux.

- **Relación con Internet**. Debido a que Linux creció gracias a Internet, digamos que ambos hablan en el mismo lenguaje y por tanto, se ve claramente que navegar por Internet con Linux es más rápido que con Microsoft Windows. Obviamente, ésta es una opinión personal del autor de este documento.
- **Entornos gráficos**. Hasta hace unos años, trabajar en Linux sólo era posible desde consola, ese entorno negro con caracteres blancos (similar al MS-DOS, pero más potente). Con la llegada de Microsoft Windows, la comunidad Linux se vio de forma obligada a desarrollar nuevos entornos gráficos para no perder el tren. La gran diferencia con Microsoft Windows es que mientras éste sólo dispone de un escritorio, en Linux podemos elegir con qué escritorio queremos trabajar. Los más conocidos son KDE, GNOME, AfterStep, Enlightenment y Window Maker; donde los dos primeros son los más populares. Lo más curioso del asunto es que con ellos se ha iniciado una "guerra dialéctica" sobre cuál es mejor. Al final, la ventaja reside en la variedad con la que el usuario puede decidir con cuál se siente más cómodo.
- **Servidores caseros**. Parece mentira, pero en casa podemos tener un servidor web, o un servidor FTP con nuestro Linux. Sólo hay que leer la documentación de cómo hay que hacerlo.
- **La comunidad Linux**. A diferencia de Microsoft Windows, Linux tiene una comunidad de voluntarios con ganas de ayudar a los que les cuesta adentrarse en este mundo. Y todo por afán de colaborar en este proyecto. Podemos decir que esta comunidad tiene un gran sentimiento de solidaridad.

1.4.2. La cruz

Obviamente, como todo sistema operativo, Linux no está exento de desventajas, como vemos aquí:

- **Entorno árido**. Aunque en los últimos años el panorama ha mejorado considerablemente, no podemos olvidar que para trabajar con Linux, sobre todo si se viene de Microsoft Windows, hay que aprender unas cuantas nociones si no se quiere tener la sensación de que se está perdido. Todo ello radica en que Linux no es tan intuitivo como Microsoft Windows, pero afortunadamente las diferentes compañías comerciales han aportado herramientas que facilitan esas tareas al usuario. Sólo hay que cambiar un poco el "chip".
- **Soporte de hardware**. Por desgracia, como se ve claramente, el mercado de los sistemas informáticos de escritorio está orientado a Microsoft Windows. Prueba de ellos es que el 90% de los PC lo tienen instalado sin usar otro. Así que, los fabricantes de hardware sólo se han preocupado de crear los controladores de dispositivo compatibles con Microsoft Windows sin pensar en los restantes sistemas como Linux, Mac, BeOS y otros. Ello ha obligado a que fueran los propios usuarios programadores los que desarrollaran sus propios controladores de dispositivo. Con el tiempo, dada la demanda creciente de usuarios decididos a utilizar Linux algunos fabricantes ha comenzado a acceder a las demandas de estos últimos.
- **Configuración del sistema operativo**. Hasta hace bien poco, instalar y configurar Linux en su casa en condiciones era una tarea ardua y bastante complicada. Ello hacía que mucha gente se rindiera y siguiera con Microsoft Windows. Las diferentes distribuciones han tenido en cuenta estos problemas y han aportado herramientas que ayudan al sufrido usuario a hacerlo todo de forma más intuitiva y automática. Entre estas herramientas están las encargadas de la parte de la instalación del *hardware* y *software*.
- **No todo el software necesario está presente**. Es cierto que casi todo el software que un usuario necesita ya lo aporta Linux. Pero hay situaciones en las que una persona necesita una aplicación concreta y ve que debe utilizar Microsoft Windows para usar esa aplicación. De todas formas, siempre surgen desarrolladores con afán de ayudar que terminan desarrollando esas aplicaciones.
- **Administración de Linux**. Como todo, si se quiere administrar Linux de forma profunda, ya no vale usar los asistentes gráficos como en Microsoft Windows, sino que hay que leer mucha documentación y experimentar. A cambio se gana una experiencia que en Microsoft Windows es difícil de conseguir.

1.5. Recursos para GNU/Linux

1.5.1. Software

Tal vez por su escasa repercusión en el ámbito doméstico, uno pueda pensar que la cantidad de software es bastante escasa. Todo lo contrario. De hecho se pueden encontrar bastantes utilidades en las páginas webs propias de las distribuciones.

También se pueden encontrar aplicaciones en otras páginas o servidores FTP con software propio, como StarOffice, KDE, QCAD y VariCAD (éstos dos últimos son aplicaciones que aspiran a ser serias alternativas a AutoCAD aplicadas a los entornos UNIX), etc.

Si el usuario proviene del entorno Microsoft Windows, tal vez pueda tener la sensación de que "*Sí, trabaja como Windows, pero no es igual que Windows...*", y le gustaría encontrar las herramientas con el mismo aspecto que este sistema operativo. Los desarrolladores han optado por tres caminos:

- El primero es el de dar las mismas funcionalidades que las que tienen esas aplicaciones que existen en Microsoft Windows, aunque su aspecto externo no tenga nada que ver con el Microsoft Windows.
- El segundo camino añade además el aspecto externo original de la aplicación que funciona en Microsoft Windowscon el fin de que el usuario se encuentre en un entorno familiar.
- El tercero es el más drástico ya que se crean herramientas con aspectos y funcionalidades totalmente diferentes a esas aplicaciones, pero igualmente eficientes e inclusos superiores en algunos casos.

Para plasmar en un ejemplo de lo que se acaba de decir, basta con nombrar una de las aplicaciones más utilizadas en el entorno Microsoft Windows que es MS Word. Del primer ejemplo nos encontramos con KWord, desarrollado por el equipo que creó KDE.

El segundo es AbiWord. El autor reconoce que es casi como utilizar el MS Word, aunque es evidente que aún le faltan bastantes opciones por desarrollar.

Y el tercero es el LaTeX, una herramienta muy potente, pero que quizás desconcierte a los usuarios noveles debido a su complejidad. Como anécdota aclarativa, el libro que el lector tiene ahora en sus manos y está leyendo se ha escrito usando precisamente LaTeX. Al final la elección depende del propio usuario.

Una cosa está clara: **Hay software más que suficiente para Linux como para detener un tren**.

1.5.2. Documentación

La documentación básica sobre Linux se puede encontrar en `http://es.tldp.org/` (en castellano) y `http://www.tldp.org/` (en inglés). En ellos se pueden encontrar tutoriales y cursos que pueden ayudar a los usuarios a adentrarse en el mundo Linux.

Cabe destacar que los documentos más utilizados son los COMOs (*HOW-TO* en inglés) que son una guía bastante útil sobre alguna cosa en concreto que se quiere hacer.

Otra forma de encontrar información es buscando en las múltiples páginas webs referidas al tema, sin olvidar también echar un vistazo a

Por último, si se quiere información rápida de algún programa en concreto, no hay más que utilizar los manuales, es decir, el comando `man`, las páginas info con el comando `info` o buscar en la documentación del programa en cuestión en los directorios `/usr/share/` y `/usr/share/doc/`.

TEMA 2

El intérprete de comandos

2.1. ¿Qué es un intérprete de comandos?

Si bien manejarse en Linux es cada vez más fácil, debido a la proliferación de escritorios, los comienzos no siempre fueron así de fáciles. De hecho, puede ocurrir que nos encontremos con una emergencia en el que no nos quede más remedio que trabajar con comandos.

Un intérprete de comandos tiene el aspecto de una pantalla llena de letras, generalmente con fondo negro y letras blancas[1], y que en la última línea inferior, se suele ver lo siguiente:

```
[cila@gulic]$
```

En este entorno es donde introduciremos los comandos con los que trabajaremos, y coloquialmente diremos que estamos trabajando en una consola. Estos comandos pueden ser de diferentes clases:

- Programas ejecutables.
- Scripts (guiones) del intérprete.
- Scripts (guiones) de lenguajes de script como Python, Perl, Tcl, etc.
- Macros del intérprete.

Todos tienen en común que son ficheros: al cargar un programa en Linux, se ordena al intérprete que busque el fichero con el nombre del programa y una vez encontrado, lo ejecute si el usuario tiene permisos de ejecución.

Los comandos tienen el siguiente aspecto:

```
[cila@gulic]$ fdisk
[cila@gulic]$ lsmod
[cila@gulic]$ ls
```

También funcionan con opciones:

```
[cila@gulic]$ fdisk -v
[cila@gulic]$ ls -a -l
[cila@gulic]$ ls -al
```

Y con parámetros:

[1] Aunque inicialmente eran de fondo negro con letras de color naranja o verde.

```
[cila@gulic]$ fdisk /dev/hda
[cila@gulic]$ ls /tmp
[cila@gulic]$ ls *.txt
```

Con opciones y parámetros:

```
[cila@gulic]$ rpm -qpl joe-1.0.3.rpm
[cila@gulic]$ gcc -o suma suma.c
[cila@gulic]$ ls -al /tmp
```

Al ser Linux un sistema *multitarea* y *multiusuario*, se aportan ventajas que se agradecen incluso en un sistema PC *monousuario*. Una de estas ventajas es que se puede trabajar con seis consolas virtuales, que es como si pudiéramos trabajar con varias sesiones simultáneas, entendiendo por sesión el tiempo de trabajo desde que el usuario entra tras identificarse en el `login` de entrada hasta que abandona el sistema. Lo que significa realmente que el mismo usuario puede entrar varias veces al mismo tiempo.

Para alternar entre estas consolas virtuales, basta con pulsar las combinaciones de la teclas `A-F1` a `A-F6`.

NOTA: En lo sucesivo usaremos esta nomenclatura:

- `A-F1` ⇒ `Alt` y `F1`.
- `C-F1` ⇒ `Control` y `F1`.
- `S-F1` ⇒ `Shift` y `F1`.

Donde `Shift` es la tecla de las mayúsculas.

Si se quiere acceder a una consola desde un entorno gráfico, entonces se pulsan las combinaciones `C-A-F1` a `C-A-F6`

2.2. Directorios y nombres de ficheros

2.2.1. Trabajar con directorios.

2.2.1.1. Estructura del árbol de directorios

Toda la información (ya sean textos, imágenes, bases de datos o información para la configuración del sistema) se almacena en *ficheros*, que a su vez se guardan en *directorios*. Con todas las herramientas y programas existentes se puede acceder a estos ficheros para ver su contenido o modificarlo. En la Figura 2.1 podemos ver una ejemplo de dicha estructura en árbol.

Todos los ficheros y directorios de un sistema UNIX cuelgan de un directorio principal llamado *raíz*, que se representa como `/`. En la Figura 2.1 podemos observar como del directorio raíz `/` cuelgan otros directorios, como `etc` o `home`.

En dicho esquema diferenciamos los directorios de los ficheros, complementando el final del nombre de los primeros con el carácter `/`. En el mismo esquema vemos como algunos directorios pueden contener otros directorios, como es el caso de `home`, que es el *padre* de los directorios `usr1` y `usr2`. También pueden contener algunos ficheros, como es el caso de `dev`, que es el padre de `dsp`; o el propio `usr2`, que es el padre de `trabajo.txt`. Este último fichero tiene la *ruta absoluta* `/home/usr2/trabajo.txt`, que define su localización dentro del árbol de directorios.

Todos los directorios de un sistema UNIX contienen al menos dos subdirectorios. El primero es `./` que representa al propio directorio, mientras que el segundo es `../` y representa al directorio padre. Por ejemplo, el directorio `./` dentro de `/home/usr1` es el propio `/home/usr1`, mientras que el directorio `../` dentro de `/home/usr1` es `/home`, es decir, su directorio padre.

En los sistemas UNIX todo son ficheros. Dispositivos como discos duros, scanners o disqueteras se representan como *archivos especiales* en el directorio `/dev`. Por ejemplo, en la Figura 2.1 vemos el fichero `/dev/dsp` que suele corresponderse con la tarjeta de sonido instalada en el sistema.

A diferencia de los sistemas MS-DOS/Windows, en los sistemas UNIX no se reparten los directorios en función de si están en una unidad física o en otra (`C:`, `D:`, etc.). Durante el arranque del sistema, cada uno los archivos representativos de los diferentes discos duros, particiones y demás elementos de almacenamientos son asociados a un directorio del directorio raíz. A este proceso se lo denomina *montaje*, y no sólo es revesible sino que es completamente configurable. Por ejemplo, en la Figura 2.1 el directorio `/home` podría estar en una partición diferente en un disco duro distinto al que contiene a `/` o a `/etc`.

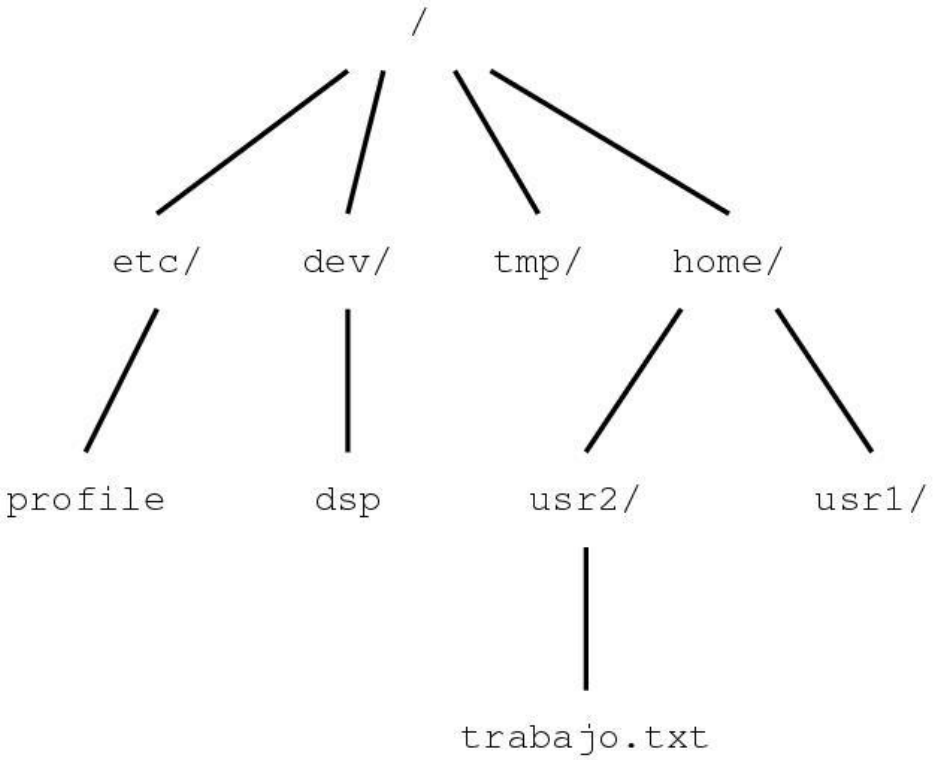


Figura 2.1: Estructura en árbol de directorios en sistemas UNIX

Nada en el esquema o en el trabajo con el sistema nos permite apreciar la diferencia. Por tanto, no tiene sentido escribir en la consola `C:`, tal y como haríamos en MS-DOS. Solamente debemos dirigirnos al directorio asociado a esa partición que para MS-DOS es `C:`.

Hay que destacar que cuando el usuario accede a una sesión, Linux "envía" al usuario a su directorio de trabajo. Cuando entro como el usuario `cila`, en el momento de entrar me encontraré en el directorio `/home/cila`. Éste será mi directorio personal, en donde tengo libertad absoluta para hacer lo que quiera con mis ficheros y directorios ubicados ahí. Sin embargo no podré hacer todo lo que quiera en el directorio `/home/pepe`. ¿Por qué? Pues por la sencilla razón de que Linux tiene un sistema de permisos que concede o restringe libertades sobre los directorios y ficheros que hay en Linux. ¿Significa eso que puede existir un usuario "dios" en Linux que puede hacer totalmente cualquier cosa en Linux? Sí, ése es el usuario `root`. Sin embargo, para los propósitos de este curso, sólo nos remitiremos a la cuenta de trabajo del usuario *cila*.

2.2.1.2. Comandos sobre el árbol de directorios

Para movernos por el árbol de directorios emplearemos el comando `cd` (Change Directory)

```
[cila@gulic]$ cd /etc
```

Es decir, nos vamos al directorio `/etc`

Si simplemente escribimos `cd` sin especificar el nombre del directorio, esto será igual que escribir `cd /home/cila` o `cd`, es decir, me envía a **mi propio directorio de trabajo** (que es como irse a casa).

¿Cómo sé yo en qué directorio me encuentro? Basta con escribir el comando `pwd` (Print Work Directory).

```
[cila@gulic cila]$ pwd
/home/cila/
```

Y la salida que obtendré es:

```
/home/cila
```

En caso de querer listar los ficheros y subdirectorios de un directorio dado escribimos `ls nombre_directorio`.

```
[cila@gulic]$ ls /home/cila/apuntes/apuntes/
Apuntes_CILA_2001.dvi   CVS/                introduccion.sgml
programando.sgml        xwindow.sgml        Apuntes_CILA_2001.sgml
editores.sgml           LEEME               recursos.sgml
cabecera.sgml           final.sgml          Makefile
resumen_temario.txt     comandos.sgml       graficos.sgml
matematicas.sgml        sobre.sgml          compila*
internet.sgml           presentacion.sgml   temario.estado
```

El comando `ls` admite parámetros tales como `-a, -l`

```
[cila@gulic]$ ls -a -l
[cila@gulic]$ ls -al
[cila@gulic]$ ls -la
```

Obsérvese que ambas formas de escribir los parámetros son igualmente válidas. Como anotación, si queremos ver que parámetros se pueden utilizar en un comando, normalmente basta con escribir `nombre_comando --help`.

```
[cila@gulic]$ ls --help
Modo de empleo: ls [OPCIÓN]... [FICHERO]...
Muestra información acerca de los FICHEROs (del directorio actual por defecto).
Ordena las entradas alfabéticamente si no se especifica ninguna de las
opciones -cftuSUX ni --sort.

Los argumentos obligatorios para las opciones largas son también obligatorios
para las opciones cortas
  -a, --all                  do not hide entries starting with .
  -A, --almost-all           do not list implied . and ..
      --author               print the author of each file
  -b, --escape               print octal escapes for nongraphic characters
      --block-size=TAMAÑO    utiliza bloques de TAMAÑO bytes
  -B, --ignore-backups       no muestra la entradas que terminan con ~
  -c                         con -lt: ordena por ctime y muestra ctime (fecha
                               de última modificación del fichero)
                               con -l: muestra ctime y ordena por nombre
                               en cualquier otro caso: ordena por ctime
  -C                         muestra las entradas por columnas
      --color[=CUÁNDO]       especifica si se usará color para distinguir los
                               tipos de ficheros. CUÁNDO puede ser 'never',
                               'always' o 'auto'
  -d, --directory            muestra las entradas de los directorios en lugar
                             de sus contenidos
  -D, --dired                genera el resultado para el modo 'dired' de Emacs
  -f                         no ordena, utiliza -aU, no utiliza -lst
  -F, --classify             añade un indicador (uno de */=@|) a las entradas
      --format=PALABRA       across -x, commas -m, horizontal -x, long -l,
                               single-column -1, verbose -l, vertical -C
      --full-time            como -l --time-style=full-iso
  -g                         como -l, pero no muestra el propietario
  -G, --no-group             no muestra la información del grupo
  -h, --human-readable       muestra los tamaños de forma legible
                             (p.e. 1K 234M 2G)
      --si                   análogo, pero utilizando potencias de 1000,
```

```
                           no de 1024
-H, --dereference-command-line  sigue los enlaces simbólicos en la línea de
                                órdenes
    --indicator-style=PALABRA  añade un indicador con estilo PALABRA a los
                               nombres de las entradas: none (predeterminado),
                               classify (-F), file-type (-p)
-i, --inode                muestra el número de nodo-i de cada fichero
-I, --ignore=PATRÓN        no lista las entradas que coincidan (encajen)
                           con PATRÓN de shell
-k                         como --block-size=1K
-l                         utiliza un formato de listado largo
-L, --dereference          al mostrar la información de un fichero para un
                             enlace simbólico, muestra la información del
                             fichero al que apunta el enlace en lugar de la
                             del propio enlace
-m                         rellena el ancho con una lista de entradas
                           separadas por comas
-n, --numeric-uid-gid      como -l, pero muestra los UIDs y GIDs numéricos
-N, --literal              muestra los nombres literalmente (no trata p.ej.
                           los caracteres de control de forma especial)
-o                         como -l, pero no muestra el grupo
-p  --file-type            añade un indicador (uno de /=@|) a las entradas
-q, --hide-control-chars   imprime ? en lugar de los caracteres no gráficos
    --show-control-chars   muestra los caracteres no gráficos tal y como
                           son (predeterminado a menos que el programa sea
                           'ls' y la salida sea un terminal)
-Q, --quote-name           encierra los nombres de las entradas entre
                           comillas
    --quoting-style=PALABRA  utiliza el estilo de cita PALABRA para los
                             nombres de las entradas:
                             literal, locale, shell, shell-always, c, escape
-r, --reverse              invierte el orden, en su caso
-R, --recursive            muestra los subdirectorios recursivamente
-s, --size                 muestra el tamaño de cada fichero, en bloques
-S                         ordena los ficheros por tamaño
    --sort=PALABRA         extension -X, none -U, size -S, time -t, version -v
                           status -c, time -t, atime -u, access -u, use -u
    --time=PALABRA         muestra la fecha según PALABRA, en lugar de la
                           fecha de modificación:
                             atime, access, use, ctime ó status; utiliza
                             la fecha especificada como clave de ordenación
                             si --sort=time
    --time-style=PALABRA   muestra la fecha utilizando el estilo PALABRA:
                             full-iso, iso, locale, posix-iso, +FORMATO
                           FORMATO se interpreta como en 'date'; si FORMATO
                           es FORMATO1<nuevalínea>FORMATO2, FORMATO1 se
                           aplica a los ficheros no recientes y FORMATO2
                           a los ficheros recientes
-t                         ordena por la fecha de modificación
-T, --tabsize=COLS         establece los topes de tabulación a cada COLS
                           en lugar de 8
-u                         con -lt: ordena por atime y muestra atime (fecha
                             de último acceso al fichero)
                             con -l: muestra atime y ordena por nombre
                             en cualquier otro caso: ordena por atime
-U                         no ordena; muestra las entradas en el orden del
                           directorio
-v                         ordena por versión
```

```
  -w, --width=COLS           establece el ancho de la pantalla en lugar del
                             valor actual
  -x                         muestra las entradas por líneas en vez de por
                             columnas
  -X                         ordena alfabéticamente por la extensión de la
                             entrada
  -1                         muestra un fichero por cada línea
      --help     muestra esta ayuda y finaliza
      --version  muestra la versión y finaliza

TAMAÑO puede ser (o puede ser un entero seguido opcionalmente por) uno
de los siguientes: kB 1.000, K 1.024, MB 1.000.000, M 1.048.576, y así
en adelante para G, T, P, E, Z, Y.

Por defecto, no se emplea color para distinguir los tipos de ficheros. Esto
equivale a usar --color=none. Usar la opción --color sin el argumento opcional
CUÁNDO equivale a usar --color=always. Con --color=auto, sólo se muestran
los códigos de color si la salida estándar está conectada a un terminal (tty).

Comunicar bichos a <bug-fileutils@gnu.org>.
```

Para consultar con detenimiento esta ayuda, el autor recomienda usar `ls --help | more`, y que según se avanza con la información, se pulsa la barra espaciadora, y para salir, se pulsa la tecla `q`.

Obsérvese que se pueden escribir los parámetros de dos formas: una corta (`-l, -a`) y otra larga (`--all, --help`).

Para crear un directorio, usaremos `mkdir nombre_directorio`.

```
[cila@gulic]$ mkdir pepe
[cila@gulic]$ mkdir tmp
```

Mientras que para eliminarlo usaremos `rmdir nombre_directorio`.

NOTA: El directorio que se quiere eliminar debe estar vacío y no debe haber nadie trabajando en él en ese momento. En caso de que haya otro usuario dentro, el sistema avisará con un mensaje de error y desistirá todo intento de eliminación.

Evidentemente, podemos crear y destruir un directorio dando su ruta completa o sólo su nombre si nos encontramos en el directorio que lo contiene. En caso de querer borrar de un sólo golpe un directorio y todo su contenido disponemos del comando `rm` con las opciones `-rf`.

NOTA: Mucho cuidado con borrar directorios enteros sin comprobar lo que se hace, porque ésta es una operación irreversible.

```
[cila@gulic]$ rmdir pepe
[cila@gulic]$ rm -rf tmp
```

Obsérvese que los comandos anteriores borran los subdirectorios `pepe` y `tmp` del directorio actual. Como hemos comentado, podemos emplear rutas absolutas para crear o borrar un directorio cualquiera del árbol de directorios.

```
[cila@gulic]$ mkdir /home/cila/pepe
[cila@gulic]$ rmdir /home/cila/pepe
```

2.2.2. Trabajando con ficheros

2.2.2.1. Comandos sobre el árbol de ficheros

El comando para copiar un fichero es `cp fichero_origen fichero_destino`, es decir, que copiamos el fichero `pepe.txt` en `juan.txt`.

```
[cila@gulic]$ cp pepe.txt juan.txt
```

El comando para mover o renombrar un fichero es `mv fichero_origen fichero_destino`, es decir, que copiamos el fichero `pepe.txt` en `juan.txt`, pero `pepe.txt` deja de existir físicamente.

```
[cila@gulic]$ mv pepe.txt juan.txt
```

NOTA: El intérprete de comandos **SÍ** distingue en mayúsculas y minúsculas, tanto en el caso de los comandos como en el de los ficheros y directorios. Esto significa que el comando mv es totalmente diferente a `Mv`, `mV` y `MV`. Asimismo, el fichero `pepe.txt` no es el mismo fichero que `Pepe.txt`, ni que `PEPE.TXT`, etc.

En caso de que deseemos borrar definitivamente un fichero podemos emplear el comando `rm nombre_fichero`.

```
[cila@gulic]$ rm pepe.txt juan.txt
```

2.2.2.2. Utilización de comodines

En ocasiones el nombre de los directorios o ficheros sobre los que estamos trabajando contienen partes comunes que podemos utilizar con ayuda de comodines para facilitarnos el uso de la interfaz de comandos.

En general el carácter `*` al indicar el nombre de un fichero o directorio es sustituido por un número indeterminado de cualquier combinación de caracteres. Por ejemplo,

```
[cila@gulic]$ rm pe*
```

borrará cualquier ficheros que empiecen por `pe` en el directorio actual. Mientras que,

```
[cila@gulic]$ cp *txt* tmp/
```

copiará los ficheros que contengan la cadena `txt` en el nombre al directorio `tmp`.

El carácter `?` al indicar el nombre de un fichero sólo representa a *un* carácter cualquiera.

```
[cila@gulic]$ mv pepe?.txt tmp/
```

El ejemplo anterior moverá archivos como `pepe.txt` o `pepa.txt` al directorio `tmp`.

2.2.2.3. Sistema de permisos

A la hora de trabajar con ficheros, es necesario entender el sistema de permisos de los ficheros y directorios. Si escribimos `ls -l`, nos encontramos con la siguiente salida:

```
[cila@gulic]$ ls -l
total 468
-rw-rw-r--    1 cila    cila      163004 oct 29 10:05
 Apuntes_CILA_2001.dvi
-rw-rw-r--    1 cila    cila      119151 oct 29 10:05
 Apuntes_CILA_2001.sgml
-rw-rw-r--  1 cila  cila    1617 oct 28 22:15 cabecera.sgml
-rw-rw-r--  1 cila  cila   13329 oct 29 10:05 comandos.sgml
-rwx------  1 cila  cila      33 oct 29 00:32 compila*
drwxrwxr-x  2 cila  cila    4096 oct 28 23:20 CVS/
-rw-rw-r--  1 cila  cila   17250 oct 28 12:11 editores.sgml
-rw-rw-r--  1 cila  cila      12 oct 27 23:10 final.sgml
-rw-rw-r--  1 cila  cila     157 oct 27 23:10 graficos.sgml
-rw-rw-r--  1 cila  cila    2816 oct 28 21:51 internet.sgml
-rw-rw-r--  1 cila  cila   23308 oct 28 23:05 introduccion.sgml
-rw-rw-r--  1 cila  cila     402 oct 27 23:10 LEEME
-rw-rw-r--  1 cila  cila    2295 oct 28 22:19 Makefile
-rw-rw-r--  1 cila  cila   13087 oct 28 17:30 matematicas.sgml
-rw-rw-r--  1 cila  cila     652 oct 28 21:56 presentacion.sgml
-rw-rw-r--  1 cila  cila    34797 oct 28 21:56 programando.sgml
-rw-rw-r--  1 cila  cila      47 oct 28 21:56 recursos.sgml
-rw-rw-r--  1 cila  cila    1320 oct 25 13:35 resumen_temario.txt
-rw-rw-r--  1 cila  cila    4662 oct 28 22:17 sobre.sgml
-rw-rw-r--  1 cila  cila    5247 oct 28 22:19 temario.estado
-rw-rw-r--  1 cila  cila    7417 oct 27 23:10 xwindow.sgml
```

La primera letra a la izquierda de cada línea nos indica si se trata de un fichero ("-") o un directorio ("d").

Después nos encontramos con tres grupos de tres letras (`rwx`), que según estén activados (la propia letra, `r`,`w`,`x`) o desactivados (un guión, -) nos concede o deniega permisos de lectura (`r`), escritura (`w`) y ejecución (`x`). ¿Y por qué son tres grupos? Pues porque las tres primeras letras se refieren al propio usuario que es el dueño de esos ficheros, el segundo grupo se refiere al grupo de usuario que pertenece ese usuario, y el tercero a los usuarios "extraños" o "ajenos" al usuario. Por tanto, si leemos:

$$-\underbrace{rw-}_{u}\underbrace{rw-}_{g}\underbrace{r--}_{o}$$

vemos que se trata de un fichero (-) con permisos de lectura y escritura para el usuario y el grupo al que pertenece, y de sólo lectura para un "extraño".

El siguiente sería un fichero de lectura, escritura y ejecución únicamente para el usuario propietario del fichero.

$$-\underbrace{rwx}_{u}\underbrace{---}_{g}\underbrace{---}_{o}$$

Este último ejemplo es un directorio (`d`) con permisos de lectura, escritura y ejecución para el usuario y el grupo, y de lectura y ejecución para el "extraño". En el caso de directorios, el permiso de ejecución es equivalente a permiso para "ejecutar" los programas que hay en el directorio.

$$d\underbrace{rwx}_{u}\underbrace{rwx}_{g}\underbrace{r-x}_{o}$$

En el ejemplo del listado anterior vemos dos veces el nombre de "cila". El de la primera columna se refiere al usuario propietario, y el segundo es el nombre del grupo, que casualmente coincide con el nombre del usuario.

NOTA: Debemos tener en cuenta que **jamás podremos eliminar un fichero o retocarlo si no tenemos permisos de escritura sobre él**.

Si queremos que un fichero cambie de propietario, lo haremos, con `chown`:

```
[cila@gulic]$ chown miguel pepe.txt
```

si antes, `pepe.txt` era de `cila`, ahora pasa a ser de `miguel`.

De igual forma, para cambiarlo de grupo, usaremos `chgrp`. Si `pepe.txt` era del grupo de los profesores, y queremos que sea del grupo de los estudiantes, sólo habrá que escribir lo siguiente:

```
[cila@gulic]$ chgrp estudiantes pepe.txt
```

En algunos sistemas no se puede cambiar el propietario de un fichero bajo ciertas condiciones:

```
$ ls -l hola
-rw-r--r--    1 miguev   108             0 ago 10 20:27 hola
$ chown frodo hola
chown: hola: Operación no permitida
```

Esto es normal en sistemas donde hay cuotas de usuario. Las cuotas son un mecanismo de limitación para que los usuarios no puedan ocupar más de un determinado volumen (su cuota) en el disco. Si este mecanismo está activo no se permite a los usuarios cambiar el propietario de ningún fichero, ya que podría usarse este cambio para ocupar la cuota de otro usuario.

Finalmente, para cambiar los permisos de un fichero, lo haremos con `chmod`, indicando a que tipo de usuario queremos asignarlos y sobre qué permisos. Para indicar el usuario propietario, usaremos el parámetro `u`, el de grupo será `g` y el ajeno será `o`, (de `otros`). Para indicar el tipo de permiso, usaremos las letras `r`, `w`, `x`, según sean de lectura, escritura o ejecución respectivamente. Y para conceder o denegar, usaremos los símbolos "+" y "-":

```
[cila@gulic]$ chmod u+rwx pepe.txt
```

Este ejemplo sirve para dar todos los permisos al usuario.

En el siquiente ejemplo daremos permisos de lectura y ejecución al usuario y al grupo, pero no de escritura sobre el fichero `compila`.

```
[cila@gulic]$ chmod ug+r-x compila
```

O quitar el permiso de ejecución a todos los usuarios sin que se vean afectados los otros tipos de permisos:

```
[cila@gulic]$ chmod -x probar
```

2.3. Comandos básicos para sobrevivir

A parte de los comandos para el manejo de ficheros y directorios existen algunos otros que conviene conocer puesto que simplifican notablemente nuestro quehacer diario.

2.3.1. Teclas especiales

La interfaz de comandos está llena de atajos de teclado diseñados para facilitarnos la vida. La mayor parte se apoyan en el uso de un historial de los últimos comandos ejecutados. Éstos son algunos de los atajos más importantes:

`Teclas del cursor` Las flechas hacia arriba y hacia abajo nos permiten elegir un comando de entre los almacenados en el historial de la interfaz de comandos. Las flechas hacia la izquierda y hacia la derecha nos permiten movernos por la línea de comandos para editarla.

`TAB` Si mientras escribimos el nombre de un comando o el de un fichero tenemos alguna duda podemos pulsar `TAB`. La interfaz de comandos nos ayudará completando el nombre en la medida de lo posible. En caso de que no sepa darnos una respuesta por haber varias soluciones disponibles podemos pulsar nuevamente para que nos muestre una lista de las mismas. La interfaz de comandos nos lo indicará con una señal sonora si no hay ninguna solución posible.

`S-Re. Pag.` y `S-Av. Pag.` Nos permiten movernos por el búffer de pantalla de la consola para ver texto que en condiciones normales no podemos observar puesto que el desplazamiento vertical lo ha dejado fuera de la misma.

`C-l` Limpiar la pantalla de la consola.

`C-r` Buscar comandos en el historial.

2.3.2. Imprimir en la salida estándar

Uno de los comandos más prácticos y utilizados es `cat`. Dicho comando encadena los archivos especificados y los imprime por pantalla uno detrás de otro.

```
[cila@gulic]$ cat pepe.txt juan.txt
```

En realidad los comandos de la consola de GNU/Linux no entienden la salida por pantalla y la entrada por teclado de la misma forma que la entendemos nosotros. La mayor parte de los comandos toman toda o parte de la información que necesitan para realizar su trabajo de lo que se denomina la *entrada estándar*. De la misma manera, utilizan la *salida estándar* para mostrar los resultado de su trabajo, y la *salida de error estándar* para mostrar los mensajes de los errores producidos durante la realización de la misma.

En general la *entrada estándar* suele estar asociada a la entrada por teclado, mientras que la *salida estándar* y la *salida de error estándar* suelen estar sociadas a la salida por pantalla. Sin embargo, Linux nos proporciona mecanismos para que cualquiera de estas *entrada/salidas* pueda ser redirigida a un fichero. De esa manera la entrada a un comando puede haber sido almacenada previamente; o podemos guardar la salida de un comando en un fichero para su posterior análisis.

En todo caso la caraterística de la interfaz de comandos que nos interesa es aquella que permite redirigir la *salida estándar* de un comando a la *entrada estándar* de otro.

```
[cila@gulic]$ cat pepe.txt | sort
[cila@gulic]$ ls -l | more
```

En el ejemplo anterior utilizamos el *metacarácter* `|` para redirigir la salida del comando `cat` (es decir, el contenido del fichero `pepe.txt`) a la entrada del comando `sort`. El comando `sort` toma las líneas de texto que provienen de la *entrada estándar*, las ordena, y las muestra por su *salida estándar*. Por lo tanto, el ejemplo muestra por pantalla el contenido de `pepe.txt` ordenado alfabéticamente.

En el mismo ejemplo redirigimos la salida de `ls` al comando `more`. El comando `more` resulta muy útil cuando el contenido de un fichero o la salida de un comando es lo suficientemente grande como para no caber completamente en la pantalla. En esos casos `more` nos permite ver el texto pantalla a pantalla, utilizando la barra espaciadora para avanzar por el mismo.

Un comando mucho más potente pero con una utilidad similar a `more` es `less`.

```
[cila@gulic]$ ls -l | less
```

El comando `less` nos permite utilizar las teclas del cursor, `Re. Pag.` y `Av. Pag.` para avanzar y retroceder por el texto. También podemos iniciar una búsqueda, o continuar una ya iniciada, con las teclas `/` y `n` respectivamente. Muchas son las características de `less`, aunque nos conformaremos con saber que con `h` podemos consultar la ayuda del comando mientras que con `q` salimos del mismo.

En general los comandos que hemos estudiado son programas de consola como otros cualesquiera. Por lo tanto pueden ser llamados directamente, e incluso en muchos casos disponen de opciones de línea de comandos.

```
[cila@gulic]$ sort -r pepe.txt
[cila@gulic]$ less pepe.txt
```

2.3.3. El comando `man`

El comando `man` es muy útil, ya que nos dará mucha información sobre la mayoría de los comandos con los que vamos a trabajar.

```
[cila@gulic]$ man bash
```

Omitimos la información de salida ya que puede ser muy extensa e invita al lector a que lo pruebe él mismo.

En cualquier caso, en el tema 6 estudiaremos este comando con más detenimiento.

2.3.4. El comando gzip

El comando `gzip nombre_fichero` comprime un fichero utilizando el algoritmo Lempel-Ziv.

```
[cila@gulic]$ gzip pepe.txt
búsqueda
```

Por regla general el fichero desaparece y en su lugar se crea otro comprimido y con el mismo nombre más el sufijo `.gz`.

La descompresión se realiza utilizando la opción `-d`. Es importante destacar que, al igual que antes, el archivo comprimido desaparece para dejar en su lugar la versión descomprimida.

```
[cila@gulic]$ gzip -d pepe.txt.gz
```

Debido a la incomodidad de tener que comprimir/descomprimir para poder acceder a la información son muchos los comandos que cuentan con versiones especialmente diseñadas para manipular archivos comprimidos directamente. Es el caso de `zless`, `zgrep`, `zcat`, `zmore`, etc...

Pese a lo extendido del uso de `gzip` en la actualidad hay muchos otros algoritmos con ratios de compresión mayores. Por ello se han creado comandos compatibles, en cuanto a opciones de la línea de comandos, con `gzip`, pero que implementan esos otros algoritmos. Es el caso de `bzip2` y de los comandos `bzless`, `bzgrep`, `bzcat`, `bzmore`, etc...

2.3.5. El comando tar

El comando `tar` permite la manipulación de *ficheros de archivo* en formato TAR. Dichos ficheros están diseñados para almacenar uno o más ficheros y/o directorios y toda la información relacionada con los mismos. Entre esa información se encuentran las fechas de acceso y modificación, los permisos, el propietario, el grupo, etc. El origen del comando `tar` se remonta al uso de dipositivos sencuenciales (como por ejemplo cintas magnéticas) para almacenar copias de seguridad de los archivos del sistema. En la actualidad no es sino una forma de empaquetar en un único archivo pedazos concretos del árbol de directorios de nuestro sistema.

Para crear un archivo TAR basta con que utilicemos la opción `-c` seguida por la lista de ficheros y/o directorios que queremos empaquetar. En general el archivo TAR resultante se vuelca a la *salida estándar*. Puesto que eso resulta poco práctico es habitual utilizar la opción `-f` seguida del nombre del fichero de destino.

```
[cila@gulic]$ tar -cf fichero.tar pepe.txt juan.txt /bin
```

En el ejemplo anterior `fichero.tar` almacena tanto los ficheros `pepe.txt` y `juan.txt` como el contenido del directorio `/bin`.

Los datos almacenados en los ficheros TAR no están comprimidos. Por ello es habitual utilizar las opciones `-z` o `-j` para que `tar` pase el fichero por `gzip` o `bzip2` respectivamente, cuando termine de empaquetar los ficheros.

```
[cila@gulic]$ tar -zcf fichero.tar.gz pepe.txt /bin
```

Para recuperar los ficheros originales se sustituye la opción `-c` por `-x`, mientras que para verificar la integridad del archivo sin tener que desempaquetarlo se utiliza la opción `-t`.

```
[cila@gulic]$ tar -xf fichero.tar
[cila@gulic]$ tar -zxf fichero.tar.gz
[cila@gulic]$ tar -ztf fichero.tar.gz
```

2.3.6. El comando grep

Cuando deseamos localizar un cadena de texto dentro de uno o varios ficheros solemos recurrir al comando `grep`. Dicho comando requiere que se especifique la cadena de texto a buscar seguida del nombre de los ficheros en los que realizar la búsqueda. Puesto que alguno de los caracteres de la cadena de texto a buscar pueden tener algún significado especial para la interfaz de comandos dicha cadena suele ir entrecomillada.

```
[cila@gulic]$ grep 'cila' *
[cila@gulic]$ grep 'CaSa' pepe.txt
```

Como se puede observar, en el primer ejemplo estamos buscando la cadena de texto `'cila'` dentro de todos los ficheros del directorio actual.

En general `grep` es sensible a mayúsculas y minúsculas. Si queremos eliminar dicho comportamiento podemos emplear la opción `-i`. También podemos pedir que nos indique dónde **NO** está la cadena indicada utilizando la opción `-v`. Por otro lado las búsquedas pueden extenderse por directorios y subdirectorios utilizando la opción `-r`.

En realidad `grep` no sólo nos permite utilizar simples cadenas para buscar dentro de los ficheros indicados. El comandos `grep` nos permite utilizar *expresiones regulares*. Es decir, dentro de nuestra cadena de texto podemos utilizar caracteres con un significado especial con los que podemos buscar casi cualquier tipo de expresión entre nuestros archivos. Puesto que las expresiones regulares se salen completamente del alcance de este tema recomendamos consultar el tema correspondiente.

```
[cila@gulic]$ man grep
```

2.3.7. Otros comandos

clear Limpia la pantalla de la consola (tecla `C-l`)

locate Es la orden de búsqueda más rápida y sencilla para localizar un archivo.

reset Si observamos que escribimos en pantalla y no aparece el texto pero al pulsar `Enter` realmente se está escribiendo, o que los colores o los textos de la consola se corrompen, puede ser que alguna aplicación en modo texto haya finalizado bruscamente no restaurando los valores estándar de la consola al salir. Con esto forzamos unos valores por defecto, regenerando la pantalla.

top Muestra los procesos que se ejecutan en el momento actual, informando de los recursos que se están consumiendo.

whoami El curioso nombre de este comando proviene *Who am I?* (*¿Quién soy?*), que indica que este comando es capaz de informarnos del nombre de usuario con que se entró en esa consola. Puede parecer una tontería, pero si una persona entra en dos sesiones, en una como `root` y en otra como usuario normal, si no se sabe quién es en ese momento, podrían ocurrir accidentes catastróficos.

2.4. Unidades de disco

Como hemos dicho anteriormente, en Linux no existen las unidades como `A:` o `C:`. Para acceder a un disco es necesario primero *montarlo*, esto es asignarle un lugar dentro del árbol de directorios del sistema.

Por ejemplo, podemos asignar a la disquetera el directorio `/floppy`, al CD-ROM el directorio `/cdrom` o a la grabadora de CDs el directorio `/grabata`. Normalmente los directorios para la disquetera y el lector de CD-ROM están ya asignados desde el momento de instalar el sistema, aunque se puede cambiar a voluntad (si somos `root`).

Para montar un disco utilizamos el comando `mount` indicándole como parámetros el dispositivo al que queremos acceder, el directorio en el que lo queremos montar, y el sistema de archivos utilizado para ordenar la información. Sin embargo, el administrador (`root`) suele indicar todos estos parámetros en el fichero `fichero`, y los usuarios tienen que conformarse montar lo que `root` les permita. En nuestro caso el señor `root` ha determinado que los usuarios sólo podemos montar la disquetera en el directorio `/floppy`.

```
[cila@gulic]$ mount /dev/fd0 /floppy
mount: only root can do that
[cila@gulic]$ mount /floppy
```

Una vez montado el disquete en el directorio `/floppy` ya podemos acceder y manipular sus ficheros y subdirectorios como más nos convenga. Desde la perspectiva del usuario no hay ninguna diferencia entre trabajar en ese directorio y trabajar en otro cualquiera.

Pobre de quien saque el disquete sin desmontarlo. ¿Por qué? Pues por tres razones:

1. Existe el riesgo de que perdamos la información que hayamos grabado en el disquete.
2. Ningún otro usuario podrá usar la disquetera, al menos hasta que se reinicie el ordenador o el `root` tenga tiempo para forzar el que la unidad sea desmontanda, lo cual no le hará gracia a nadie.
3. El Sr. root puede mosquearse con quien lo haga, y a nadie le conviene tener mosqueado al Sr. root.

Para desmontar el disquete simplemente utilizamos el sencillo comando `umount`:

```
$ umount /floppy
```

Como último ejemplo, hagamos lo siguiente:

```
$ mount /floppy
$ cd /floppy
$ umount /floppy
```

¿Verdad que no funciona? Esto se debe a que en el momento de desmontar la disketera, **no debe haber NADIE dentro** de ese directorio. Recordemos que estamos en un sistema multiusuario y puede ocurrir que más de una persona acceda a la disquetera o a otro dispositivo desmontable, como el CD-ROM. Por tanto, hemos de asegurarnos que no hay nadie.

Para comprobar en un momento dado si el disquete está montado podemos usar el comando `df`, que nos informa sobre los *sistemas de ficheros* que están montados y su estado de almacenamiento. La opción `-h` nos muestra las cantidades en cifras *humanas*.

```
[cila@gulic]$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda2             1.9G  1.8G   80M  96% /
euler:/home            20G  3.2G   16G  17% /home
euler:/usr/soft       3.9G  5.4M  3.7G   0% /usr/soft
/dev/fd0              1.4M  758k  665k  53% /floppy
```

Aquí vemos que la segunda partición del primer disco duro `/dev/hda2` está montada en el directorio `/`, los directorios `/home` y `/usr/soft` del servidor `euler` están montados en sus equivalente locales, y el disquete `/dev/fd0` está montado en el directorio `/floppy`.

2.5. Unidades de disco con mtools

En general, la mayor parte de los disquetes que usamos están en formato DOS/Windows. Cuando disponemos de uno esos disquetes podemos acceder a su contenido de forma sencilla con las mtools.

mtools es una colección de herramientas de dominio público que permite a sistemas UNIX manipular ficheros en un sistema de archivos DOS/Windows (típicamente un disquete). mtools es suficiente para dar acceso a sistemas de archivos DOS/Windows. Por ejemplo, órdenes como `mdir a:` funcionan en el disquete `A:` sin ningún montaje preliminar ni otro procedimiento de inicio (suponiendo que el `/etc/mtools.conf` sea correcto). Gracias a todo ello, con mtools se puede cambiar de disquete sin tener que desmontar y montar.

2.5.1. Nombres de ficheros

Muchas de las herramientas de mtools requieren que se especifiquen nombres de archivo en el sistema de archivos DOS/Windows al que estamos accediendo. Las rutas de ficheros en el sistema de archivos DOS/Windows se componen de: una *letra de dispositivo* seguida de dos puntos, un subdirectorio y un nombre de fichero. Los nombres de directorio pueden emplear como separador / o \. El uso del separador \ requiere que los nombres se entrecomillen para protegerlos, por lo que recomendamos utilizar / con el fin de evitar problemas. Los nombres de ficheros que no van precedidos de una letra de dispositivo se consideran ficheros del sistema UNIX.

Por ejemplo, el siguiente comando copia el archivo `TEST1.TXT` desde el directorio `TEST` del primer dispositivo de disquete (`A:`), hasta nuestro directorio de trabajo actual, con el nombre de archivo `test2.txt`.

```
$ mcopy 'a:\TEST\TEST1.TXT' test2.txt
```

que es completamente equivalente a:

```
$ mcopy a:/TEST/TEST1.TXT test2.txt
```

Como es obvio las mtools no distinguen entre mayúsculas y minúsculas en el acceso al sistema de archivos DOS/Windows. Sin embargo, si permiten la utilización de comodines (como * o ?) tanto cuando especificamos un nombre de archivo Linux, como cuando hacemos lo mismo con nombres de archivo DOS/Windows.

En cuanto a las letras de dispositivo, comúnmente la unidad `A:` es la primera unidad de disquete, la unidad `B:` es la segunda unidad de disquete, la unidad `J:` es una unidad *Jaz* y la unidad `Z:` es una unidad *Zip*. Sin embargo todo esto puede configurarse mediante el fichero de configuración `/etc/mtools.conf`.

2.5.2. Lista de comandos

A continuación presentamos algunos de los comandos más utilizados de las mtools.

2.5.2.1. mattrib

Se emplea para cambiar los atributos de ficheros DOS/Windows de forma semejante a como lo hace el comando `ATTRIB` del MS-DOS.

2.5.2.2. mcd

El comando `mcd` se emplea para cambiar el directorio de trabajo actual de mtools en los discos DOS/Windows.

```
$ mcd <directorio_dos>
```

Sin argumentos, `mcd` informa de la unidad y directorio de trabajo actuales. De otra forma, mcd cambia la unidad en curso y el directorio de trabajo relativos a un sistema de archivos DOS/Windows.

Por ejemplo, si ejecutamos la siguiente secuencia de comandos:

```
$ mcd a:/TEST
$ mcopy a:TEST1.TXT test2.txt
```

Copiaríamos el archivo `TEST1.TXT` tal y como lo hicimos anteriormente (página 22).

A diferencia de los sistemas DOS/Windows, con mtools sólo hay un directorio de trabajo actual para todas las unidades, y no un directorio de trabajo diferente para cada unidad.

2.5.2.3. mcopy

El comando `mcopy` permite copiar ficheros desde o hacia sistemas de archivos DOS/Windows. Las formas de uso son:

```
$ mcopy fichero_fuente fichero_destino
$ mcopy fichero_fuente <fichero_fuente> directorio_destino
$ mcopy fichero_fuente_dos
```

El comando copia el `fichero_fuente` al `fichero_destino`, o copia múltiples ficheros al directorio de destino indicado. Fuente y destino pueden ser ficheros de DOS o de Linux. La presencia, o no, del indicador de letra del dispositivo es el que determina qué ficheros son de DOS y cuáles de Linux.

Si sólo se suministra uno de los parámetros fuente, se supone como destino el directorio actual de trabajo en el sistema Linux.

2.5.2.4. mdel

El comando `mdel` se emplea para borrar ficheros.

```
$ mdel fichero_dos
```

Por ejemplo:

```
$ mdel a:/TEST2.TXT
```

2.5.2.5. mdeltree

El comando `mdeltree` se utiliza para borrar un directorio DOS/Windows y todos sus archivos y subdirectorios.

```
$ mdeltree directorio_dos
```

2.5.2.6. mdir

El comando `mdir` se emplea para mostrar el contenido de un directorio DOS/Windows.

```
$ mdir <directorio_dos>
```

Por ejemplo:

```
$ mdir a:
 Volume in drive A has no label
 Volume Serial Number is 3E48-13E9
Directory for A:/

TRABAJO  TXT     13838 11-01-1993   3:11
PROGRA~1 EXE    268232 12-14-2002  22:15  programinstall.exe
TEST         <DIR>     12-14-2002  22:13
        3 files             282 070 bytes
                          1 174 528 bytes free
```

2.5.2.7. mformat

El comando `mformat` se utiliza para crear un sistema de archivos DOS/Windows vacio en la unidad indicada.

```
$ mformat <unidad:>
```

2.5.2.8. mmd

El comando `mmd` se emplea para crear un nuevo directorio en un sistema de archivos DOS. El comando informará de un error si el directorio ya existe.

```
$ mmd directorio_dos
```

2.5.2.9. mmove

El comando `mmove` se utiliza para mover o renombra un fichero o subdirectorio existente en un sistema de archivos DOS/Windows. Las formas de uso son:

```
$ mmove fichero_fuente fichero_destino
$ mmove fichero_fuente <fichero_fuente> directorio_destino
$ mmove directorio_fuente directorio_destino
```

2.5.2.10. mrd

El comando `mrd` se emplea para borrar un directorio de un sistema de archivos DOS/Windows. El comando volverá con un error si el directorio no existe o no está vacío.

```
$ mrd directorio_dos
```

2.5.2.11. mtype

El comando `mtype` muestra el fichero DOS/Windows especificado, en la pantalla o en la salida estándar.

```
$ mtype fichero_dos
```

Con este capítulo quedan explicados los rudimentos para saber moverse por Linux.
¡Feliz exploración!

TEMA 3

El entorno X-Window

3.1. ¿Qué es X-Window?

En algunos sistemas operativos el entorno gráfico, también conocido como *GUI (Graphic User Interface)* es una parte inherente al sistema. En dichos casos los diseñadores no concibieron el funcionamiento de la máquina en situaciones donde el GUI sea inútil o implique un desperdicio innecesario de recursos. En esos sistemas el entorno gráfico no sólo no suele ser sustituible sino que sencillamente el sistema operativo no puede funcionar sin él. La mayor parte de los sistemas operativos que cumplen esta descripción son sistemas jóvenes, desarrollados recientemente, con la clara convicción de que el entorno gráfico era la gran panacea. El tiempo ha demostrado que estaban parcialmente equivocados. Disponer de GUI facilita el acceso del gran público a la informática, pero resta mucha flexibilidad a los profesionales o usuarios avanzados, cuyas necesidades siempre van un paso por delante de las soluciones aportadas por el GUI. Prueba de ello es que algunos sistemas netamente gráficos han comenzado a cuidar su interfaz de comandos con el objetivo de poder introducirse en el segmento de los servidores y estaciones de trabajo profesionales.

La bases de los sistemas operativos UNIX actuales se establecieron por Bell Laboratories en torno a 1970. En aquella época los ordenadores no contaban con otra cosa que no fuera una consola en modo texto. Dichos sistemas han evolucionado a lo largo del tiempo incorporando las correspondientes innovaciones tecnológicas debidas a la implantación del modo gráfico por los ordenadores de todo el mundo. A causa de dicho proceso de evolución, en Linux el entorno gráfico es una aplicación más de las muchas que pueden estar ejecutándose, o no, en el sistema. Por lo tanto puede ser sustituido según nuestra conveniencia.

El entorno gráfico más ampliamente extendido en el mundo UNIX/Linux son las X-Window, conocidas comúnmente como X a secas.

3.1.1. La arquitectura de las X

Las X presentan una arquitectura (Figura 3.1) cliente/servidor que en muchos casos ha sido alabada pero en otros muchos criticada.

La arquitectura de las X se divide en dos elementos fundamentales. Por un lado el *servidor X* encargado de gestionar el uso de los dispositivos de salida (p. ej. tarjetas gráficas) y de los dispositivos de entrada (p. ej. teclados, ratones, tabletas digitalizadoras, etc.) Por el otro lado, los *clientes X* nombre con el que se conoce a cada una de las aplicaciones que hacen uso del sistema X. Las pulsaciones de tecla, el movimiento del ratón o cualquier otra acción del usuario en los dispositivos de entrada es detectada por servidor y transferida a los clientes. De la misma manera los clientes transfieren al servidor las peticiones de operaciones a realizar sobre el dispositivo de salida (p. ej. trazar un línea en pantalla, dibujar un punto, volcar un bitmap, etc). La comunicación entre cliente y servidor sigue un protocolo estandarizado que corre sobre los servicios de red del sistema. Esto permite que los clientes no tengan por qué estar en la misma máquina que el servidor. Eso quiere decir que podemos tener un servidor X en nuestra máquina local a través del que interactuar con clientes que se ejecutan de forma remota en otro ordenador, sin que por ello notemos diferencia alguna. Aparte de todo esto las X realizan ciertas optimizaciones en los casos en los que cliente y servidor se encuentran en la misma máquina. Por ejemplo, se permite el acceso directo al hardware de vídeo ignorando el protocolo descrito, con lo que se aprovechan las características de aceleración 3D de muchas tarjetas modernas. Con ello las X se convierten en una plataforma potente para ejecutar aplicaciones exigentes desde un punto de vista gráfico sin por ello perder su flexibilidad.

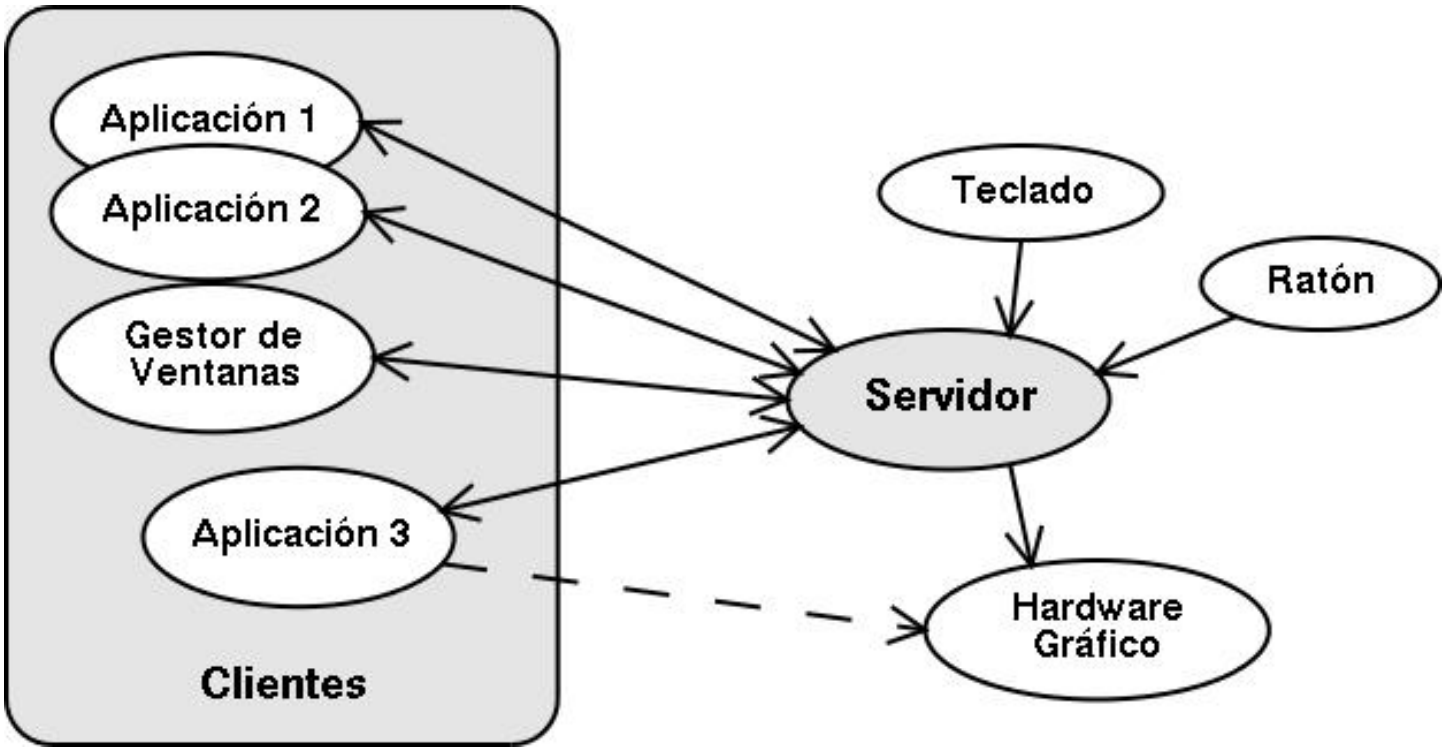


Figura 3.1: Arquitectura cliente/servidor del sistema X-Window

Otra característica interesante del sistema X es que podemos tener varios servidores en una misma máquina, donde cada uno es un *ente* independiente que utiliza sus propios dispositivos de entrada y salida. Evidentemente no es habitual disponer de más de un monitor o teclado pero en ocasiones se pueden utilizar servidores sin salida gráfica. También es posible que todos usen los mismos dispositivos y que se habilite algún mecanismo para que podamos conmutar entre ellos a voluntad.

3.1.2. Gestores de ventanas

En todo buen entorno gráfico moderno las aplicaciones utilizan ventanas para interactuar con el usuario. Sin embargo el servidor X sólo entiende un número limitado de primitivas gráficas, como dibujar líneas y puntos o copiar áreas de la pantalla. Por ello se hace necesaria la presencia de una aplicación cliente *especial* que se encargue de crear, destruir, mover, gestionar el foco y en general gestionar todas las cuestiones referentes al comportamiento de las ventanas, utilizando para ello las primitivas del servidor X. A dicha aplicación se la denomina *gestor de ventanas*. En la Figura 3.1 podemos ver su situación dentro de la arquitectura de las X-Window.

En una distribución de Linux puede haber cerca de 40 gestores de ventanas entre los que un usuario debe elegir. Cada uno de ellos imprime un *feeling* o sabor diferente a nuestro entorno de trabajo. Tengamos en cuenta que cada gestor dibuja los elementos del marco de nuestras ventanas de forma diferente (e.j. la barra de título, los botones de control, los bordes), dotándolos de un comportamiento particular y característico según las preferencias del equipo de programadores que lo diseñó. Empezando por los estéticamente más valorados como Window Maker o Enlightenment; pasando por los que imitan a otros sistemas como AfterStep o F(?) Virtual Window Manager (en una de sus variantes nos proporciona el look de Microsoft® Windows® 95); o los que permiten ser personalizados con *temas* diferentes como Ice Window Manager o Sawfish; y terminando por los más ligeros y rápidos pero no menos funcionales como Fast Light Window Manager se cubre toda una variedad de necesidades del usuario.

3.1.3. Entornos de escritorio

Ahora que disponemos de ventanas se hace necesario rellenarlas con algo. A la hora de programar una aplicación para las X se suele recurrir a los *toolkits*. Se trata de librerías diseñadas para proporcionar diferentes tipos de controles (p. ej. botones, barras de menús, cuadros de edición, etc) facilitando su gestión. Las toolkits crean los controles allí dónde le digamos, los

destruyen, los redibujan cuando es necesario, manejan todas las acciones que se hagan sobre ellos a través del uso de alguno de los dispositivos de entrada posibles, etc. Trabajar sin hacer uso de los toolkits implicaría diseñar y gestionar nuestros propios controles.

Es importante destacar que una aplicación X funciona sea cual sea el gestor de ventanas seleccionado. Por tanto, el uso de uno u otro es una elección personal del usuario. Sin embargo, el uso de un toolkit u otro en una aplicación es elección del equipo de programadores que ha trabajado en ella. Eso unido a la gran variedad de toolkits existente en Linux convierte nuestro escritorio en una selva donde podemos ver una fauna de aplicaciones con interfaces gráficas de lo más variopinto.

Para garantizar que las aplicaciones presenten una interfaz similar, reduciendo la curva de aprendizaje de los usuarios, han aparecido los *entornos de escritorio*. Básicamente se trata de establecer una serie de reglas comunes que suelen incluir: el toolkit a utilizar, el formato de la ayuda, el modelo de componentes, librerías de tratamiento de imágenes y sonido, etc. Todo ello genera un marco de trabajo tanto para los desarrolladores de nuevas como de viejas aplicaciones. Si los desarrolladores se ciñen a dicho marco el resultado es un entorno de trabajo uniforme y cómodo donde no existen diferencias sustanciales en la interfaz de una aplicación a otra.

Afortunadamente la variedad es una característica de Linux. En la actualidad disponemos de dos grandes entornos de escritorio:

GNOME Funciona con el toolkit GTK que fue originalmente diseñado para GIMP. Quizás las principales ventajas de este entorno no estén tanto del lado del usuario como del de los programadores, de la tecnología que subyace debajo del sistema y de la declaración de intenciones con la que se inició el proyecto.

KDE Funciona sobre el toolkit Qt, lo cual provocó una gran polémica en sus orígenes al no disponer de una licencia todo lo *libre* que se deseaba. En la actualidad esos problemas se han resuelto y es, hoy por hoy, considerado por muchos como el entorno con la interfaz más atractiva de los dos.

Cada uno tiene sus más y sus menos pero la gran realidad es que presentan una interfaz muy intuitiva que se aprende a manejar desde el primer momento.

3.2. Trabajar con las X-Window

Al iniciar un sistema Linux con las X instaladas es probable que se nos permita autentificarnos desde el modo gráfico. En caso contrario sólo dispondremos de una consola esperando a que introduzcamos el nombre de usuario. Si nuestro caso es este último debemos autentificarnos, y ejecutar el comando `startx` cuando el sistema nos indique que está disponible para recibir nuestros comandos. El comando `startx` iniciará el entorno gráfico. Evidentemente al terminar nuestro trabajo volveremos a la consola.

Para que la autentificación desde el modo gráfico sea posible es necesario tener instalado un *display manager*. Algunos de ellos son `xdm`, `kdm` (KDE), `gdm` (GNOME), etc. En la Figura 3.2 podemos ver un ejemplo de la clásica ventana que nos muestra GDM nada más terminar la carga del sistema. Básicamente nos está pidiendo nuestro nombre de usuario, para una vez pulsado `Enter` pedirnos la contraseña. Con esa información seremos autentificados en el sistema y se iniciará una nueva sesión con nuestra cuenta de usuario. Es importante destacar que dentro de nuestra sesión todo lo que hagamos es personal. Eso quiere decir que los cambios, sean del tipo que sean, sólo afectarán a nuestra sesiones futuras pero no a las de los otros usuarios de la máquina.

Antes de continuar es importante destacar algunas cosas. En Linux existe lo que se denomina *terminales virtuales*. Si estamos en modo texto podemos usar la combinación de teclas `A-F*` para pasar de un terminal a otro. Eso se comprueba fácilmente porque vemos cómo cambia el contenido de la pantalla. El uso de los terminales virtuales nos permite mantener varias sesiones abiertas y trabajar de forma independiente en ellas. Cuando se inicia un servidor X éste, queda asociado a uno de esos terminales virtuales (podemos tener varios servidores asociados a distintos terminales); normalmente al `F7`. Debido al mecanismo por el que las X manejan el teclado, en los terminales asociados a servidores X ya no se puede usar `A-F*` sino que para cambiar de terminal debemos usar `C-A-F*`. Usando la primera combinación de teclas en los terminales en modo texto, y la última en los asociados a las X, no tendremos problemas para movernos por ellos y pasar de modo gráfico a consola a nuestro antojo.

Otro punto importante es que sólo cuando veamos un mensaje del tipo `Kernel Panic` podemos decir que nuestro Linux se ha colgado. En cualquier otro caso estamos frente a sencillos cuelgues de las aplicaciones con las que trabajamos. No debemos preocuparnos puesto que Linux proporciona los suficientes recursos como para que podamos recuperar el control de la máquina. Por ejemplo, podemos cambiar a un terminal diferente de la consola para matar aplicaciones que se han quedado

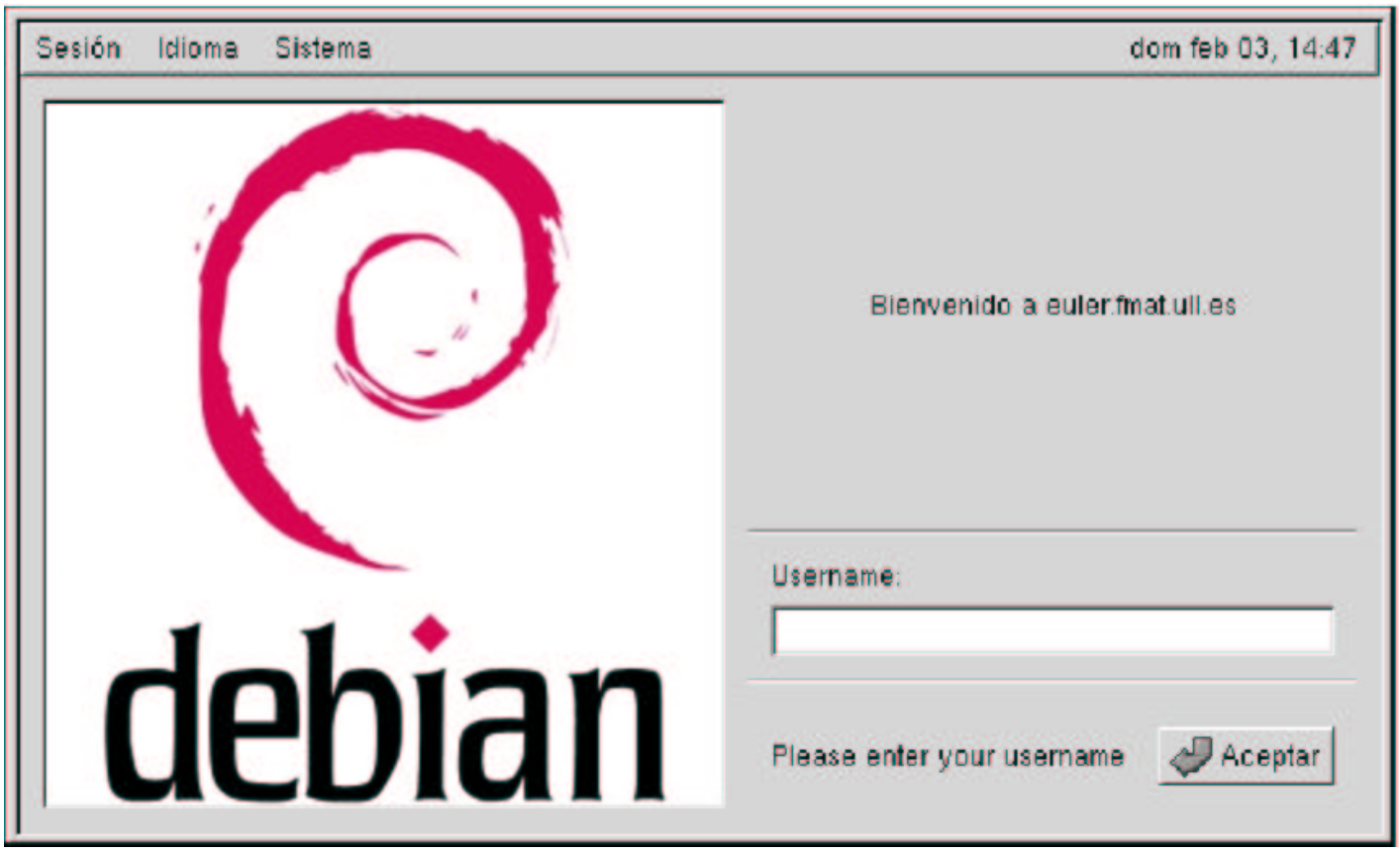


Figura 3.2: Ventana de autentificación de GDM

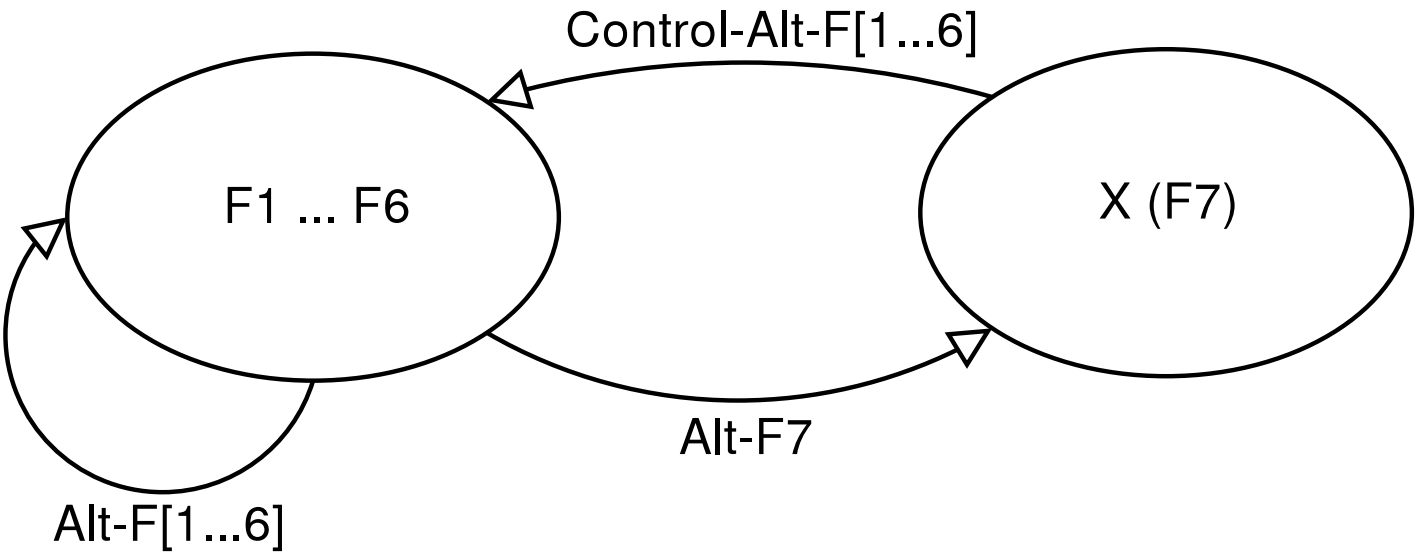


Figura 3.3: Diagrama de cómo se pasa de una terminal a otra.

colgadas en nuestro terminal de trabajo o en las X . En situaciones críticas puede ser necesario terminar con las X de forma prematura. Para ello se pulsa la secuencia `C-A-Backspace` que cierra el servidor X abortando todas las aplicaciones gráficas. Evidentemente los datos no guardados se perderán.

En Linux, cuando queremos ejecutar aplicaciones de modo consola dentro de una ventana X se utilizan los *emuladores de terminal*. El más básico es el `xterm` que viene con la instalación estándar de las X, pero existen muchos otros y cada entorno suele venir con uno propio. El de GNOME está en `MENÚ PRINCIPAL → SISTEMA → TERMINAL UNIX DE GNOME`, mientras que al de KDE podemos acceder pulsando en el icono con un terminal y una concha disponible en el panel de dicho entorno de escritorio (Figura 3.16). En ocasiones es posible que intentemos ejecutar una aplicación que nunca llega a mostrar su ventana principal o que da algún tipo de error grave. En esos casos es recomendable ejecutar el programa desde un emulador de terminal. Como ya hemos comentado, las aplicaciones para X no son diferentes de otras, por lo que suelen mostrar información por la consola, si esta está disponible. Esa información puede ser vital para resolver nuestro problema.

Cuando ejecutamos una aplicación X desde el emulador de terminal (p. ej. `xclock`) vemos que éste se queda bloqueado a la espera de que la aplicación termine. Esto no debería sorprendernos puesto que en realidad ocurre lo mismo cuando ejecutamos una aplicación de consola. Para que eso no suceda debemos añadir un `&` al final del comando de la aplicación X, o pulsar `C-Z` y ejecutar el comando `bg` sobre el emulador después de que la aplicación haya sido iniciada. De esta manera la aplicación X pasa a ejecutarse en segundo plano, liberando al emulador de terminal para que pueda recibir nuevos comandos.

En general mostrar el clásico aviso de que una aplicación tiene archivos modificados y va a ser cerrada es responsabilidad de la propia aplicación. El cierre de la ventana principal por la pulsación del correspondiente botón en la barra de título es notificado al proceso propietario de la ventana. Normalmente se suele mostrar dicho mensaje y una vez aceptado se termina el proceso. Sin embargo es posible que nuestra aplicación no disponga de esa característica. Por ejemplo, si cerramos el emulador de terminal (muy pocos avisan de la posible pérdida de datos) cuando estamos ejecutando sobre él cualquier tipo de aplicación (sea de consola o de X) dicha aplicación termina inmediatamente perdiéndose los datos que no hubieran sido guardados. O sea, si ejecutamos `gnome-edit` o `kedit` sobre el terminal, escribimos algo en el editor, y cerramos el terminal, veremos como nuestra aplicación de edición terminará sin advertir nada de la pérdida de datos, y el terminal se cerrará sin indicarnos que se encuentra esperando por el editor.

Figura 3.4: Barra de título del gestor de ventanas Sawfish

Por otro lado es posible que el proceso esté bloqueado y no reciba el mensaje. En ese caso la ventana no se cerrará a menos que forcemos su destrucción con la opción correspondiente en el menú del marco de la venta (Figura 3.4, botón de la izquierda). Entonces el gestor de ventanas la cerrará pero la aplicación seguirá ejecutándose en segundo plano aunque para nosotros ya no exista. La única solución será matarla a mano con el comando `kill` desde un emulador de terminal. Si vemos un consumo de CPU excesivo o un sistema demasiado lento puede deberse a la existencia de procesos en segundo plano que no fueron destruidos en su momento.

Una aplicación interesante es `xkill`. Si la ejecutamos nos aparecerá un punto de mira en el cursor de nuestro ratón. Al pulsar sobre una ventana la aplicación se encarga de destruir al proceso asociado a la misma. La aplicación `xkill` puede ser lanzada desde un emulador de terminal o desde los cuadros de diálogo de las opciones `LANZAR` o `EJECUTAR` de los menús principales de los distintos entornos de escritorio.

Como hemos dicho anteriormente, a la hora de aprender a usar las X no basta con jugar con los botones izquierdo y derecho de nuestro ratón. En las X el botón central es de tanta importancia que suele disponer de funciones adicionales que no están presentes en los otros dos. En entornos de escritorio como KDE o GNOME esto suele implicar un menú de contexto adicional al que se muestra usando el botón derecho; pero en otras aplicaciones podemos encontrar formas de manejo de lo más curiosas. Tal es su importancia que en caso de no disponer de él se emula por la pulsación simultanea de los dos botones estándar del ratón. Una aplicación interesante del botón central es su uso para copiar texto entre aplicaciones, sea cual sea la toolkit bajo las que hayan sido desarrolladas. Basta con marcar nuestro texto en la aplicación de origen para que al pulsar el botón central en la de destino éste se pegue a continuación de la posición actual del cursor. En el servidor X existe un búffer que se llena cada vez que marcamos texto en alguna aplicación. El botón central del ratón lo único que hace es volcar ese texto como si hubiera sido escrito desde el teclado. Por tanto para utilizar este sistema es importante ser lo suficientemente cuidadoso como para no marcar otra cosa después de marcar el texto a copiar.

3.3. El escritorio de GNOME

Debido a la sencillez de los entornos de escritorio resulta inútil intentar explicar detalladamente su manejo. La mejor forma de aprender es sentarnos delante de uno de ellos, ser un poco osado, y ante todo tener la buena costumbre de leer detenidamente todo aquello que nos indique el sistema. Sin embargo, vamos a realizar una primera toma de contacto guiada en la que aprenderemos algunos conceptos básicos que nos serán de gran ayuda en el futuro. Para empezar nos centraremos en uno de los dos entornos de escritorio que hemos comentado anteriormente. Concretamente echaremos un vistazo al entorno de escritorio GNOME. Sin embargo, los devotos de KDE no deben preocuparse puesto que conociendo uno resulta extremadamente sencillo llegar a dominar el otro. Aun así, al final del capitulo, comentaremos algunos aspectos de interés sobre el fantástico entorno de escritorio de KDE.

3.3.1. GNOME Display Manager

Si nuestro sistema tiene instalado el entorno de escritorio GNOME y permite la autentificación desde el modo gráfico es probable que tengamos instalado el GDM (GNOME Display Manager) como *display manager*(Figura 3.2). GDM dispone de algunas opciones adicionales que podemos emplear antes de iniciar el proceso de autentificación, y que no tienen porque estar presentes en otros display manager.

SESIÓN Nos permite elegir la naturaleza de la próxima sesión a iniciar. Opciones habituales son GNOME y KDE. Eligiéndolas iniciaremos sesión con el entorno de escritorio que prefiramos. También puede haber alguna sesión especial *a prueba de fallos*, o que carezca de un entorno de escritorio específico, o cualquier otro tipo de sesión que el administrador del sistema haya deseado incluir. La opción por defecto es `ÚLTIMA` que hace referencia a entrar con el tipo de sesión que tenemos asignado nosotros por defecto. Si nunca hemos elegido ninguno lo normal es que el tipo por defecto sea GNOME. Si hacemos una selección diferente a la opción por defecto el sistema nos preguntara si queremos que el nuevo tipo de sesión sea nuestra sesión por defecto, y por tanto a la que que corresponderá la opción `ÚLTIMA` la próxima vez.

IDIOMA Aparte de elegir sesión podemos seleccionar el idioma con el que queremos trabajar. Las aplicaciones traducidas mostrarán toda la información en dicho idioma. Igual que antes, la opción `ÚLTIMA` habla de nuestra última selección, es decir el idioma por defecto.

SISTEMA Nos permite apagar, suspender o reiniciar la máquina. En algunos sistemas, y siempre y cuando dispongamos de la contraseña de la cuenta del `root`, nos permite configurar GDM. Es importante recordar que al cerrar nuestra sesión volvemos a la ventana de GDM por lo que si deseamos apagar el sistema éste sería el momento de seleccionar la opción correspondiente.

3.3.2. Paneles y menús de GNOME

Suponiendo que iniciamos sesión en la máquina lo primero que nos llamará la atención son los *paneles* (Figuras 3.5 y 3.6).

Figura 3.5: Menú de panel de GNOME

Si pulsamos en la huella del panel inferior, o seleccionamos alguna de las opciones del panel superior, dispondremos de acceso a las aplicaciones instaladas en el sistema. La mayor parte de las aplicaciones GNOME están clasificadas y disponibles en alguno de los submenús correspondientes. (p. ej. `APLICACIONES`, `UTILIDADES`, `GRÁFICOS`, etc). En `MENÚS KDE` se encuentran clasificadas todas las aplicaciones KDE instaladas, y en `MENÚS DEBIAN` todas las aplicaciones de la distribución, independientemente del entorno de escritorio para las que fueron desarrolladas. En este punto es importante recordar que podemos ejecutar cualquiera de las aplicaciones sin importar que estemos trabajando en un entorno de escritorio diferente a aquel para el que fueron diseñadas.

Otras opciones son el menú `FAVORITOS`, que es personalizable con nuestras aplicaciones preferidas usando la opción `AÑADIR DESDE EL MENÚ` (Figura 3.7), la opción `BLOQUEAR LA PANTALLA`, para proteger nuestro escritorio si tenemos que abandonarlo por cualquier motivo (es importante recordar que de esa manera evitaremos que manipulen nuestros datos pero siempre queda la posibilidad de que pulsen `C-A-Backspace` para abortar todas las aplicaciones gráficas, perdiendo los datos no guardados y volviendo a la ventana de inicio de sesión del GDM), y la opción `TERMINAR SESIÓN` para terminar nuestra sesión y volver al GDM (Figura 3.2).

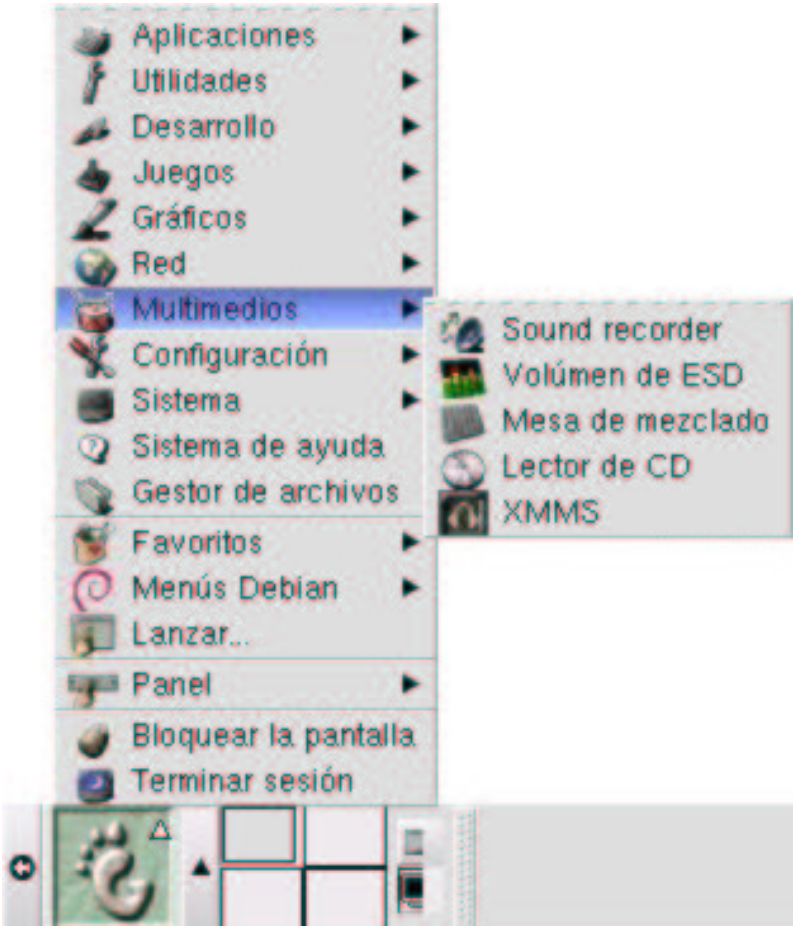


Figura 3.6: Panel alineado de GNOME y menú principal

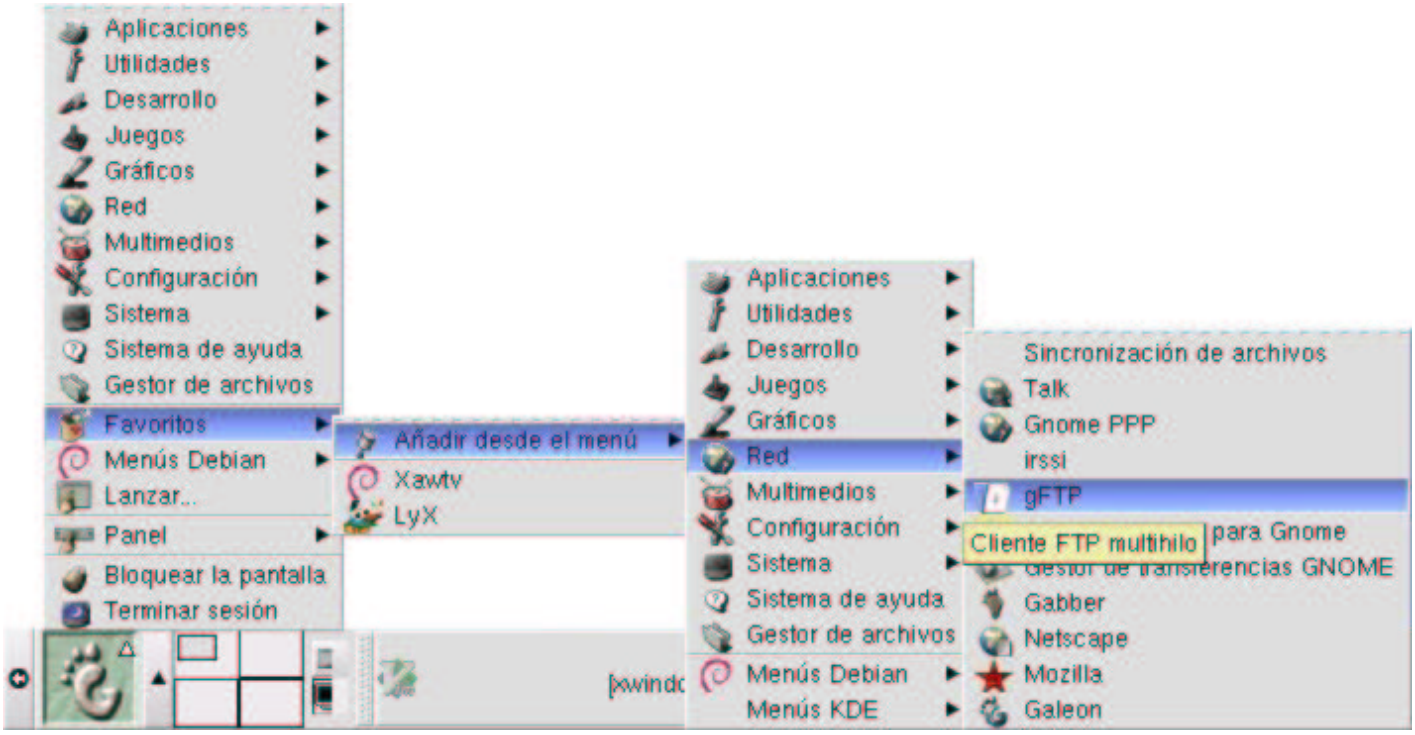


Figura 3.7: Añadir elemento al menú `FAVORITOS`

En GTK, y por tanto en GNOME, todos los menús cuentan con una pequeña marca en la parte superior (Figura 3.8). Si pulsamos en ella el menú se desprende de la aplicación y pasa a estar en una ventana independiente. De esa manera podemos acceder a un menú con opciones de uso habitual de forma más rápida y sencilla.

Figura 3.8: Hacer que un menú sea persistente

Tal y como podemos apreciar en la Figura 3.9, todos los elementos del menú principal disponen de un menú de contexto que se despliega cuando pulsamos con el botón derecho de nuestro ratón. Entre las opciones de dicho menú contamos con algunas que permiten añadir el elemento al panel, quitarlo del menú principal, añadirlo al menú `FAVORITOS` o alterar sus propiedades, entre muchas otras alternativas.

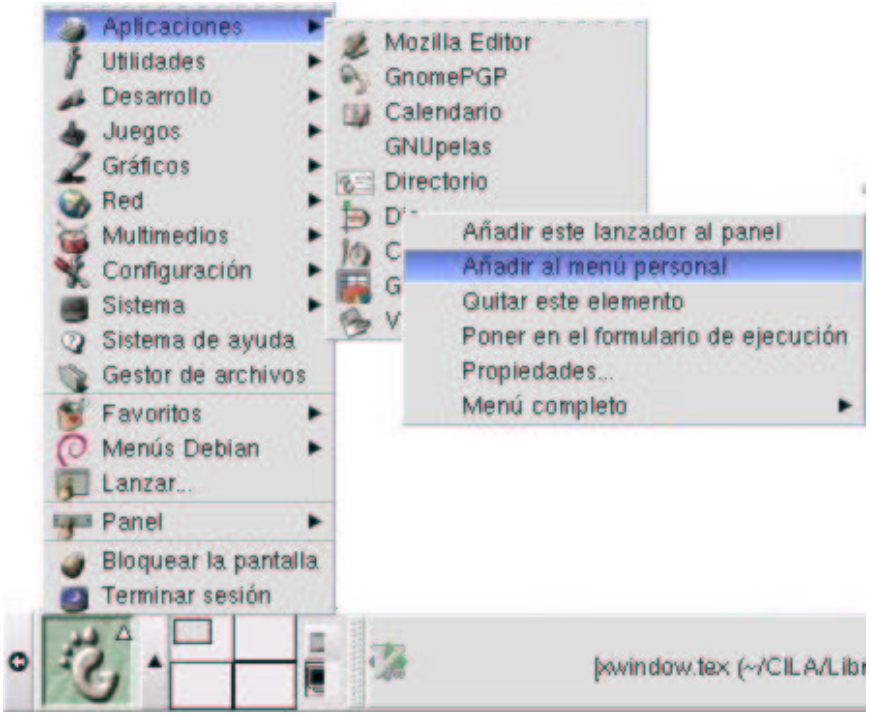
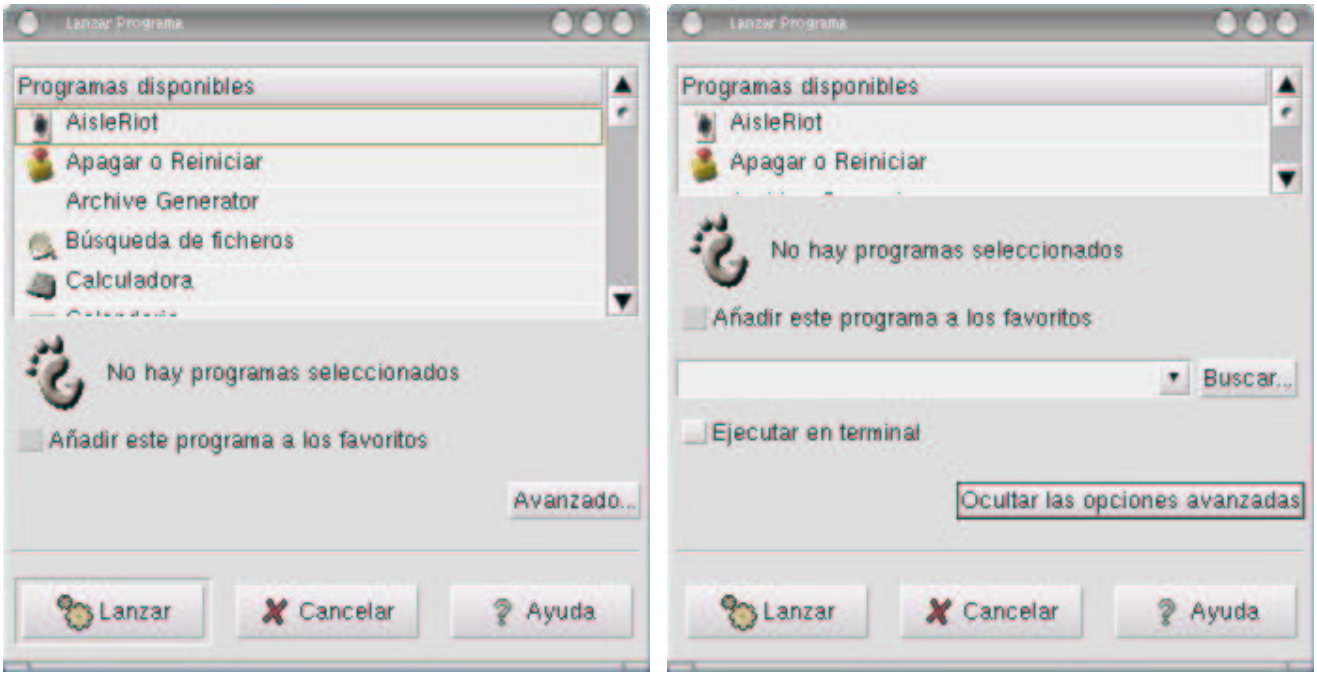


Figura 3.9: Menú de contexto de los elementos del menú principal

Una de las opciones más interesantes de dicho menú de contexto es la que se encuentra bajo el texto de `PONER EN EL FORMULARIO DE EJECUCIÓN`. Esa opción abre el cuadro de diálogo `LANZAR PROGRAMA` y lo rellena con las propiedades del elemento del menú principal sobre el que estamos trabajando. Ese mismo cuadro está disponible como `LANZAR` en el menú principal. En la Figura 3.10(a) vemos el cuadro de diálogo `LANZAR PROGRAMA`, que podremos observar si seleccionamos dicha opción. En dicho cuadro de diálogo podemos seleccionar la aplicación GNOME que deseamos ejecutar, así como añadirla

al menú de favoritos. El mismo cuadro de diálogo dispone de *opciones avanzadas* (Figura 3.10(b)) que nos permiten especificar la línea de comandos a ejecutar o especificar que deseamos que la aplicación se ejecute dentro de un emulador de terminal. La primera de las opciones resulta especialmente interesante si la aplicación no está disponible entre las que nos permite elegir el cuadro de diálogo (p. ej. `xkill`) o porque queramos pasarle alguna opción de la línea de comandos de la aplicación.



(a) Cuadro de diálogo `LANZAR PROGRAMA`

(b) Cuadro de diálogo avanzado `LANZAR PROGRAMA`

Manejar GNOME es cuestión de jugar un rato y probar las diferentes opciones. Disponemos de tres botones en nuestro ratón (ya hablamos de eso en apartados anteriores) y se trata de utilizarlos sobre los elementos del entorno para ver como reaccionan. Por ejemplo, si pulsamos con el botón derecho del nuestro ratón sobre algún área sin elementos de algunos de los paneles, veremos un menú de contexto similar al menú principal del que hemos estado hablando hasta el momento. En el menú del que estamos hablando existe un elemento denominado `PANEL` que contiene las opciones de configuración del panel. Con dichas opciones podemos:

AÑADIR AL PANEL Esta opción nos permite añadir nuevos elementos al panel. A parte de los tipos de botones entre los que nos deja elegir, podemos acceder a `MENÚ` para añadir algunos de los diversos tipos de menús de aplicaciones. También podemos acceder a `LANZADOR DESDE MENÚ` y elegir alguna aplicación del menú principal. Esto crea un botón que lanzará la aplicación automáticamente con sólo pulsar sobre el mismo. Todos estos elementos pueden ser movidos, incluso entre paneles, con sólo pulsar sobre ellos con el botón central. También podemos modificar sus propiedades pulsando con el botón derecho y seleccionando la opción adecuada.

CREAR UN PANEL Nos permite crear diferentes tipos de paneles en cualquier parte de la pantalla. Es importante destacar que para mover un panel de sitio basta con pulsar con el botón central de nuestro ratón en cualquier área descubierta del panel que pensamos mover. También podemos pulsar sobre las flechas de los extremos del mismo para ocultarlo, y ampliar de esta manera el área de trabajo de nuestro escritorio.

PROPIEDADES Con esta opción podemos alterar las características del panel. Algunas de esas características son el tipo de panel, su tamaño, la forma en que le panel se oculta, y la presencia de los botones de escondido. Además, bajo en nombre de `PROPIEDADES GLOBALES` se encuentran las opciones para configurar las características por defecto para todos los paneles.

EDITAR MENUS Carga el *editor de menús* que nos permite personalizar los menús de aplicaciones.

Uno de los aspectos más interesantes del entorno de escritorio de GNOME es la existencia de los *apliques* (o *applets*). Los apliques no son más que pequeñas utilidades cuya interfaz de usuario está confinada al área del panel donde está instalado

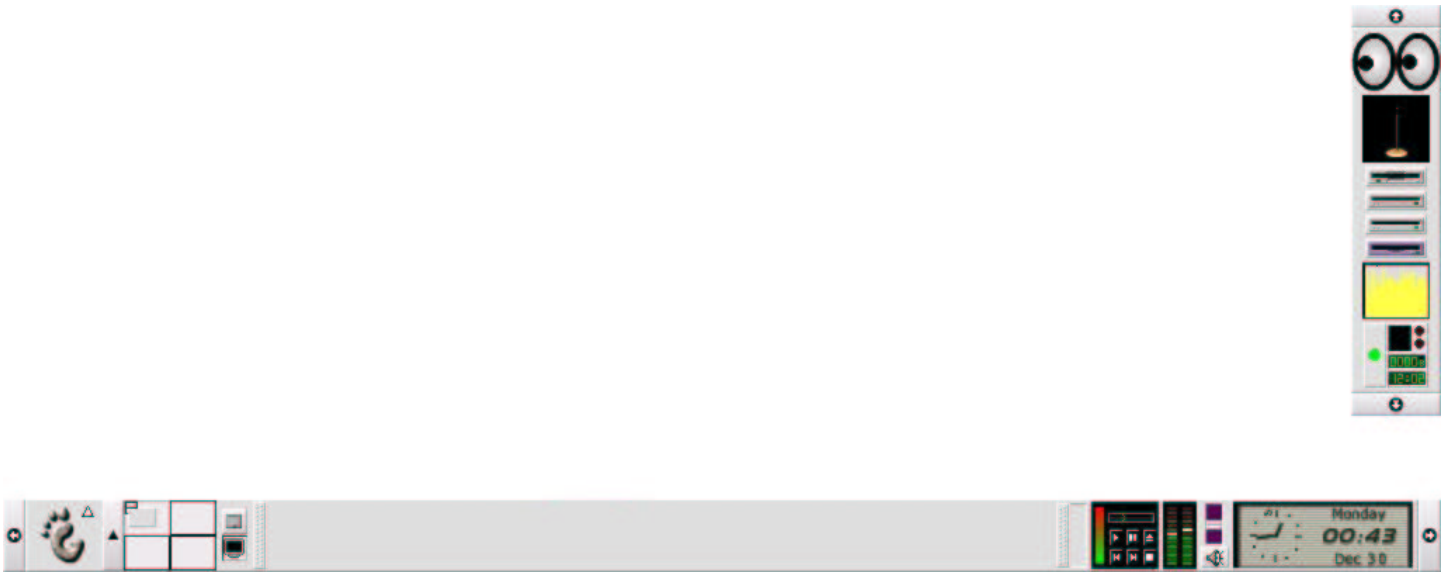


Figura 3.10: Ejemplos de paneles con apliques

(Figura 3.10). Su instalación es tan simple como seleccionar `PANEL` → `AÑADIR AL PANEL` → `APLIQUE` en el menú del panel. Los apliques están clasificados, por lo que debemos desplazarnos por las diferentes categorías hasta que seleccionemos el que deseamos instalar. Al igual que el resto de los elementos de los paneles, podemos moverlos utilizando el botón central del ratón o podemos alterar su propiedades utilizando el botón derecho. El comportamiento exacto frente a los diferentes uso del ratón sobre los apliques dependen de las preferencias del programador, por lo que puede ser interesante consular la ayuda de cada uno para aprender a explotar todo su potencial.

3.3.3. Administración de archivos

Al igual que los entornos de escritorio de otros sistemas operativos, GNOME permite la administración sencilla y eficiente de nuestros archivos. Dos son los programas que habitualmente nos podremos encontrar instalados en GNOME y que sirven para esta tarea:

3.3.3.1. GNOME Midnight Commander

Midnight Commander nació como una herramienta de consola que pretendía imitar y mejorar en lo posible a otra conocida utilidad del mundo de MSDOS denominada Comandante Norton®. Dicha utilidad simplificaba de forma considerable todas las tareas de administración de archivos (tales como la edición, la copia, la búsqueda, etc.) al proporcionar una completa interfaz en modo texto. Con el tiempo Midnight Commander fue portado al entorno gráfico de GNOME, aunque las principales mejoras siguen desarrollándose para la versión de consola que podemos utilizar con sólo ejecutar el comando `mc`.

GMC (GNOME Midnight Commander) se encarga de gestionar nuestro fondo de escritorio como área de trabajo[1]. En ella podemos depositar los archivos o directorios que utilicemos habitualmente. Con sólo pulsar con el botón derecho de nuestro ratón sobre el mismo, dispondremos de todo un conjunto de opciones para organizar y configurar nuestro escritorio. También podremos crear diferentes tipos de nuevos elementos, como por ejemplo directorios o lanzadores de aplicaciones.

Al igual que en otros entornos gráficos, podemos arrastrar y soltar los diferentes elementos para moverlos entre directorios. También podemos hacer una doble pulsación con el botón izquierdo para abrirlos, o pulsar el botón derecho de nuestro ratón para acceder a un completo menú con las acciones que podemos realizar sobre los elementos seleccionados.

Si elegimos abrir un directorio, GMC suele responder abriendo una ventana similar a la de la Figura 3.11. En el área de las izquierda de dicha ventana disponemos de una representación de todo el árbol de directorios de nuestro sistema, lo cual simplifica notablemente la navegación. Mientras que en el área de la derecha podemos ver los archivos o subdirectorios del

[1] En ocasiones y pese a que el GMC esté instalado el programa puede que no esté cargado, y por tanto carecemos de iconos en el escritorio. Para resolverlo podemos lanzar el comando `gmc` desde el cuadro de diálogo `LANZAR PROGRAMA`. Y a continuación seleccionar `CONFIGURACIÓN` → `SESIÓN` → `GUARDAR SESIÓN ACTUAL` en el menú principal de GNOME

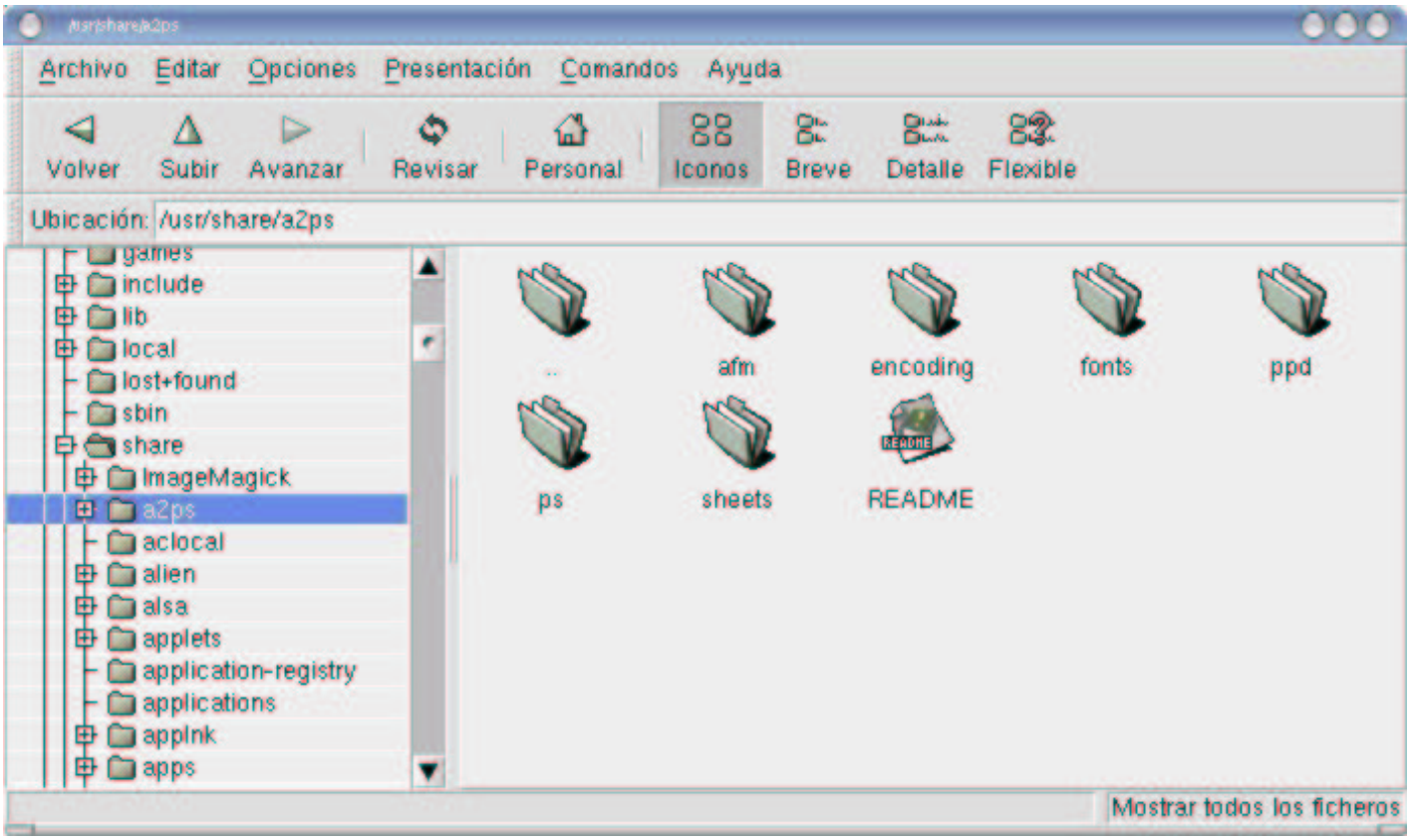


Figura 3.11: GNOME Midnight Commander

directorio seleccionado. GMC nos permite seleccionar diferentes tipos de vistas para los archivos, así como manipularlos a nuestra conveniencia utilizando el ratón. Nuevamente podemos arrastrar y soltar para mover archivos entre directorios, o podemos utilizar el botón derecho del ratón para seleccionar una de las múltiples acciones de las que disponemos.

3.3.3.2. Nautilus

Aunque GMC es un administrador de archivos sencillo y ligero su desarrollo está prácticamente abandonado. En su lugar se está imponiendo el uso de una aplicación mucho más potente, y visualmente más atractiva, denominada Nautilus. Aun así es importante destacar que Nautilus es una aplicación excesivamente pesada para ordenadores un tanto antiguos o escasos de recursos.

Nautilus pretende ser par GNOME lo que Konqueror es para KDE. Es decir, un administrador de archivos, un navegador web y un visor de documentos. Como podemos observar en la Figura 3.12 la disposición de todos los elementos en similar a la que tenemos en GMC. Es más, en lo que a la administración de archivo se refiere se comporta de forma exactamente igual. Si estamos acostumbrados a manejar a uno no tendremos problemas en utilizar el otro.

3.3.4. Configuración del escritorio

La configuración para personalizar nuestro escritorio se puede realizar desde el Centro de control (Figura 3.13). Para acceder al mismo basta con seleccionar `CONFIGURACIÓN` → `CENTRO DE CONTROL DE GNOME` en el menú principal del entorno.

En la parte derecha de la ventana contamos con la lista de categorías en las que se clasifican las diferentes opciones de configuración del entorno de escritorio GNOME. Cuando seleccionamos una de dichas categorías el lado izquierdo cambia para mostrarnos las opciones disponibles.

Cualquier cambio en la configuración puede ser validado con `ACEPTAR` o rechazado con `CANCELAR`. Sin embargo, podemos comprobar los efectos de esos cambios con sólo pulsar en `PROBAR`. El sistema automáticamente se ajustará a nuestras preferencias pero sin guardar los cambios definitivamente. Antes de cerrar el centro de control es necesario que pulsemos `ACEPTAR` en cada una de las categorías, independientemente de que las hallamos probado o no. El título de la categoría modificada

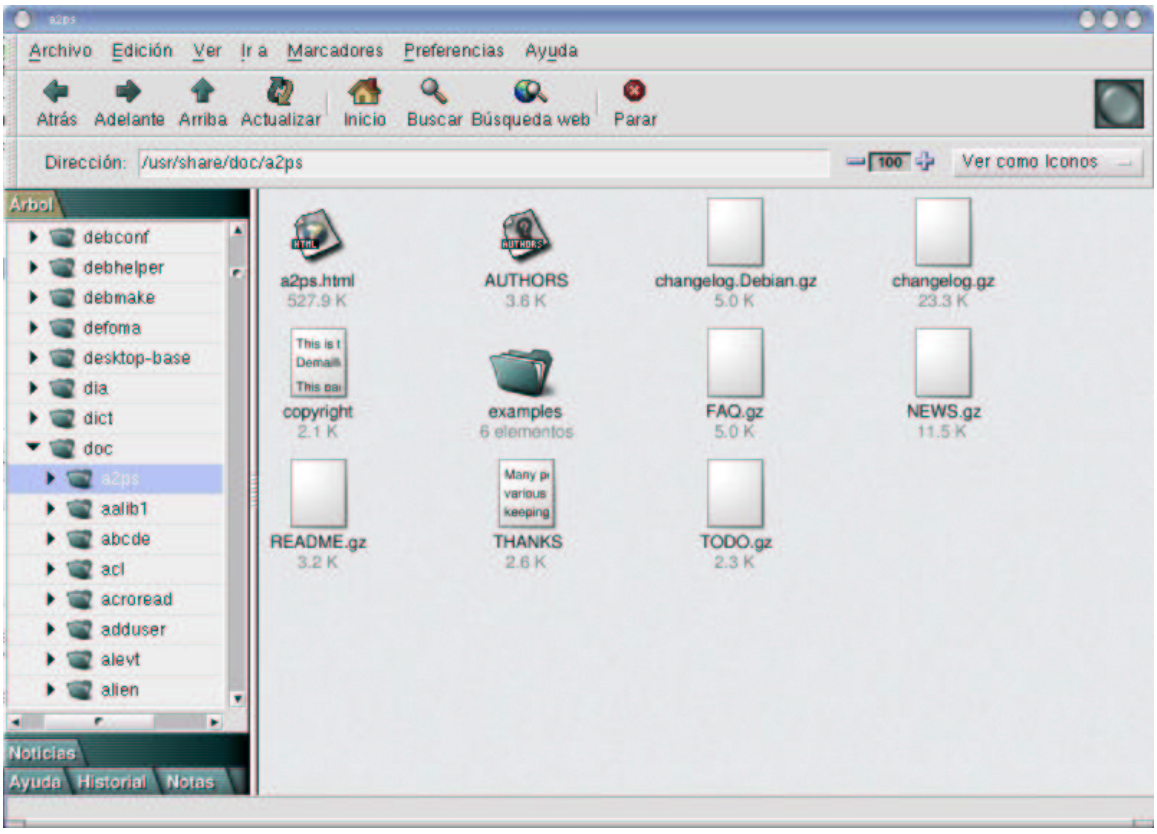


Figura 3.12: Administrador de archivos Nautilus

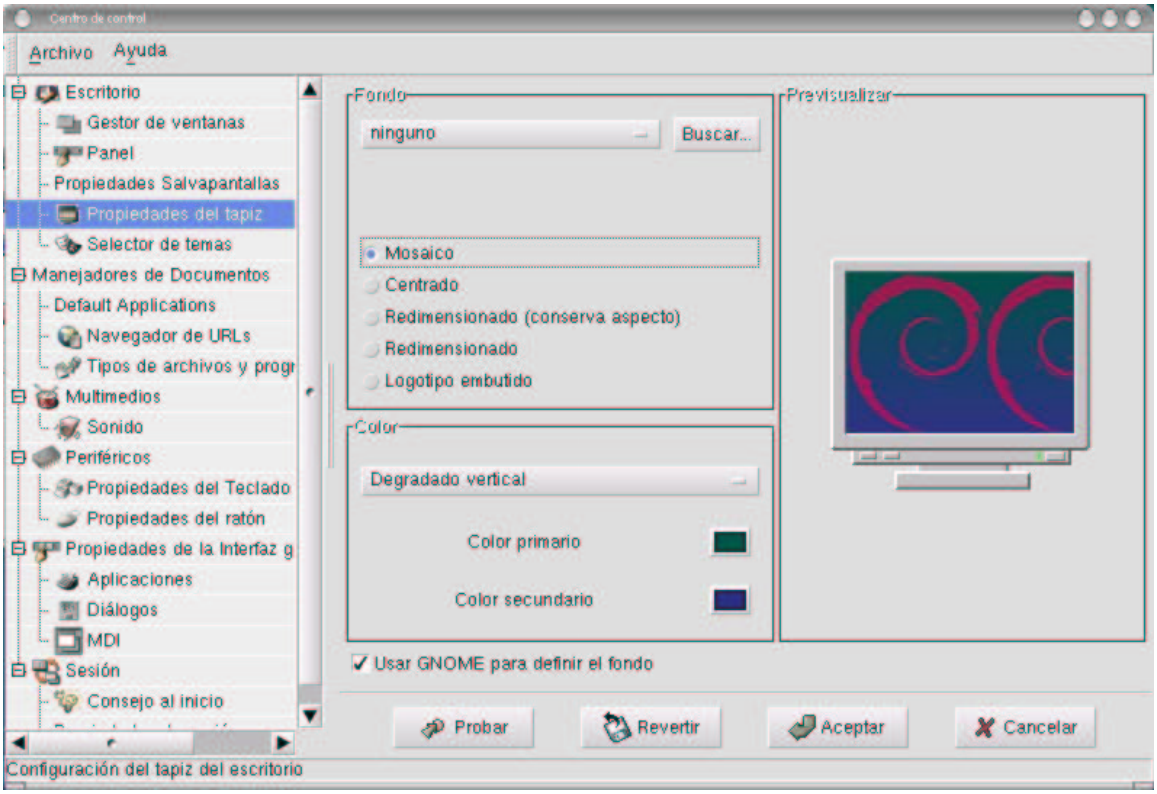


Figura 3.13: Centro de control GNOME

se muestra en rojo para indicarnos que todavía no hemos aceptado o rechazado los cambios. Por último, el botón `REVERTIR` nos permite devolver las opciones al estado anterior al de las modificaciones.

3.4. El escritorio de KDE

KDE es un entorno de escritorio con gestor de ventanas propio basado en X-Window que nos da un entorno gráfico amigable para realizar las operaciones más comunes con nuestro ordenador. En la Figura 3.14 podemos observar un ejemplo clásico del escritorio de KDE. Es importante recordar que en muchos sistemas para llegar a acceder a un escritorio como este es posible que tengamos que pasar por un *display manager*.

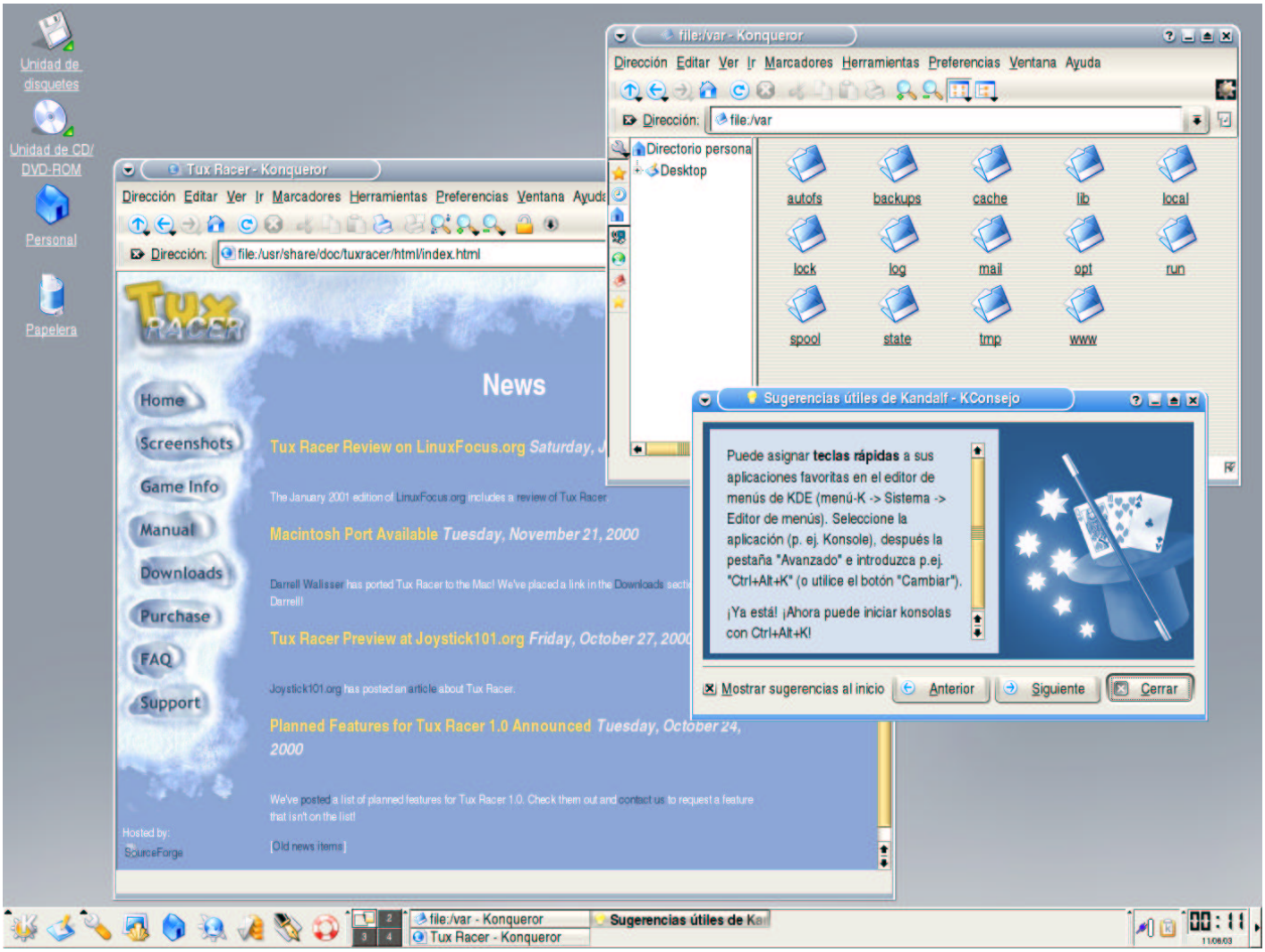


Figura 3.14: Aspecto del escritorio de KDE

El uso de un display manager u otro es independiente del escritorio que vayamos a utilizar. Sin embargo KDE dispone del suyo propio para garantizar una integración perfecta, en todos los sentidos, entre el entorno de escritorio y el display manager. En la Figura 3.15 podemos ver la clásica ventana de KDM, que como se puede apreciar presenta las características habituales en la mayor parte de los display managers existentes.

3.4.1. Paneles y menús de KDE

Al igual que en GNOME, en KDE contamos con un panel en el que se organizan todos los menús y aplicaciones (Figura 3.16). Este concepto es familiar en la inmensa mayoría de los entornos gráficos de usuario que podamos encontrar en cualquier ordenador, sólo que en ocasiones lo hayamos bajo un nombre diferente.

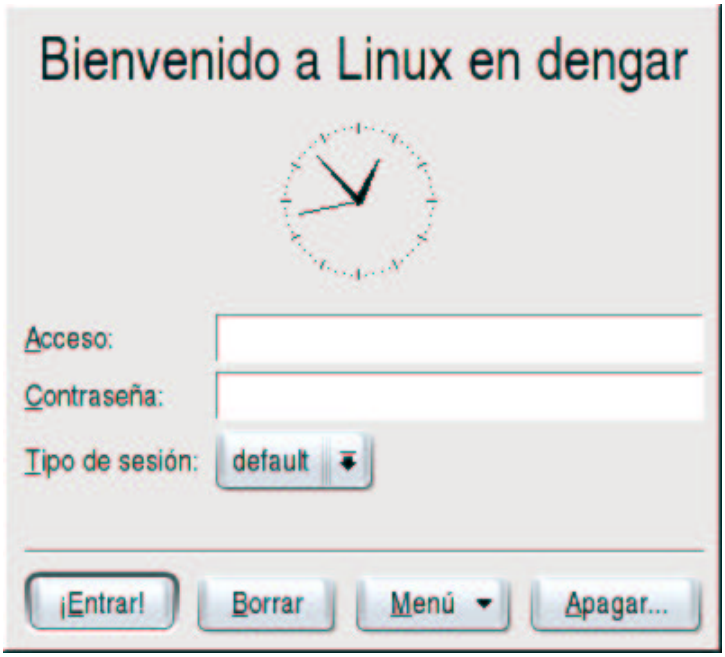


Figura 3.15: Ventana de autentificación de KDM

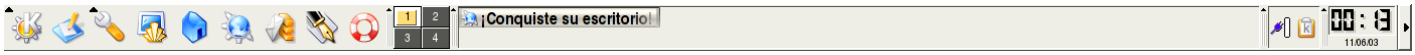


Figura 3.16: Panel principal de KDE

Si pulsamos con el botón derecho de nuestro ratón sobre alguna área sin elementos del panel, veremos un menú de contexto pensado para configurar los paneles de KDE. En el mismo disponemos de las siguientes opciones:

AÑADIR Esta opción nos permite añadir nuevos elementos al panel. Por ejemplo, podemos añadir BOTONES DE APLICACIÓN eligiendo la aplicación adecuada en el menú o submenús que se despliegan al elegir dicha opción. Dentro de esos submenús podemos seleccionar la opción AÑADIR ESTE MENÚ para añadir al panel un acceso al submenú completo (Figura 3.17). Con BOTONES ESPECIALES podemos crear accesos a algunas funciones integradas en KDE. Entre dichas funciones contamos con el MENÚ K, los MARCADORES, el SISTEMA DE IMPRESIÓN, o los DOCUMENTOS RECIENTES. Por último podemos añadir *applets* de forma similar a como haríamos en GNOME.

ELIMINAR Nos permite eliminar cualquier elemento integrado en el panel.

TAMAÑO Fijar el alto del panel a cualquiera de los valores predefinidos, o personalizarlo con el alto en pixels.

CONFIGURAR PANEL Con esta opción podemos alterar las características del panel. Algunas de esas características son la posición del panel, el alto y el largo, la forma en que el panel se oculta, y la presencia de los botones de escondido. También podemos crear *paneles hijo* adicionales en diferentes posiciones y con diferentes tamaños.

AYUDA Acceso a la ayuda del panel de KDE.

Para acceder a las aplicaciones instaladas podemos utilizar el MENÚ K. Dicho menú se activa al pulsar sobre el botón con una K y un engranaje a la izquierda del panel (Figura 3.18). Las aplicaciones se distribuyen en varios submenús que se clasifican por categorías. Normalmente estos submenús suelen tener títulos como APLICACIONES, MULTIMEDIA, INTERNET, etc. Dentro de cada submenú se encuentran los accesos a las aplicaciones más comunes de una categoría dada. Además podemos encontrar elementos como MÁS PROGRAMAS, que despliega un submenú con otros programas de la misma categoría pero no tan comunes, y DEBIAN, que da acceso a todas las aplicaciones de dicha categoría instaladas en la distribución, independientemente de que estén integradas en KDE o no.

Otras opciones son KFIND, que es una aplicación pensada para realizar búsquedas de archivos, y PERSONAL, que es un acceso rápido a nuestro directorio personal en la máquina. En el menú MARCADORES podemos tener acceso a archivos o contenidos de la Web que hemos decidido almacenar ahí. El objetivo es disponer de un acceso rápido, y ordenado por categorías, a dichos contenidos.

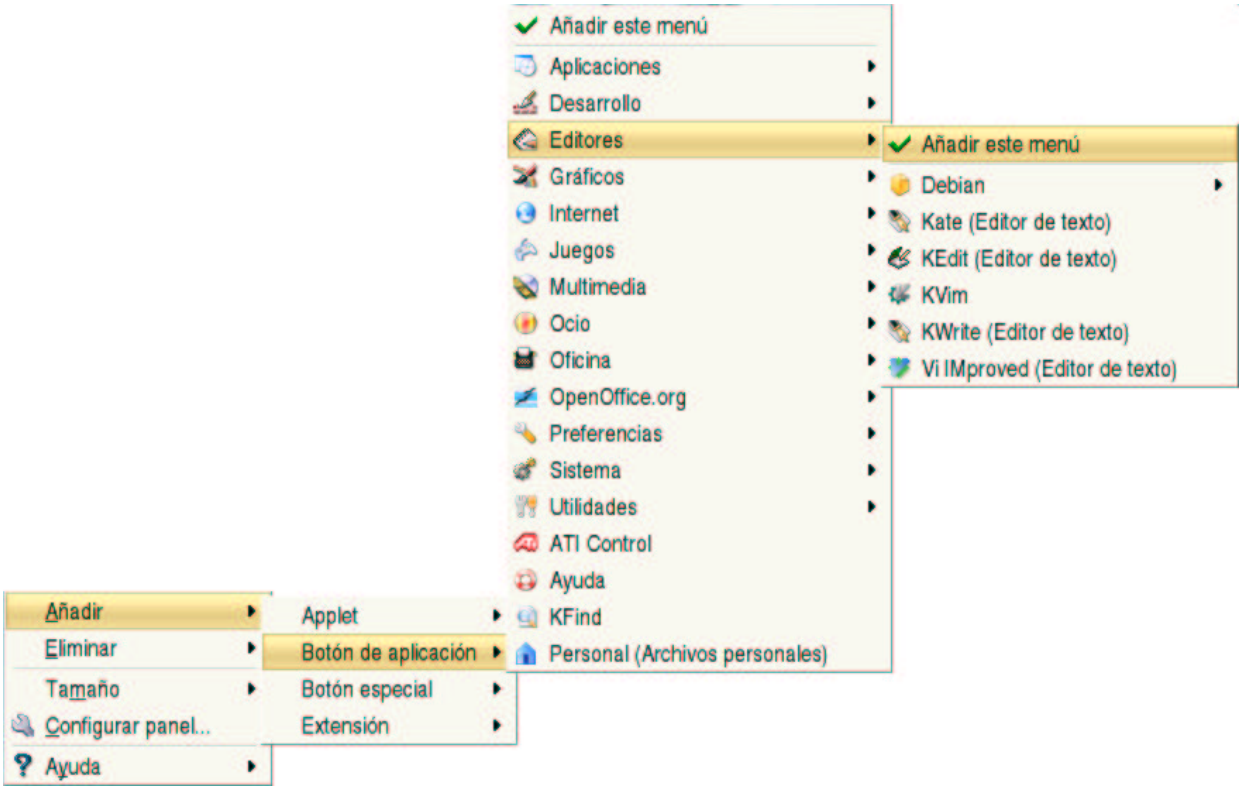


Figura 3.17: Añadir al panel un acceso al menú EDITORES

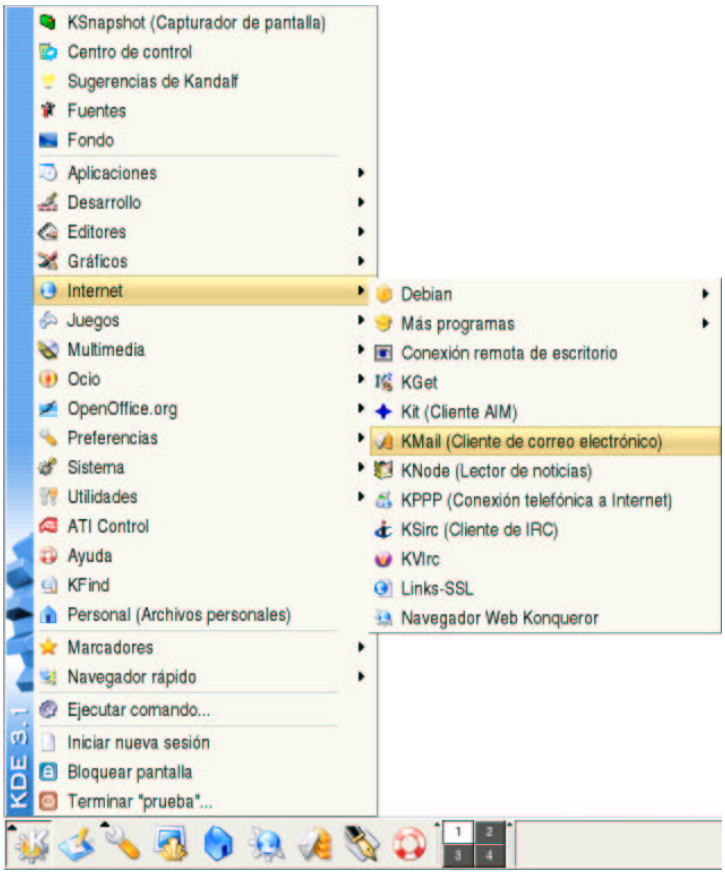
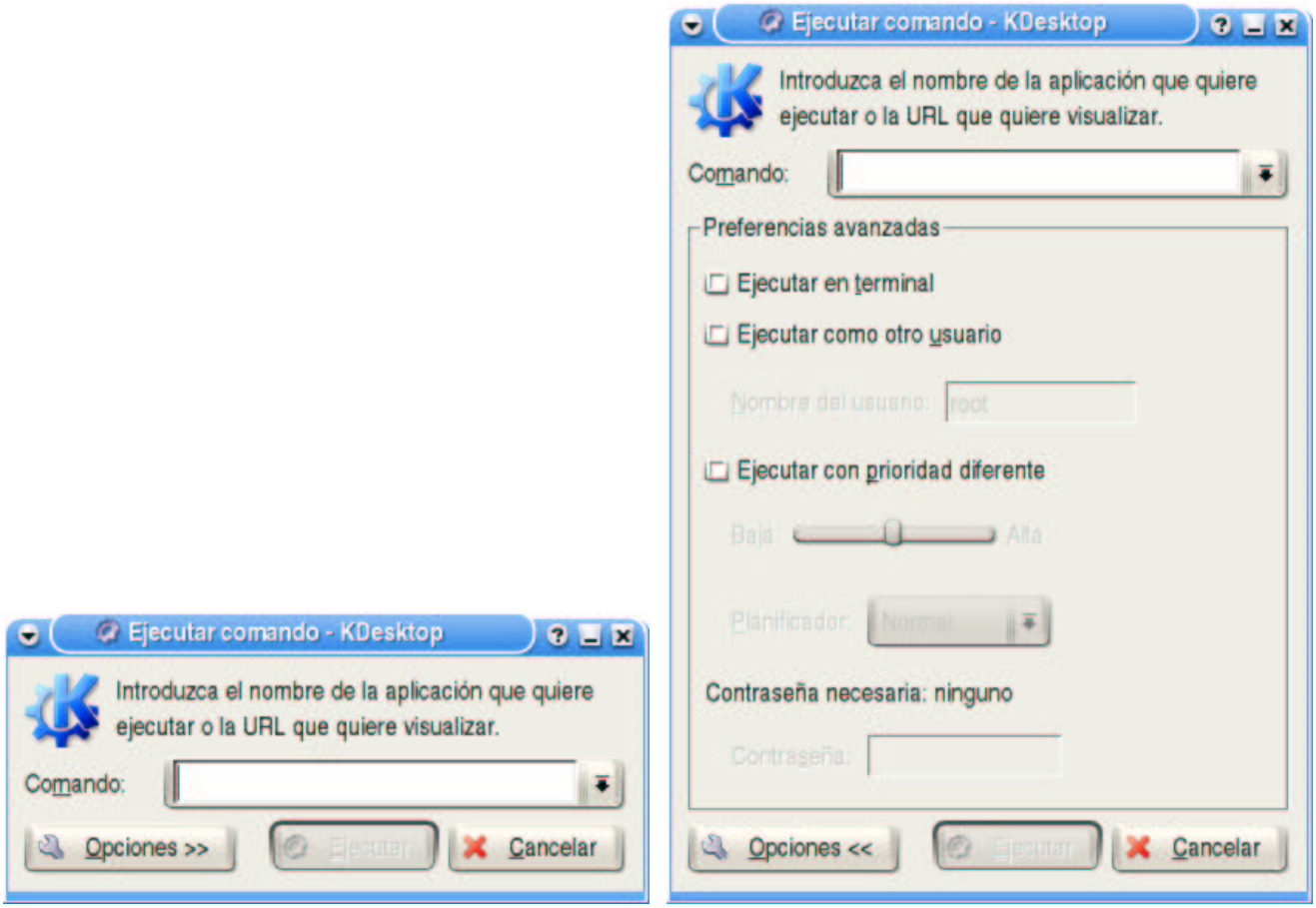


Figura 3.18: Menú K de KDE

La opción EJECUTAR COMANDO nos abre el cuadro de diálogo del mismo nombre, que podemos ver en la Figura 3.19(a). Dicho cuadro también puede ser abierto haciendo uso de la combinación de teclas A-F2. En él podemos indicar cualquier comando que queramos ejecutar. El mismo cuadro dispone de opciones (Figura 3.19(b)) que nos permiten especificar que el comando sea ejecutado como un usuario diferente, con un nivel de prioridad distinta, o indicar que deseamos que la aplicación sea ejecutada dentro de un emulador de terminal. A estas opciones se accede pulsando el botón OPCIONES del cuadro de diálogo.



(a) Cuadro de diálogo EJECUTAR COMANDO

(b) Cuadro de diálogo avanzado EJECUTAR COMANDO

Igual que en el entorno de escritorio GNOME en KDE podemos utilizar BLOQUEAR PANTALLA para proteger nuestro escritorio, si tenemos que abandonarlo por cualquier motivo, o TERMINAR SESIÓN para terminar nuestra sesión y volver al KDM (Figura 3.15). En este sentido es importante comentar que una de las opciones más interesantes, disponible a través del MENÚ K, es INICIAR NUEVA SESIÓN. Gracias a dicha opción podemos crear una nueva sesión de las X y asociarla a uno de los terminales virtuales (ver página ??). Esto nos permite tener dos o más sesiones abiertas en las X y conmutar entre unas y otras.

Ya hemos comentado que el panel de KDE puede contener uno o varios botones de aplicación, entre otros elementos. Habitualmente entre esos botones está el acceso al emulador de terminal de KDE, más conocido como Konsole. Dicho botón suele ser el que se simboliza como un monitor (o terminal) con una pequeña concha (shell en inglés) en una de sus esquinas. No debemos confundirlo con el que se encuentra por defecto inmediatamente a su derecha en algunas configuraciones de KDE, con unas barras de colores pintadas en el interior de un monitor. Este es el que abre el centro de control de KDE, mediante el cual podemos configurar la totalidad de los parámetros gráficos, estéticos, de sonido, de rendimiento, de manejo, etc.

Respecto a las funciones de los botones del ratón en KDE, el botón derecho sobre el escritorio nos muestra un menú con unas pocas opciones, como crear nuevos elementos en nuestro escritorio (accesos directos o enlaces, directorios, etc), ejecutar comandos, o configurar el escritorio. Si pulsamos con el botón derecho del nuestro ratón sobre alguno de los elementos del panel, veremos el menú de contexto asociado al elemento. Los menús de contexto de los applets suelen disponer de opciones acorde con las funciones asignadas a los mismos. Sin embargo, las opciones de los otros elementos suelen ser bastante estándar.

Por ejemplo, para mover el elemento por el panel, eliminarlo, cambiar sus propiedades y acceder a la configuración del panel. Concretamente esta última opción se denomina `MENÚ DEL PANEL`.

Los diferentes elementos del panel suelen venir delimitados por un separador con un pequeña flecha negra. Tanto si pulsamos con el botón derecho del ratón sobre dichos separadores, como si pulsamos con el izquierdo sobre la flecha dispondremos de un menú similar al que hemos comentado en el párrafo anterior (Figura 3.19).

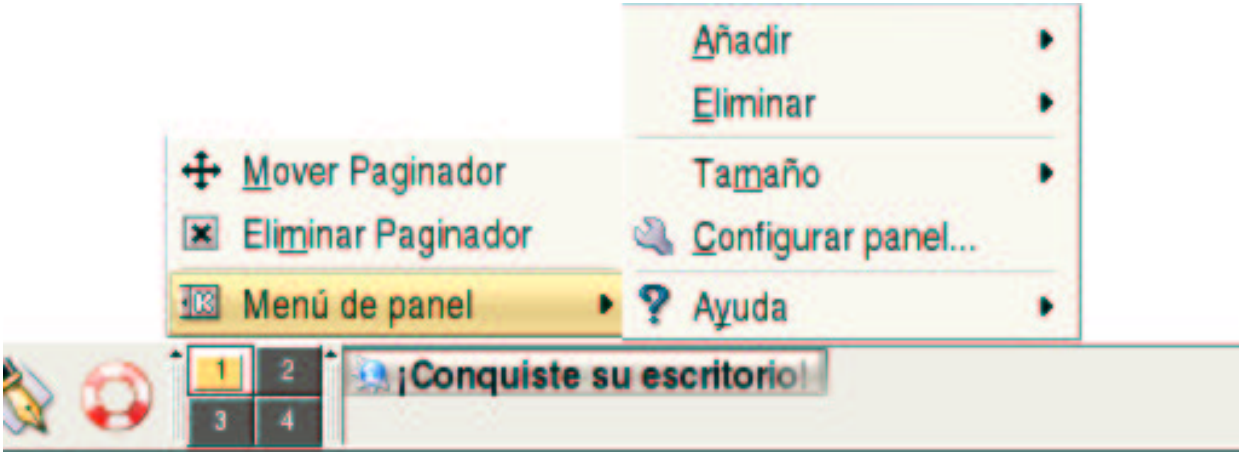


Figura 3.19: Menú del panel de KDE

3.4.2. Administración de archivos

Uno de los puntos fuertes a favor de KDE es el programa Konqueror. Konqueror integra en una sola aplicación las funciones de administración de archivos, navegación web y visor de documentos. Además de todo eso se trata de un programa completamente ampliable que puede aumentar sus funcionalidades a partir del uso de *plugins*. Gracias a ellos podemos realizar tareas como administrar los archivos en los equipos de nuestra red de área local (LAN), administrar y configurar nuestro sistema de impresión, examinar nuestros CDs de audio, etc.

Cuando pulsamos sobre los directorios o dispositivos presentes en nuestro escritorio se nos abre Konqueror con el objetivo de que podamos ver su contenido. En la Figura 3.20 podemos ver una ventana de Konqueror donde se puede explorar el directorio `/etc`.

Como podemos apreciar la ventana de Konqueror como administrador de archivos es similar a la de muchos otros administradores de archivos que hayamos podido ver. La mayor parte de dicha ventana está ocupada por los iconos de los directorios y archivos presentes en el directorio actual. Inmediatamente encima tenemos la *barra de herramientas de dirección* donde podemos escribir la ruta del directorio que deseamos explorar. Puesto que Konqueror es un programa que integra múltiples funcionales, en dicha barra podemos poner la ruta de otros recursos diferentes a un directorio. Por ejemplo, podemos poner una ruta hacia una página web (`http://nombre_equipo/ruta_página`), a un directorio en un FTP (ftp://nombre_equipo/nombre_directorio) e incluso a una carpeta compartida de Microsoft® Windows® (`smb://nombre_netbios/carpeta`).

Como ya hemos comentado Konqueror dispone de características de visor. Eso significa que pulsando sobre los iconos de los archivos podemos ver su contenido. En los supuestos en que Konqueror no esté capacitado para la visualización de un determinado contenido se abre el programa adecuado. De la misma manera pulsando sobre los directorios podemos navegar por el sistema de ficheros de nuestro equipo. En la barra de herramientas de la parte superior disponemos de botones para subir un nivel en el árbol de directorios, avanzar o retroceder, o cambiar el modo de vista de los archivos. Usando *arrastrar y soltar* podemos mover, copiar o crear un enlace a archivos y directorios. Mientras que pulsando con el botón derecho disponemos de un menú de contexto con algunas opciones adicionales, como renombrar, eliminar, o editar las propiedades de un archivo o directorio dado.

En el lado de la izquierda de la ventana contamos con el panel de navegación, que puede desarrollar diferentes funciones según la pestaña de la izquierda que hayamos seleccionado. Podemos navegar por nuestros marcadores, por el historial de navegación, por la Red, por el árbol de directorios de nuestro equipo, etc. En general Konqueror es un programa muy potente con decenas de funciones que debemos probar.

Para utilizar Konqueror como navegador debemos pulsar en el botón con un globo terráqueo y un engranaje situado en nuestro panel (ver Figura 3.16). En la Figura 3.21 podemos ver el ejemplo de un ventana de Konqueror con una pagina web cargada. La forma de manejo del mismo no dista de la forma de manejo de cualquier otro navegador que hayamos utilizado.

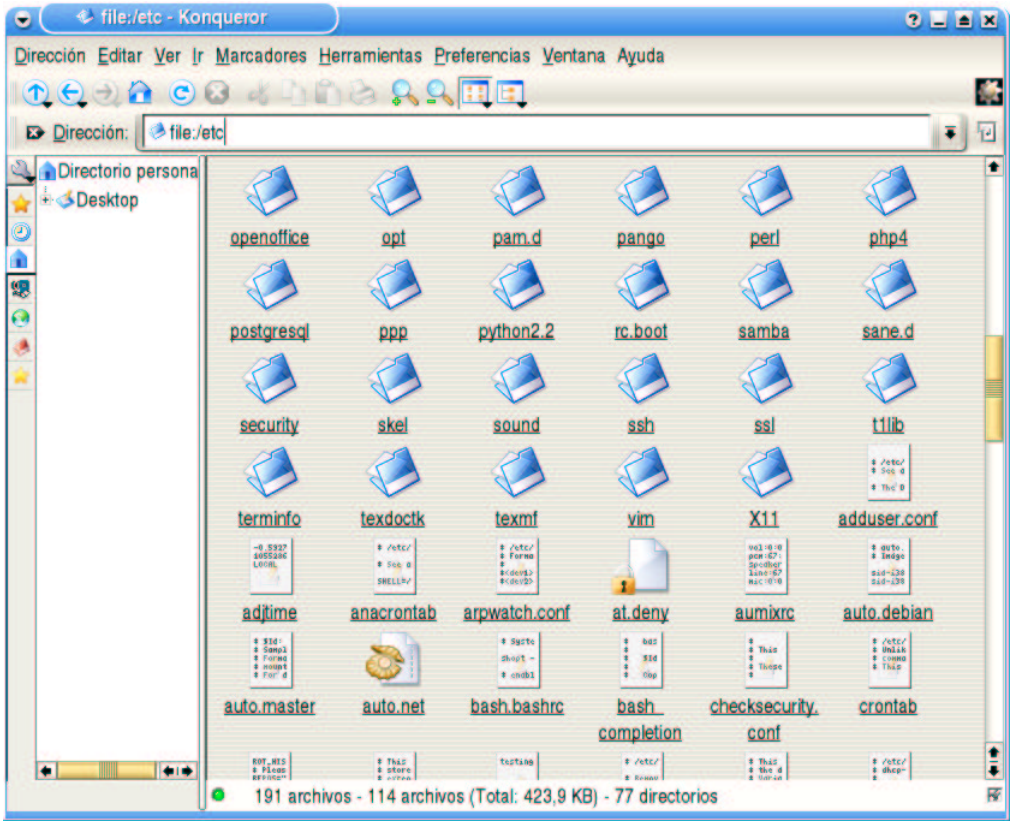


Figura 3.20: Konqueror como administrador de archivos

Figura 3.21: Konqueror como navegador web

3.4.3. Configuración del escritorio

La configuración para personalizar el escritorio se puede realizar desde el Centro de control (Figura 3.22). Para acceder al mismo basta con seleccionar `PREFERENCIAS` → `CENTRO DE CONTROL` en el menú del entorno.

Figura 3.22: Centro de control de KDE

En la parte derecha de la ventana contamos con un árbol de categorías en las que se clasifican las diferentes opciones de configuración del entorno de escritorio KDE. Cuando seleccionamos una de dichas categorías el lado izquierdo cambia para mostrarnos las opciones disponibles.

Cualquier cambio en la configuración debe ser validado con `APLICAR` para que tenga efecto. Adicionalmente podemos utilizar `RESTAURAR` para cancelar las modificaciones realizadas en la configuración, o `PREDETERMINADO` para cargar la configuración por defecto de KDE en la categoría seleccionada. En caso de realizar modificaciones en la configuración de una categoría, y de que intentemos cambiar a otra sin validar los cambios de la actual, el sistema nos avisará de la situación y nos permitirá aplicar los cambios, deshacerlos, o cancelar el cambio de categoría.

Con todo esto, el amigo lector ya tiene opciones gráficas para elegir. ¡Que lo disfrute!.

TEMA 4

Editores de texto

4.1. VI & VIM

VI es un editor de texto visual, de pantalla completa, basado en el editor de línea `ex`. Es un editor poco intuitivo y con mala prensa entre los estudiantes que dan sus primeros pasos en Linux/UNIX, pero por otra parte es el editor favorito de los usuarios avanzados y de muchos programadores. Es además un editor que se puede encontrar en cualquier sistema UNIX, desde antiguas estaciones Sun Solaris o HP-UX hasta las más recientes distribuciones de GNU/Linux o FreeBSD, OpenBSD, etc.

VI es además un editor muy potente, que permite hacer complicadas operaciones en grandes ficheros con unos pocos comandos, por lo que su aprendizaje puede ahorrarnos mucho tiempo. Otra ventaja de VI es que al ser tan corriente suele encontrarse incluso en disquetes de rescate. Lógicamente poco se puede rescatar si no se sabe manejar el único editor disponible en un momento de emergencia.

Pero el manejo de VI es realmente incómodo si nos enfrentamos a la versión clásica. Por ejemplo no podemos usar los cursores para movernos por el texto, debemos pasar al llamado modo *comando* y utilizar letras para movernos.

En este curso utilizaremos el editor VIM. VIM significa "**V**i **IM**proved" (en español "**VI M**ejorado"), y como su nombre indica es un clon (muy) mejorado del clásico editor VI. VIM es bastante más amigable que VI, ya que permite un uso más intuitivo (p.ej. los cursores y otras teclas para moverse). Además, si se compila adecuadamente (y suele ser así) incorpora coloreado de sintaxis para casi todos los lenguajes de programación y ficheros de configuración que puedas encontrar en Linux/UNIX. Otra característica muy interesante de VIM es la posibilidad de utilizar un interfaz gráfico, lo que aumenta el confort y mejora el rendimiento del programador.

Lo primero que hay que aprender con VI es la filosofía de los dos modos de trabajo: el modo *comando* y el modo *edición*. El modo comando se utiliza solamente para dar órdenes al editor, decirle que haga cosas como borrar una línea, buscar un patrón, ir a una determinada línea, guardar el fichero, salir, etc. El modo *edición* se utiliza solamente para escribir texto en el fichero. Es muy importante familiarizarse con esta filosofía de funcionamiento, ya que resulta imprescindible para cualquier operación que se quiera realizar con VIM.

En adelante nos referiremos al editor como VI cuando digamos algo que sea común a todas las versiones de VI, y como VIM cuando se trate de extensiones propias de VIM.

Para ejecutar este editor el comando es `vi`. Si tienes instalada otra versión de VI pero quieres utilizar VIM no hay problema, simplemente ejecuta `vim`. Si quieres utilizar la interfaz gráfica de VIM ejecuta `gvim`.

Aunque conserva el nombre de VI estamos trabajando con VIM. Este comando admite varias opciones que se le pueden pasar como parámetros, p.ej. el nombre del fichero que queremos editar:

```
$ vi fichero
```

También puedes especificar la línea en la que quieres empezar con el parámetro `+n`. Por ejemplo para empezar en la línea 12:

```
$ vi fichero +12
```

`i`	Comienza a insertar texto en la posición del cursor.
`a`	Comienza a insertar texto en la posición siguiente a la del cursor.
`o`	Comienza a insertar texto en una nueva línea debajo del cursor.
`s`	Borra el carácter en la posición del cursor y comienza a insertar texto.
`I`	Comienza a insertar texto al principio de la línea.
`A`	Comienza a insertar texto al final de la línea.
`O`	Comienza a insertar texto en una nueva línea encima del cursor.
`S`	Borra la línea en la posición del cursor y comienza a insertar texto.

Tabla 4.1: Pasar el modo *edición* en Vi

`VIM` comienza siempre en modo `comando`, preparado para realizar operaciones sobre el fichero. Una de estas operaciones es pasar al modo `edición`. Esta transición puede hacerse de 8 formas diferentes.

Para pasar del modo *edición* al modo *comando* basta con pulsar `ESC`. A continuación vamos a editar un pequeño fichero para familiarizarnos con sus comandos básicos. Comenzamos invocando al editor desde la línea de comandos (suponemos que estás utilizando la interfaz gráfica, aunque también puedes prescindir de ella):

```
$ gvim hola.c
```

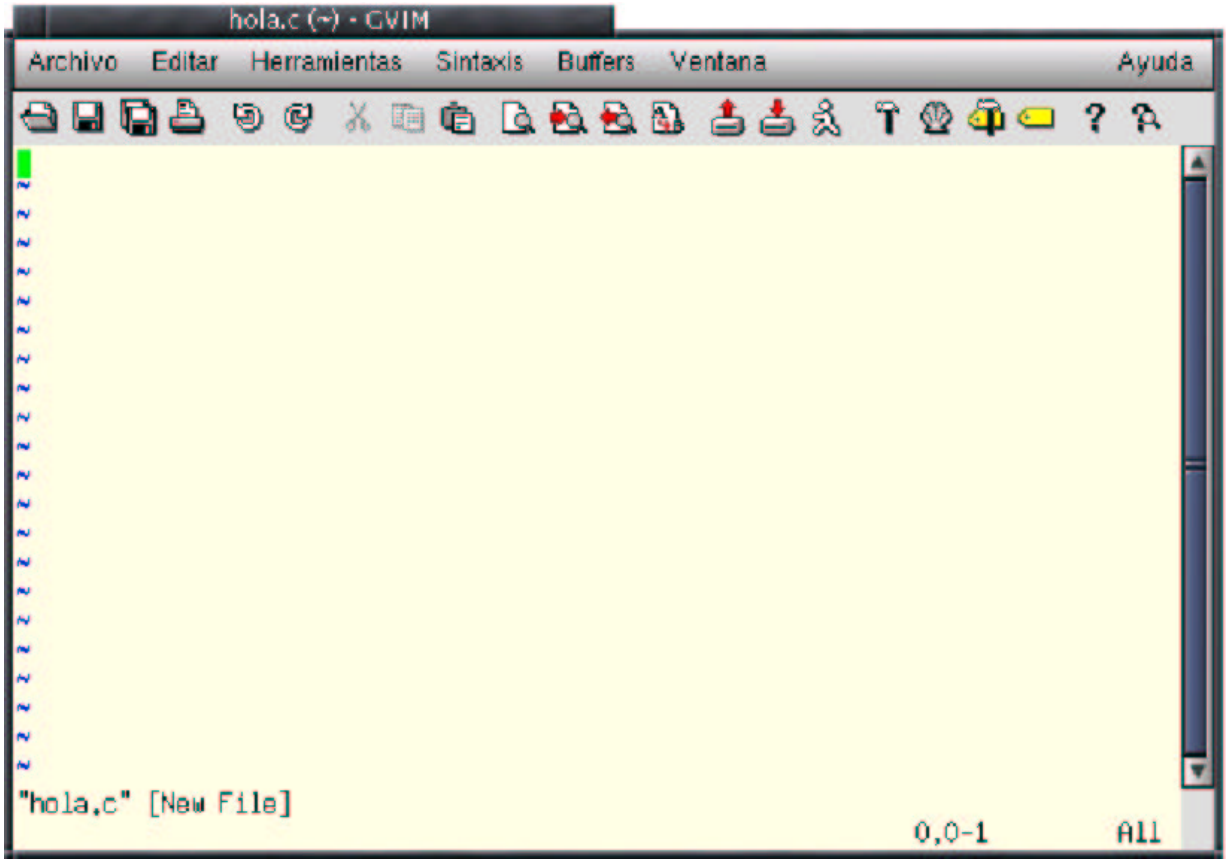


Figura 4.1: Interfaz gráfica de VIM con un fichero nuevo (vacío)

La última línea de la interfaz (o de la consola) debería parecerse a esto:

```
"hola.c" [New File]                                0,0-1         All
```

Esta línea es la *barra de estado* del editor. Es aquí donde teclearemos los comandos y donde aparecerá cierta información como el modo en el que estamos (comando, edición, reemplazo, etc.), la línea y columna en la que estamos, el porcentaje del documento en el que estamos, si el fichero ha sido salvado y su tamaño.

A continuación pulsamos la tecla `i` para pasar al modo *edición*. Observamos que la barra de estado se muestra diferente:

```
-- INSERT --                                          0,1           All
```

Ahora es el momento de teclear, escribe el siguiente código:

```
#include <stdio.h>

int main()
{
    printf("Hola Mundo\n");
}
```

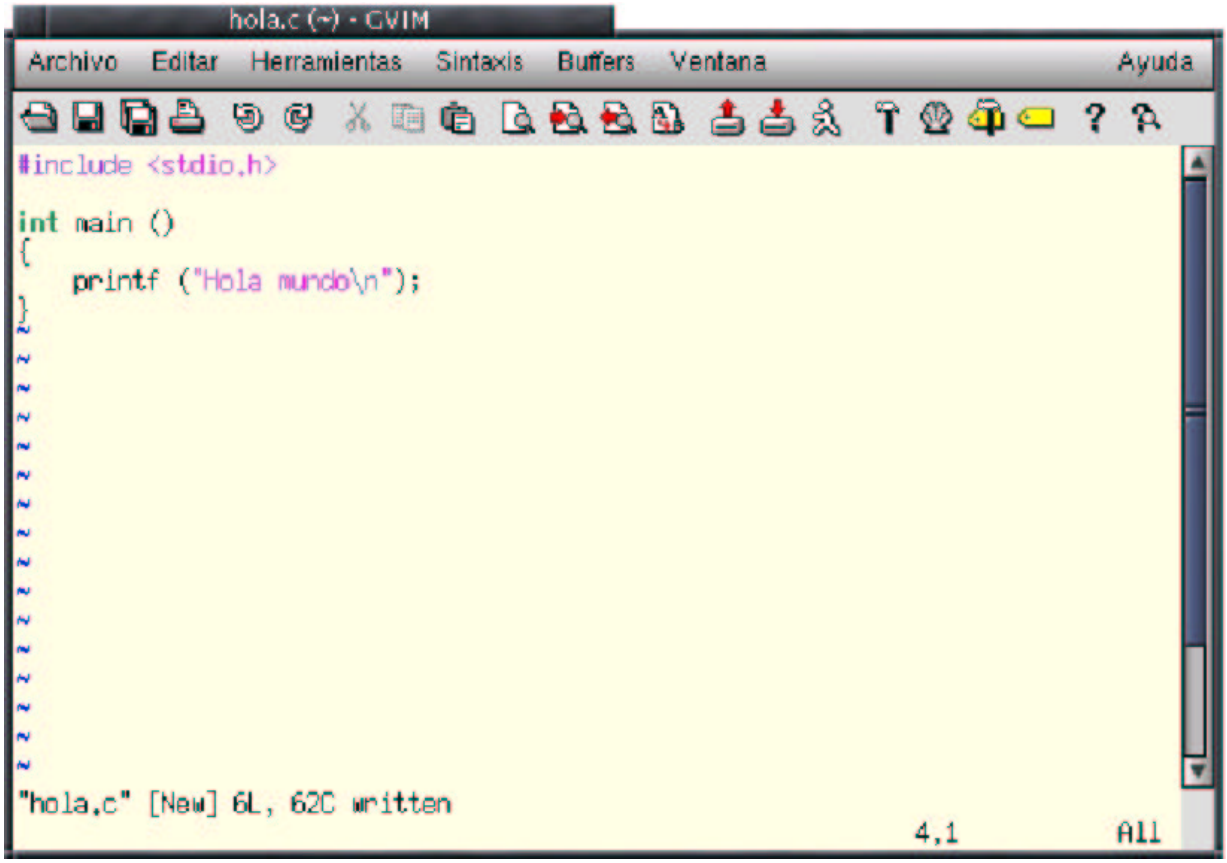


Figura 4.2: Interfaz gráfica de VIM con código C coloreado

Cuando tengas esto escrito, pulsa `ESC` para pasar al modo *comando*. Entonces teclea la orden `:w` y pulsa `Enter`. Veremos como la orden `:w` aparece en la barra de estado mientras la tecleamos, y luego al ejecutarla se muestra información sobre el resultado, en este caso información sobre el fichero que acabamos de guardar.

```
"hola.c" [New] 6L, 62C written                        6,1           All
```

En esta línea Vi nos indica que ha escrito el fichero hola.c, que es nuevo, que tiene un tamaño de 62 caracteres en 6 líneas. Además al terminar de escribirlo estamos en la línea 6 y en la columna 1, y podemos ver todo el fichero.

Si tu versión de VIM tiene soporte para coloreado de sintaxis probalmente ya estará funcionando y verás las palabras `int` e `include` de distinto color. Sin aún no estás viendo el código coloreado prueba a pasar al modo *comando* (pulsando `ESC`) y ejecutar el comando `:syntax on` (para los comandos que empiezan en `:` hay que pulsar `Enter` al final). Si te devuelve el mensaje de que esa función no está implementada es que no tienes disponible el coloreado de sintaxis. Si por el contrario funciona pero no te gusta y quieres quitarlo ejecuta el comando `:syntax off` y desaparecerán los colores. En la interfaz gráfica de VIM hay un menú Esquemas de colores dentro del menú Editar. También puedes cambiar el esquema de colores desde la interfaz de consola del VIM con el comando `:colors esquema`. Por ejemplo, activa el coloreado de sintaxis y ejecuta `:colors koehler`.

En adelante, el comando `x` significará pulsar la tecla `x` en modo *comando*, el comando `:letras` significará teclear `:letras` en modo comando y pulsar `Enter`. Cualquier comando del primer tipo puede modificarse teclando un número, lo que hará que se ejecute ese número de veces (el comando `4x` es equivalente a `xxxx`). Si te equivocas tecleando cualquier comando puedes cancelarlo pulsando `Esc`.

Para deshacer cualquier operación realizada pulsamos en modo *comando* la tecla `u`. Para salir del editor **sin** guardar el fichero utiliza el comando `:q!`. Probablemente realizarás esta operación con mucha frecuencia al principio, cuando se cometas algún error grave como teclear una palabra sin pasar al modo *edición*. En el modo *comando* cada tecla tiene su función, y es diferente además si está en mayúsculas que si está en minúsculas. Por lo tanto, una regla de oro con VI es: **cuando no sepas con qué comando hacer algo, no pruebes teclas al azar**. Tampoco es nada bueno utilizar la tecla. Y por supuesto, cuando estés editando algo importante guarda el fichero con frecuencia con el comando `:w`, sobre todo en las primeras semanas de uso. Para salir de VI guardando el fichero utiliza el comando `:x`.

Una utilidad que te puede resultar interesante es ejecutar comandos del sistema desde la misma consola en la que estamos trabajando con el VIM. En modo *comando* escribimos `:!comando`, por ejemplo, si escribimos `:!ls` nos mostrará un listado de archivos del directorio en el que nos encontramos. Esto te permite compilar programas o documentos mientras editas uno o más ficheros de código fuente, incluso puedes ejecutar una consola de sistema desde el editor, con el comando `:sh`

Para buscar palabras dentro de un documento es fácil, solo tienes que escribir en modo *comando* `/palabra` y si salen varios resultados, pulsando `n` pasarás al siguiente y pulsando `N` al anterior. Si prefieres que la búsqueda sea hacia atrás (desde el final hacia el principio del fichero) el comando que necesitas es `?palabra`. También con esta búsqueda pasas de un resultado a otro pulsando `n`.

Si lo que necesitas es desplazarte por un documento muy grande (como por ejemplo el código fuente de un programa) a líneas concretas VIM lo soluciona. Para desplazarnos a la línea 60 de un documento, escribe en modo *comando* `60` y luego pulsa `S-g` o `G`. Si pulsas sólo `S-g` o `G` te desplazas al final del fichero, y si pulsas `gg` (pulsas `g` dos veces seguidas) te desplazas al principio del fichero.

Para desplazamientos más cortos dispones del comando `0` (cero) que te sitúa el principio de la línea, `^` que te sitúa en el primer caracter no blanco (espacio o tabulador) de la línea, `$` que te sitúa al final de la línea, `w` te sitúa al principio de la siguiente palabra, y `b` hace lo mismo con la anterior palabra. El comando `)` te desplaza a la siguiente frase, mientras que `(` te desplaza a la anterior. Del mismo modo puedes utilizar `{` y `}` para saltar de párrafo en párrafo.

Si en lugar de desplazarte por el fichero lo que quieres es borrar parte de su contenido estos mismos comandos, pulsando previamente `d`, hacen lo siguiente: `d0` borra desde el principio de la línea hasta justo antes del cursor, `D` o `d$` borra desde el cursor hasta el final de la línea, `dw` borra desde el cursor hasta el principio de la siguiente palabra, `db` borra desde justo antes del cursor hasta el principio de la palabra en la que se encuentre el cursor, `dd` borra la línea en la que se encuentre el cursor, `dj` borra la línea en la que se encuentre el cursor y la que estén justo debajo, `34dj` borra la línea en la que se encuentre el cursor y las siguientes 34 líneas que estén justo debajo, `d)` borra desde el cursor hasta el principio de la siguiente frase, `d(` borra desde el cursor hasta el principio de la frase en la que se encuentre el cursor, `d}` borra desde el cursor hasta el final del párrafo en la que se encuentre el cursor, `d{` borra desde el cursor hasta el final del párrafo anterior.

A lo mejor te interesa pegar en otro sitio ese trozo de fichero que has borrado, o prefieres copiar y pegar en lugar de cortar y pegar. Claro que puedes hacerlo con VI y seleccionando exactamente lo que quieras. Para seleccionar una parte del fichero sitúate en un extremo de la selección (principio o final), pulsa `v` y desplázate hasta el otro extremo de la selección. Para cancelar la selección pulsa `Esc`. Una vez que tienes el texto seleccionado puedes cortarlo pulsando `d` o copiarlo pulsando `y`.

Para pegar lo que has copiado o cortado has de situar el cursor en la posición deseada y contemplar dos opciones para pegar el texto: pulsando `p` el texto se insertará *justo después* del cursor, pulsando `S-p` o `P` el texto se insertará *justo antes* del cursor.

Cuando cortas o copias una región de texto ésta se guarda en una zona de memoria, lo que se denomina un *buffer*. Mientras no salgas de VIM este buffer se conserva completamente hasta que vuelvas a cortar o copiar otra región. Ten cuidado porque borrar algo con cualquier comando también se considera cortar, por lo que sobreescribe el buffer. Este buffer se conserva además entre ficheros, por lo que puedes cortar o copiar en un fichero y pegar en otro, siempre que no salgas del editor para hacer esta maniobra. Si sales del editor se conserva sólo una parte del buffer, suele ser suficiente para regiones no muy grandes, pero no te confíes[1].

Si quieres cortar o copiar de un fichero y pegar en otro lo mejor es que abras ambos ficheros con el mismo VI, simplemente ejecutando:

```
$ vi fichero1 fichero2
```

[1] En el momento de escribir esto la versión de VIM 6.1-474 limita el buffer a 50 líneas

Con esto tienes varios ficheros abiertos, con un buffer para cada uno (aquí entendemos buffer en el sentido con el que lo hacemos al hablar de Emacs en la sección 4.2) y puedes pasar de un buffer a otro con el comando `:bN` donde `N` es el número del buffer (se numeran empezando por 1). Para pasar de un buffer a otro primero debes guardar a fichero el buffer en el que te encuentres (con el comando `:w`).

Otro comando que te ayudará si tienes que programar es el comando %. Si estás trabajando con algunos lenguajes cuyas sintaxis manejan con mucha frecuencia `(`, `{` o `{` , VIM te permite con pulsar la tecla % situarte sobre uno de estos caracteres que te muestre el de cierre o apertura que le corresponda.

4.2. GNU Emacs

Emacs, junto con Vi, ha sido uno de los primeros editores de texto para UNIX. A pesar de visualmente presenta un interfaz similar al de un editor de texto corriernte, como podría ser joe, el edit de MS-DOS, o similar, lo cierto es que tiene muchísimas posibilidades que no atribuirías a un editor de texto para consola. Por ejemplo, el indentado automático de código Pascal, Java, C, o cualquier lenguaje para el que haya escrito un módulo para Emacs de asistencia a la programación, nos ofrece posibilidades de trabajar con CVS, enviar correo electrónico, y un largo etcétera de posibilidades. Como anécdota cabe contar, para que te hagas una idea, el manual de GNU Emacs, en formato ASCII ocupa cerca de 1.1 MB.

Hablemos de cómo manejarse con los menús de Emacs. Existen cientos de combinaciones de teclas en Emacs que nos permiten hacer cualquier cosa sin ver un menú. Los usuarios expertos de Emacs valoran esta posibilidad, pues a la hora de escribir con prisas, un menú puede ser algo muy incómodo. Pero para nosotros que estamos empezando, debemos recordar: la tecla `F10` es nuestra amiga.

Esta tecla nos da acceso a todos los menús de Emacs, menu, archivo, edición, cambio entre las distintas ventanas de edición de texto, etc.

Empecemos viendo cómo editar un fichero de texto básico. En la consola ponemos:

```
$ emacs hola.c
```

y escribe un programa común y corto:

```
#include <stdio.h>
int
main()
{
  printf("\nHola Mundo\n\n");
}
```

Como podrás apreciar, Emacs hace retroceder el cursor al cerrar los corchetes y los paréntesis, para indicarnos dónde los abrimos y tener una referencia de cuáles quedan aún por cerrar. Probemos ahora a guardar nuestro pequeño programa C. Para ello pulsamos `F10` y una vez pulsada `F10` vemos que la tecla `B` nos daría acceso al menú buffers (que no son otra cosa que las distintas ventanas que tenemos abiertas), la `F` nos daría acceso al menú `files` (el cual hace prácticamente lo mismo que el menú archivo de cualquier editor de texto, etc... y la tecla `C` nos daría acceso, si las tenemos instaladas, a las posibilidades que ofrece emacs para la edición de código en C. Como sólo queremos guardar, pulsamos después de `F10`, `S`. Ya lo tendríamos guardado. Otra cosa muy importante, en cualquier programa es saber salir. Esto se hace con `F10`, `F` y a continuación la tecla `E`. Nos pregunta, si no lo hemos hecho ya, que si deseamos guardar. Escribimos `yes` (hay que escribirlo entero) o `no`, y a continuación nos pregunta si realmente queremos salir, a lo cual ahora sí, responderemos `y` para `sí` o `n` para `no`.

Emacs es famoso por sus innumerables combinaciones de teclas para realizar cualquiera de las operaciones que ofrece. Algunas de las combinaciones de teclas para moverse cómodamente por los documentos son:

Algunas teclas para eliminar texto:

Hay que añadir un truco bastante útil que es la combinación de teclas `C-u`, la cual nos permite repetir cualquier comando n veces.

Por ejemplo, si quisiéramos avanzar 24 palabras, utilizaríamos la secuencia de teclas `C-u 24 A-f`

Otra función muy básica también es la de buscar y reemplazar texto. Esto puede hacerse cómodamente con la combinación `C-s`, y a continuación poniendo que queremos buscar y pulsando `Enter`. Una vez encuentre la primera coincidencia, puede seguir buscándose el mismo patrón pulsando de nuevo simplemente `C-s`.

A-f	Sitúa el cursor al principio de la palabra siguiente.
A-b	Sitúa el cursor al principio de la palabra anterior.
C-a	Sitúa el cursor al principio de la línea actual.
C-e	Sitúa el cursor al final de la línea actual.
A-a	Retrocede el cursor hasta el principio de la frase.
A-e	Avanza el cursor hasta el final de la frase.
Home	Nos permite situarnos al principio del "buffer".
End	Nos permite situarnos al final del "buffer".
C-l	Sitúa el cursor justo al principio de la línea del medio de la pantalla

Tabla 4.2: Combinaciones de teclas para moverse con GNU Emacs

A-d	Elimina la palabra que sigue al cursor.
A-delete	Elimina la palabra anterior al cursor.
C-k	Elimina la línea actual entera.
A-k	Elimina la frase siguiente
C-x + BackSpace	Elimina la frase anterior

Tabla 4.3: Combinaciones de teclas para borrar texto con GNU Emacs

Podemos saber en todo momento qué estamos haciendo fijándonos en la línea inferior de la pantalla de Emacs.

Para reemplazar trozos de texto, cosa también de supervivencia, podemos hacerlo fácilmente de la siguiente forma:

1. Pulsamos F10
2. Pulsamos la S, que corresponde al menú Search
3. Nos sale el siguiente menú:

```
Possible completions are:
S==\frq Search...                    R==\frq Regexp Search...
B==\frq Search Backwards...          0==\frq Regexp Search Backwards...
1==\frq Repeat Search                2==\frq Repeat Regexp
3==\frq Repeat Backwards             4==\frq Repeat Regexp Backwards
5==\frq Bookmarks                    F==\frq Find Tag...  (M-.)
Q==\frq Query Replace...  (M-%)      6==\frq Query Replace Regexp...
```

Pulsamos Q y nos dice:

Query replace: donde escribiremos lo que queremos buscar para ser reemplazado, y pulsamos Enter.

Luego, emacs nos pregunta: query replace with: donde escribiremos el texto con el cual queremos sustituir, y pulsaremos enter.

En este menú, encontramos también una serie de comandos, como query regexp, query replace regexp, etc., que aunque no entraremos en ellos, son muy interesantes, pues nos permiten buscar, no ya patrones de texto concretos, sino un tipo de búsqueda más avanzada por medio de expresiones regulares (**regular expressions** en inglés), esto es, "todas las palabras que empiecen por c y terminen por j" o "todas las mayúsculas cambiarlas por minúsculas" en el caso de query replace regexp.

Insertando un fichero de texto en nuestro documento:

Para los amantes del Cut & Pasting (un deporte muy extendido en algunos círculos de programadores), aquí va un comando que nos permite introducir un documento de texto dentro de otro:

Pulsamos C-x, seguido de la i, para insertar un fichero en la posición actual del cursor. Entonces, en la línea de comandos de emacs, aparecerá el directorio actual.

Escribimos la ruta completa al archivo y aparecerá en la posición actual del cursor.

4.3. Joe

El primer editor que suele aprenderse en Linux es Joe, por ser muy sencillo y rápido. Puede usarse para editar cualquier fichero, pero aquí trataremos su uso básico, lo justo para editar pequeños textos, ficheros de configuración, pequeños programas. Todos estos ficheros pueden editarse con más comodidad con editores más potentes como Emacs o VIM pero para ello es necesario aprender a usarlos primero, lo cual puede no resultar tan sencillo como aprender Joe.

Veremos sólo un par de comandos de Joe, simplemente como editar un fichero existente o nuevo, guardarlo sin salir, salir sin guardarlo y salir guardándolo. Para obtener más información sobre otras opciones del editor existe el comando de ayuda que explicaremos luego.

El manejo de `Joe` se basa en combinaciones de teclas con la tecla `Control`. Denotaremos por `C-x y` a la combinación de teclas que se obtiene al pulsar la tecla `Control`, seguidamente (sin soltar la primera) pulsar la tecla `x` y después (soltando las teclas anteriores) pulsar la tecla `y`. Veamos un rápido ejemplo del uso de este editor. En un terminal ejecuta:

```
$ joe prueba.txt
```

Escribe una frase sencilla, tal como:

```
Si algo funciona, no lo toques.
```

Para guardar el fichero utilizamos la combinación de teclas `C-k s` y entonces `Joe` te preguntará el nombre con el que quieres guardar el fichero.

```
Name of file to save (^C to abort): hola.txt
```

Si pulsamos ahora ahora `C-c` simplemente cancelamos la orden de guardar el fichero, pero no perdemos su contenido. Para guardar el fichero pulsamos `Enter`, modificando el nombre del fichero si lo deseamos. Ahora añadimos otra línea:

```
Si algo funciona, no lo toques.
Si algo funciona, y no sabes por qué, úsalo siempre.
```

Ahora saldremos del editor **guardando los cambios**, mediante la combinación de teclas `C-k x`. De esta manera volveremos al prompt del sistema siendo informados de que el fichero ha sido guardado.

```
File prueba.txt saved.
$
```

Como última maniobra, abrimos de nuevo el fichero, borramos la última línea y salimos sin guardar los cambios.

```
$ joe prueba.txt
```

Boramos la última línea usando las teclas de borrado habituales, y pulsamos `C-c` para salir sin guardar los cambios. El editor nos pedirá confirmación antes de salir. Podemos responder que sí queremos salir pulsando `y`, o por el contrario cancelar la maniobra pulsando `n` o `C-c`. Decimos que sí (`y`) y volvemos al prompt del sistema.

```
Lose changes to this file (y,n,^C)?
File hola.txt not saved.
$
```

Esto es lo mínimo que debes saber para editar con `Joe`, y con esto será suficiente aquí pues en lo sucesivo aprenderás a hacer operaciones más avanzadas con otros editores. Para conocer sobre otras opciones de este editor, se puede utilizar la opción `C-k h`.

TEMA 5

Aplicaciones para Internet

5.1. Navegadores de la World Wide Web

Durante muchos años Netscape Communicator 4 fue el único navegador multiplataforma real, dando cobertura a muchos de los distintos UNIX comerciales existentes. Puesto que Linux no podía ser menos, casi desde que Linux tiene interfaz gráfico ha existido una versión del navegador de Netscape para este sistema operativo.

Netscape Communicator 4 proporciona soporte para navegación de páginas web con JavaScript y Flash 5, y permite visualizar documentos PDF dentro del navegador (mediante un plugin para el Adobe Acrobat Reader). También nos permite gestionar el correo electrónico y componer páginas web.

Los Linuxeros siempre hemos considerado que el navegador de Netscape consumía demasiados recursos en Linux, además de tener bastantes problemas de estabilidad. Debido a éste y a otros factores importantes, como fueron la forma de competir con la casa *Microsoft Corporation*, *Netscape Communications Corporation* llegó a la sana conclusión de que la mejor manera de mantener su navegador en el mercado era liberando su código fuente. Así nació el proyecto Mozilla.

Sin embargo, Mozilla sigue siendo un navegador excesivamente pesado para un número importante de máquinas. Dentro de la comunidad del Software Libre, se alzaron voces en contra de ese desperdicio de recursos, proponiendo la creación de navegadores alternativos. Aquí listamos algunas de las alternativas que podemos encontrar en el área de los navegadores web dentro del Software Libre:

Chimera 2 Navegador web para las X. Simple y rápido, todavía se encuentra en un temprano estado de desarrollo por lo que cuenta con numerosos errores.

Netscape Communicator Bajo dicho nombre podemos encontrar el navegador original de Netscape. La última versión es la 4.77.

Dillo Navegador multiplaforma pequeño, estable y rápido. Está basado en GTK pero no requiere dispone de GNOME.

Encompass Navegador libre para GNOME.

Galeon Es un navegador que utiliza el motor de rendering *Genko* de Mozilla para mostrar el contenido de la World Wide Web. Sin embargo, puesto que utiliza las bibliotecas de GNOME y GTK es ligeramente más rápido que Mozilla, y se integra perfectamente con el resto de las aplicaciones GNOME.

Konqueror Gestor de ficheros, navegador web y visor de documentos del KDE.

Links Navegador web para la consola. Incluye soporte para visualizar tablas, marcos y descargas en segundo plano.

Lynx Navegador web para la consola.

Mozilla Es un sofisticado navegador gráfico de la World Wide Web que soporta un gran número de tecnologías, como por ejemplo soporte para HTML 4.0, CSS 2, JavaScript y Java. Además de poder ser utilizado como un sencillo visor de HTML. Mozilla está basado en parte del código de los navegadores Netscape Communicator y Netscape Navigator.

OpenOffice Suite ofimática que incluye un buen navegador web.

W3m Visor de la World Wide Web para la consola. Dispone de un excelente soporte para tablas y marcos.

Bueno, seguro que en el momento de leer este apartado, habrán surgido nuevos navegadores web dentro del mundillo del Software Libre.

5.1.1. Mozilla

Mozilla es probablemente el más completo de los navegadores multiplataforma. Esto es debido a que implementa soporte para un gran número de tecnologías de la World Wide Web, y se ciñe rigurosamente a los estándares del W3C (nombre con el que se conoce al *World Wide Web Consortium*, que es el organismo encargado de la estandarización de las diferentes tecnologías presentes en la World Wide Web).

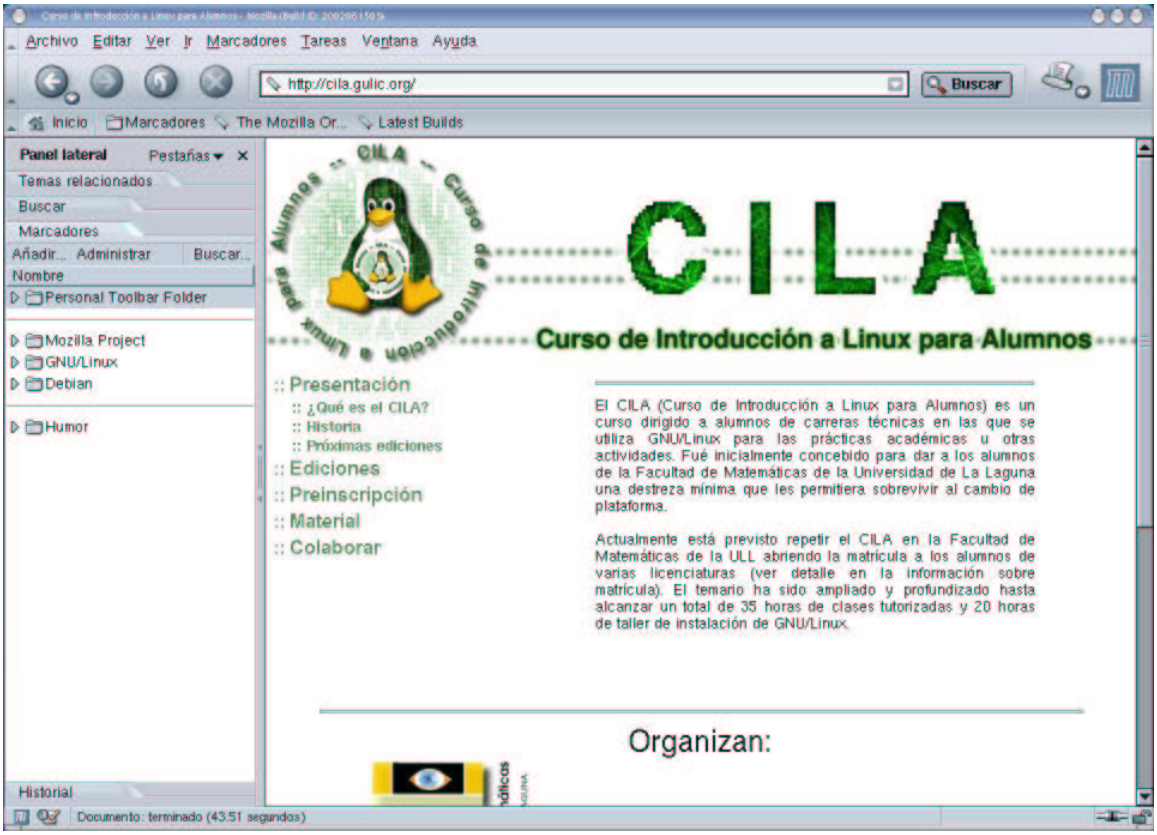


Figura 5.1: Ventana del navegador web Mozilla

Para centrarnos en el manejo de Mozilla empezaremos mirando la Figura 5.1. En ella podemos observar la clásica ventana de navegación de Mozilla que es semejante a la de otros muchos navegadores. A continuación enumeramos los elementos de la ventana de arriba a abajo y de izquierda a derecha:

1. *Barra de menús*
2. *Barra de herramientas de navegación*
3. *Barra de herramientas personales*
4. *Panel lateral*
5. *Área de visualización de la navegación*
6. *Barra de tareas*

La visualización de la mayor parte de estos elementos puede activarse o desactivarse desde el menú `VER` → `BARRA DE HERRAMIENTAS` de la barra de menú del programa.

Adicionalmente disponemos de una barra denominada *barra de componentes*. Dicha barra se muestra como unos pequeños iconos a la izquierda de la barra de tareas (parte inferior de la ventana del programa). Dicha barra nos permite lanzar de forma sencilla y rápida algunas de las otras herramientas de Internet que acompañan al navegador web Mozilla. Entre dichas herramientas disponemos de un editor de HTML y de un lector de noticias y de correo electrónico.

5.1.1.1. Navegación básica

Para navegar por la World Wide Web basta con introducir la dirección de la máquina o recurso al que deseamos acceder en la barra de herramientas de navegación. La misma barra dispone a la izquierda de botones para avanzar o retroceder a través de las páginas visitadas, recargar la página actual, o detener la descarga. A la derecha de la barra disponemos de un botón de acceso rápido al menú de impresión, con el que podemos imprimir la página actual. Para simplificar el aprendizaje, si dejamos el puntero del ratón sobre cualquiera de los botones durante unos segundos la aplicación nos informará de la función de cada uno.

Además de las funciones básicas la barra de herramientas de navegación nos permite realizar búsquedas de términos en Internet. Para ello basta con introducir las palabras a buscar en la propia barra y a continuación pulsar en el botón `BUSCAR`. El navegador consultará al buscador que tengamos configurado (por defecto se trata del de *Netscape Network*) y nos mostrará los resultados. El buscador utilizado puede ser configurado en `EDITAR` → `PREFERENCIAS...` → `NAVIGATOR` → `BÚSQUEDA EN INTERNET`. Mientras que el contenido de la propia barra de herramientas de navegación puede ser configurado en `EDITAR` → `PREFERENCIAS...` → `NAVIGATOR`. Nos ocuparemos de describir el cuadro de diálogo `PREFERENCIAS` y las opciones de configuración de Mozilla más a delante.

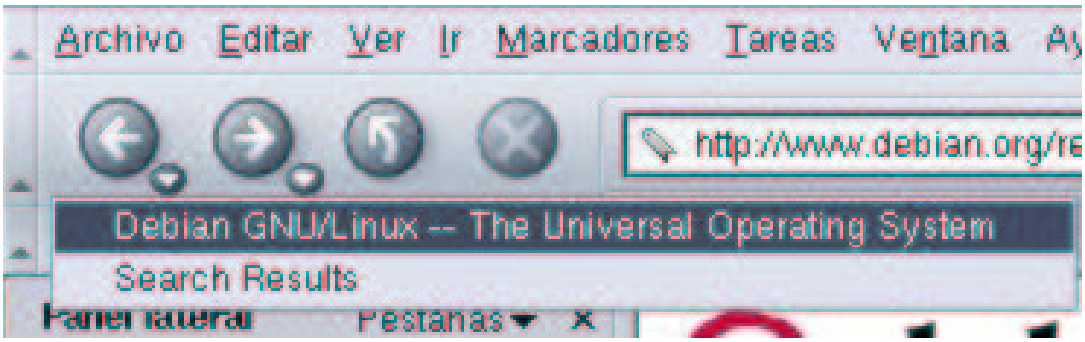


Figura 5.2: Menú del botón de retroceso de la barra de herramientas de navegación

Algunos de los botones de la barra de herramientas de navegación disponen de un pequeño icono con forma de flecha en la parte inferior-derecha de los mismos. Dicho icono suele desplegar un menú con opciones adicionales. En el caso particular de los botones de avance y retroceso dicho menú nos permite elegir a que dirección, de entre las ya visitadas, queremos avanzar o retroceder (Figura 5.2). Hay que tener en cuenta que si no existiera el menú tendríamos que retroceder o avanzar por las páginas visitadas de una en una.

Durante nuestra visita a la World Wide Web la barra de tareas suele mantenernos informados de las acciones que realiza el navegador. Por ejemplo, nos informa de si estamos detenidos y de cuanto se tardó en descarga la página actual, o de si estamos descargando alguna página.

Además, la barra de tareas dispone de una serie de iconos en la parte derecha con el objetivo de mantenernos informados del estado de la conexión. Si dejamos unos segundos el puntero del ratón sobre dichos iconos seremos informados de su significado, e incluso podremos realizar alguna acción relacionada con dicha información.

Trabajar sin conexión Mientras disponemos de conexión a la red y navegamos, Mozilla descarga las páginas que vamos visitando y las almacena en el disco duro en lo que se denomina una caché. La existencia de está caché es importante puesto que si visitamos varias veces una misma dirección Mozilla no tiene necesidad de volver a repetir la descarga. En su lugar nos proporciona directamente la página almacenada en la caché. En ocasiones no disponemos de conexión a red por lo que sería deseable que Mozilla no intentará descargarse la páginas desde la red y nos mostrara directamente la copia en la caché. También es posible que aun habiendo conexión estemos interesados en que Mozilla no haga uso de ella. Ese modo de trabajo que andamos buscando es el denominado *trabajo sin conexión*. Existe un icono que representa dos enchufes desconectados que nos indica que estamos en dicho modo. Si los enchufes están conectados significa que

estamos haciendo uso de la conexión de red, es decir *trabajo con conexión*. El paso de un modo de trabajo a otro se puede realizar pulsando con el ratón sobre el icono, o seleccionando el elemento de menú `ARCHIVO` → `TRABAJAR SIN CONEXIÓN`.

Cookies En ocasiones algunos servidores de la World Wide Web requieren que los navegadores que los acceden almacenen cierta información. En muchos casos se trata de información sobre nuestras preferencias que dichos servidores han recopilado, y que desean que guardemos para volver a reclamarla cuando nos volvamos a conectar a sus páginas. Este tipo de comportamiento puede ser un agujero de seguridad en potencia (o al menos un problema de privacidad) por lo que Mozilla dispone de un filtro de cookies que nos permite decidir en que servidores confiamos y en cuales no. Si nuestro navegador ha aceptado una cookie para ser almacenada podremos observar el icono de una *galleta*. Pulsando sobre la misma podremos averiguar la información que contiene, así como configurar el filtro de cookies frente a posibles peticiones futuras.

Seguridad Es importante recordar que todo lo que recibimos o enviamos desde o hacia la World Wide Web es fácilmente interceptarle. Cuando accedemos a páginas donde la seguridad de las comunicaciones es de vital importancia dicha comunicación se hace cifrando los datos. Si disponemos de un icono con un candado abierto en nuestro navegador significa que todo lo que hagamos en Internet puede ser observado por otros. Sin embargo, si dicho candado está cerrado es porque las comunicaciones son seguras, por lo que tenemos garantías de que nuestros datos no pueden ser fácilmente interceptados. Al pulsar sobre dicho icono obtendremos la información de seguridad para la pagina actual.

Al igual que en muchos otros navegadores Mozilla dispone de un menú de contexto activable pulsando con el botón derecho del ratón sobre alguno los elementos del área de visualización. Dicho menú nos permite, por ejemplo, recargar la página actual, descargar un enlace, o abrir un enlace en una ventana diferente, entre otras muchas posibles acciones.

Durante la navegación por la World Wide Web resulta habitual disponer de varias ventanas de Mozilla abiertas en páginas diferentes. Sin embargo, todas esas ventanas están ligadas a un mismo proceso en ejecución del navegador Mozilla. Es importante conocer esto puesto que **si seleccionamos el elemento de menú** `ARCHIVO` → `SALIR` **TODAS las ventanas de** Mozilla **se cerraran**, al terminar el proceso que las gestionaba. Si por el contrario deseamos cerrar únicamente una de las ventanas en particular, debemos seleccionar `ARCHIVO` → `CERRAR` o utilizar el botón correspondiente de la barra de título del marco de la ventana.

5.1.1.2. Pestañas

Una de las innovaciones tomada de otros navegadores e introducida en Mozilla es el uso de las *pestañas* durante la navegación. Tanto si seleccionamos la opción `ARCHIVO` → `NUEVO` → `PESTAÑA DE NAVIGATOR`, como si seleccionamos en `ABRIR EN UNA PESTAÑA NUEVA` cuando pulsamos con el botón derecho en un enlace, se nos abre una nueva área de visualización dentro de la misma ventana de Mozilla. Podemos disponer de tantas áreas como deseemos y en cada una visualizar una página diferente.

En la Figura 5.3 podemos observar cómo cada área está representada por una pestaña en la parte superior del área de visualización. Seleccionando un pestaña u otra podremos navegar por una página u otra. Cuando deseemos cerrar la pestaña seleccionada bastará con que pulsemos en botón a la derecha de todas las pestañas. También podemos arrastrar un enlace en la pagina web del área de visualización sobre una de las pestañas. Con ello conseguimos que en dicha pestaña se cargue el recurso del la World Wide Web al que apunta dicho enlace (por ejemplo, una página web, una imagen, etc).

Algunos aspectos del comportamiento de las pestañas pueden ser configurados en `EDITAR` → `PREFERENCIAS...` → `NAVIGATOR` → `PESTAÑAS`.

5.1.1.3. Marcadores

Mozilla nos permite almacenar de forma ordenada y clasificada las direcciones de los recursos de la World Wide Web que más nos interesan. Dicho almacenamiento se hace en forma de lo que se denominan *marcadores*.

Añadir la dirección de la página actual como marcador es tan sencillo como seleccionar `MARCADORES` → `AÑADIR A MARCADORES`. También podemos pulsar con el botón derecho del ratón sobre un enlace y seleccionar la opción correspondiente para añadirlo a los marcadores.

En todo caso el marcador se crea siempre al final de la lista. Si deseamos tener un control más fino para, por ejemplo, añadir el marcador a una carpeta determinada, podemos utilizar el menú `MARCADORES` → `ARCHIVAR MARCADOR...` Dicha opción del menú muestra un cuadro de diálogo que nos permite seleccionar el nombre del nuevo marcador, crear carpetas de

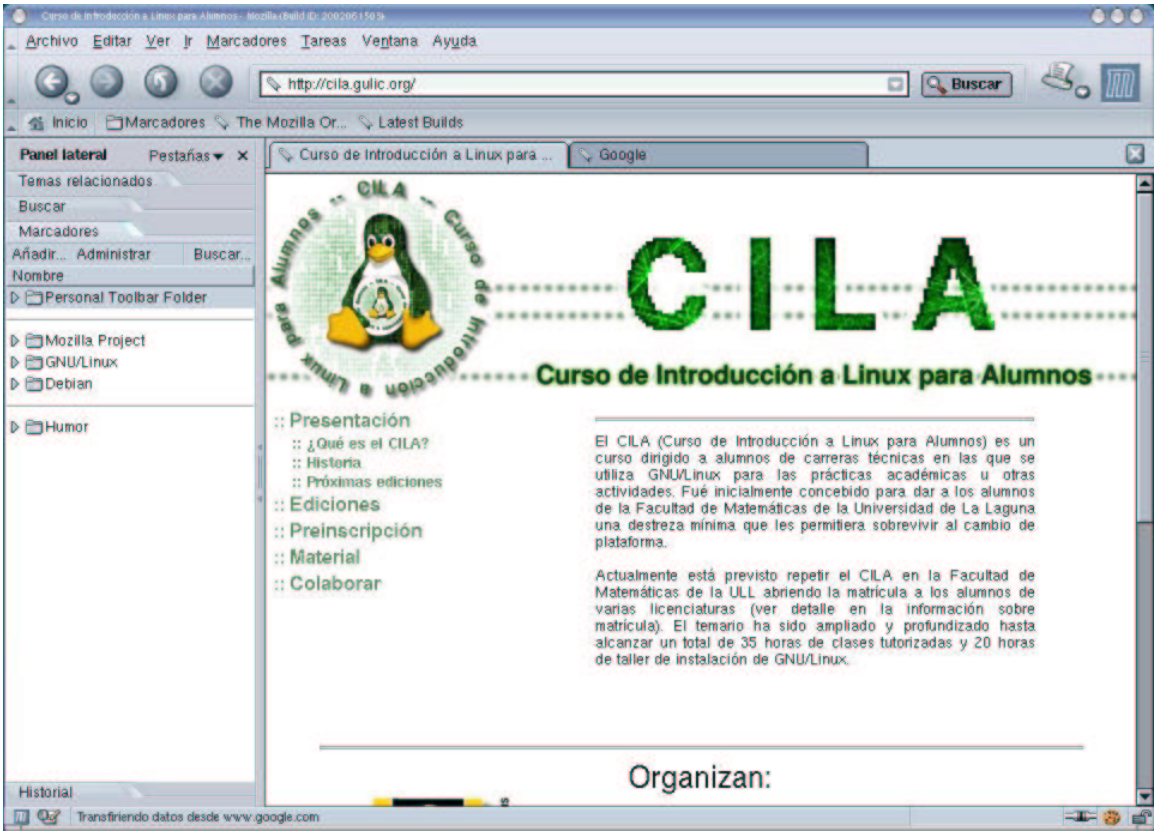


Figura 5.3: Navegación con pestañas en Mozilla

marcadores, y elegir en que carpeta deseamos guardarlo. Los marcadores así creados son accesibles en forma de menús en el menú `MARCADORES` de la barra de menús.

En ocasiones es necesario realizar sobre los marcadores tareas de administración mucho más avanzadas. La opción `MARCADORES` → `ADMINISTRAR MARCADORES` despliega un cuadro de diálogo (Figura 5.4) que nos permite manipular los marcadores a nuestro antojo. Podemos movernos por el árbol de marcadores y copiarlos o pegarlos con ayuda de ratón. También podemos alterar sus propiedades utilizando el menú de contexto que se despliega con el uso del botón derecho de nuestro ratón.

El administrador de marcadores dispone de opciones para ordenar los marcadores según diversos criterios, para realizar búsquedas, y para exportar o importar hacia o desde los marcadores de otros navegadores. Además de carpetas, el administrador de marcadores nos permite crear separadores que se utilizan para separar elementos dentro de los menús. Al igual que con el resto de las ventanas de Mozilla debemos evitar utilizar la opción `ARCHIVO` → `SALIR` puesto que ésta cierra todas la ventanas del programa.

Entre las carpetas de marcadores existe una con un significado especial. La carpeta `PERSONAL TOOLBAR FOLDER` representa la barra de herramientas personales, que está situada debajo de la barra de herramientas de navegación. Todos los marcadores insertados en dicha carpeta aparecerán automáticamente como botones en la citada barra de herramientas.

5.1.1.4. Panel lateral

A la izquierda del área de visualización está el *panel lateral* o *sidebar* de Mozilla. Dicho panel dispone de una serie de pestañas con funciones que ayudan a la navegación. Utilizando el menú `PESTAÑAS` del panel lateral podemos decidir que pestañas vemos de entra la disponibles. También podemos personalizar nuestro panel añadiendo nuevas pestañas con nuevas funciones descargadas desde Internet.

Para que no interfiera con la navegación podemos modificar el ancho del panel pulsando con el ratón en la barra que lo separa del área de visualización. Si hacemos una pulsación sencilla sobre la pequeña marca del centro podremos plegar y desplegar el panel de forma rápida. También podemos ocultarlo de forma permanente pulsando en la *equis* de la parte superior-derecha del panel.

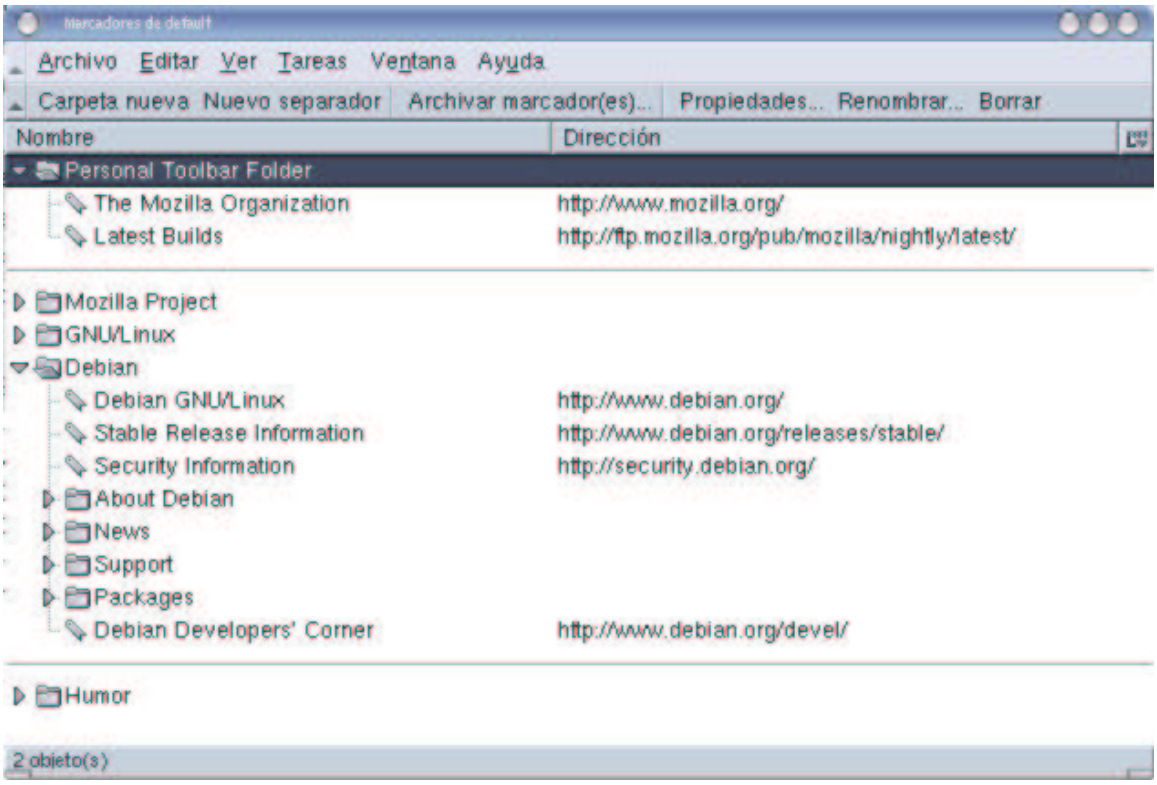


Figura 5.4: Cuadro de diálogo ADMINISTRAR MARCADORES

Por defecto, en el reducido espacio del panel lateral podemos consultar el historial y los marcadores, y realizar búsquedas por palabras o por temas relacionados en Internet. Sin embargo, como ya hemos comentado, dichas funcionalidades son completamente ampliables.

5.1.1.5. Preferencias

Muchas de las características de Mozilla son personalizables. En la Figura 5.5 podemos observar el cuadro de diálogo de preferencias de Mozilla. A dicho cuadro podemos acceder desde la opción EDITAR → PREFERENCIAS...

Las preferencias de Mozilla están clasificadas en categorías. Al seleccionar una categoría podemos observar en el lado derecho de la ventana las opciones de configuración relacionadas. Además, cada categoría puede contener subcategorías cuya lista se despliega o se pliega con una pulsación simple del ratón sobre dicha categoría.

APARIENCIA Preferencias tales como los colores, o los tipos de letras utilizados por el navegador en la visualización de las páginas web son establecidas en esta categoría. También podemos seleccionar el idioma y el *tema* de la aplicación. Seleccionar un nuevo *tema* cambia el aspecto visual de los botones, cuadros de diálogo, menú, barras de herramientas y otros objetos. En Internet resulta sencillo encontrar todo tipo de *temas* para personalizar nuestro navegador.

NAVIGATOR Las preferencias específicas del navegador son establecidas en esta categoría. Entre ellas contamos con la dirección de la página de inicio, el buscador por defecto, o la administración del historial de direcciones. En ocasiones el contenido de las páginas web está disponible en varios idiomas. En esta categoría podemos establecer los idiomas en los que preferimos ver dichas páginas web.

COMPOSER En esta categoría podemos encontrar toda una serie de opciones relacionadas con el editor de páginas web que viene con Mozilla.

MAIL & NEWS En caso de haber instalado el lector de noticias y de correo electrónico de Mozilla, podemos acceder a esta categoría para configurarlo.

PRIVACIDAD Y SEGURIDAD Las preferencias de privacidad y seguridad determinan las características del filtrado de *cookies* e imágenes. Además, nos permite configurar el navegador para que guarde contraseñas o datos de formularios que utilizamos habitualmente. También nos permite gestionar los protocolos y certificados utilizados durante las conexiones

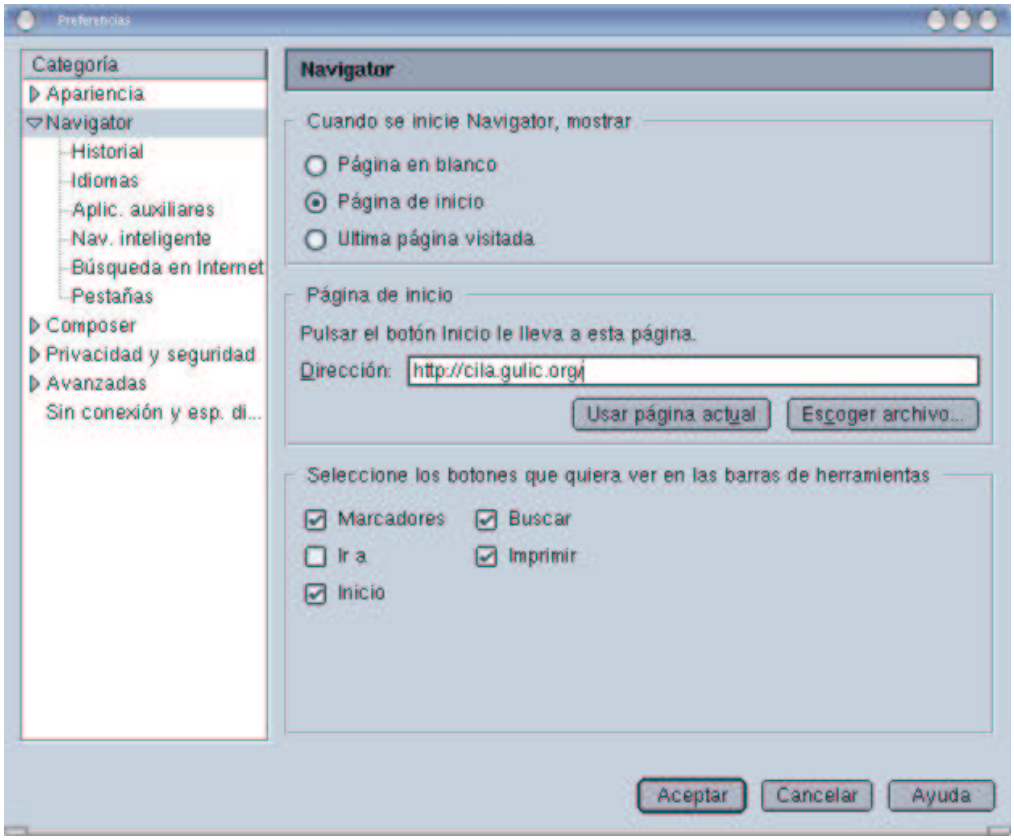


Figura 5.5: Cuadro de diálogo de preferencias de Mozilla

encriptadas. Debido a que la mayor parte de esta información confidencial es almacenada en el disco duro, Mozilla permite la encriptación de dichos datos bajo una contraseña maestra. Con ello se consigue que esa información no pueda ser accesible a terceros.

AVANZADAS En esta categoría se configuran aspectos avanzados como el soporte de Java y JavaScript, la configuración del caché, o el acceso a la red a través de proxies. La opciones de esta categoría nos permiten, por ejemplo, establecer qué permitimos y qué no permitimos hacer a las páginas web a las que accedemos.

SIN CONEXIÓN Y ESPACIO EN DISCO DURO Configuración sobre el comportamiento del navegador en los modos de trabajo con conexión y sin conexión.

Podemos utilizar la ayuda para obtener información más detallada de cada una de las opciones.

5.2. Transferencia de ficheros (FTP)

FTP (File Transfer Protocol) es un protocolo que se utiliza para transferir información, almacenada en ficheros, de una máquina remota a otra local, o viceversa. Para poder realizar esta operación es necesario conocer la dirección IP o el nombre de la máquina a la que nos queremos conectar para realizar algún tipo de transferencia. Es fundamental distinguir entre máquina local y máquina remota:

Máquina local Es aquella desde donde nos conectamos para hacer la transferencia, es decir, donde ejecutamos el cliente de FTP.

Máquina remota Es aquella a la que nos conectamos para transferir la información.

5.2.1. Cliente estándar

El cliente estándar de FTP es una aplicación de la consola que ejecuta a través del comando `ftp`. A continuación veremos cómo se utiliza.

5.2.1.1. Inicio de sesión FTP

Para realizar transferencias de ficheros por protocolo FTP se establecen conexiones (sesiones) entre la máquina local y la remota. Estas sesiones comienzan por la validación del usuario y prosiguen con las transferencias. Finalmente la sesión se cierra. Veamos un ejemplo:

```
$ ftp euler
Connected to euler.fmat.ull.es.
220 ProFTPD 1.2.0pre10 Server (Debian) [euler.fmat.ull.es]
Name (euler:miguev):
```

El servidor nos preguntará un nombre de usuario, y seguidamente la contraseña. El nombre que daremos debe ser una cuenta de usuario válida en el servidor al que intentamos acceder, y la contraseña lógicamente debe ser correspondiente a ese usuario. En servidores públicos suele existir una cuenta de acceso anónima sólo para leer (o tal vez una carpeta donde poner cosas pero no leer). Para acceder a un FTP como usuario anónimo se utiliza el nombre `anonymous` y se proporciona la dirección de correo electrónico como contraseña.

Una vez introducido el nombre y la contraseña el servidor nos recibirá y el programa cliente de FTP nos mostrará un prompt (indicador de órdenes), manifestando así que está preparado para ejecutar las órdenes que le demos. A partir de aquí se realizan las tareas deseadas mediante los comandos de FTP que veremos más adelante.

```
$ ftp euler
Connected to euler.fmat.ull.es.
220 ProFTPD 1.2.0pre10 Server (Debian) [euler.fmat.ull.es]
Name (euler:miguev): miguev
331 Password required for miguev.
Password:
230 User miguev logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftz>
```

5.2.1.2. Comandos del FTP

El protocolo FTP dispone de unos comandos estándar suficientes para las operaciones de transferencia de ficheros. A continuación presentamos un resumen de las mismas:

ascii Para hacer la transferencia en formato ASCII (ésta es la opción por defecto).

binary Para hacer la transferencia en formato binario, se utiliza el comando.

! comando Para ejecutar un comando desde el intérprete de comandos en la máquina local.

cd directorio_remoto Para moverse de un directorio a otro en la máquina remota.

delete fichero_remoto Para borrar un fichero en la máquina remota.

delete ficheros_remotos Para borrar varios ficheros en la máquina remota.

get fichero_remoto fichero_local Transfiere el `fichero_remoto` desde la máquina remota a la máquina local, guardándolo con el nombre `fichero_local` en la máquina local.

help Proporciona una lista de los comandos del FTP de la máquina local.

help comando Proporciona información sobre el comando especificado, correspondiente a la máquina local.

lcd directorio_local Para moverse de un directorio a otro en la máquina local.

lcd unidad: Para cambiar de una unidad de disco a otra, en el caso particular de que la máquina local esa un PC con Windows o MS-DOS.

lls directorio_local Para listar el contenido de un directorio en la máquina local.

[**ls—dir** directorio_remoto] Para listar el contenido de un directorio en la máquina remota.

mget lista_ficheros_remotos Transfiere los ficheros listados desde la máquina remota a la máquina local.

mkdir directorio_remoto Para crear un directorio en la máquina remota.

mput lista_ficheros_locales Transfiere los ficheros listados desde la máquina local a la máquina remota.

prompt (Des-)activa el modo interactivo de las transferencias de ficheros múltiples.

put fichero_local fichero_remoto Transfiere el `fichero_local` desde la máquina local a la máquina remota, guardándolo con el nombre `fichero_remoto` en la máquina remota.

pwd Para saber el directorio en el que se está, en la máquina remota.

quit Terminar la sesión actual y finalizar el programa.

rmdir directorio_remoto Para borrar un directorio en la máquina remota.

Como ya hemos comentado, una vez iniciada la sesión podemos utilizar estos comandos para realizar todo tipo de operaciones sobre los ficheros de la máquina remota.

Es importante destacar que por defecto las transferencias se hacen en formato ASCII. Es decir, el protocolo considera que estamos transfiriendo archivos de texto plano, por lo que realiza las conversiones oportunas entre el formato utilizado en la máquina local y el utilizado en la máquina remota. Cuando transferimos archivos que deben ser copiados *tal cual*, es decir, sin alteraciones de ningún tipo, como es el caso de ejecutables, imágenes y demás archivos binarios, es necesario especificar que deseamos el modo binario de transferencia. Para ello ejecutamos el comando `binary` justo antes del comando que da comienzo a la transferencia.

5.2.2. gFTP

Obviamente, el comando `ftp` no es el único programa cliente de FTP disponible. Existen multitud de programas clientes de FTP tanto para la consola como para el entorno gráfico.

gFTP es el cliente FTP del entorno de escritorio GNOME. Se trata, por tanto, de un cliente en modo gráfico diseñado para facilitar el acceso a los recursos FTP. El uso de un cliente en modo gráfico nos permite olvidarnos de los comandos del protocolo FTP. En su lugar todas las operaciones se reducen a sencillas acciones sobre la interfaz gráfica.

En la Figura 5.6 podemos observar la ventana de un gFTP con una conexión a `ftp.es.debian.org`. Para iniciar una sesión de FTP con gFTP debemos recurrir a la barra situada justo debajo de la barra de menús. En ella debemos especificar los datos requeridos para realizar la conexión.

SERVIDOR En este campo debemos escribir el nombre de la máquina remota a la que nos vamos a conectar. En el de la Figura 5.6 fue `ftp.es.debian.org`.

PUERTO Cada servicio que se ofrece en Internet tiene un puerto asociado. El puerto por defecto para el servicio de FTP es el 21, aunque algunas máquinas dan dicho servicio en otro puerto cualquiera. En este campo se debe especificar el puerto en el que la máquina remota está esperando las peticiones de FTP. Si no se indica nada, como es nuestro caso, gFTP asumirá que queremos utilizar el puerto por defecto, es decir, el 21.

USUARIO En este campo debemos indicar el nombre de usuario con el que queremos acceder a la máquina remota. Si el servidor es público, lo más probable que utilizando el nombre `anonymous` podamos acceder a través de la cuenta de usuario anónima.

CONTRASEÑA La contraseña asociada al usuario con el que queremos acceder. En caso de acceder a través de la cuenta de usuario anónima debemos indicar nuestra dirección de correo electrónico.

PROTOCOLO El último campo nos permite indicar el protocolo que vamos a utilizar para nuestra conexión. gFTP no sólo puede utilizar el protocolo FTP para realizar transferencias de archivos. Por ejemplo, podemos utilizar SSH2 para realizar transferencias de archivos encriptadas. Esto garantiza que tanto nuestro datos (nombre de usuario, contraseña, etc) como los de los archivos estén seguros frente a intentos de interceptación de las comunicaciones. Es importante destacar que el protocolo FTP no es un protocolo seguro, en el sentido de que nuestra contraseña y todos los demás datos que viajen por la conexión son fácilmente interceptables.

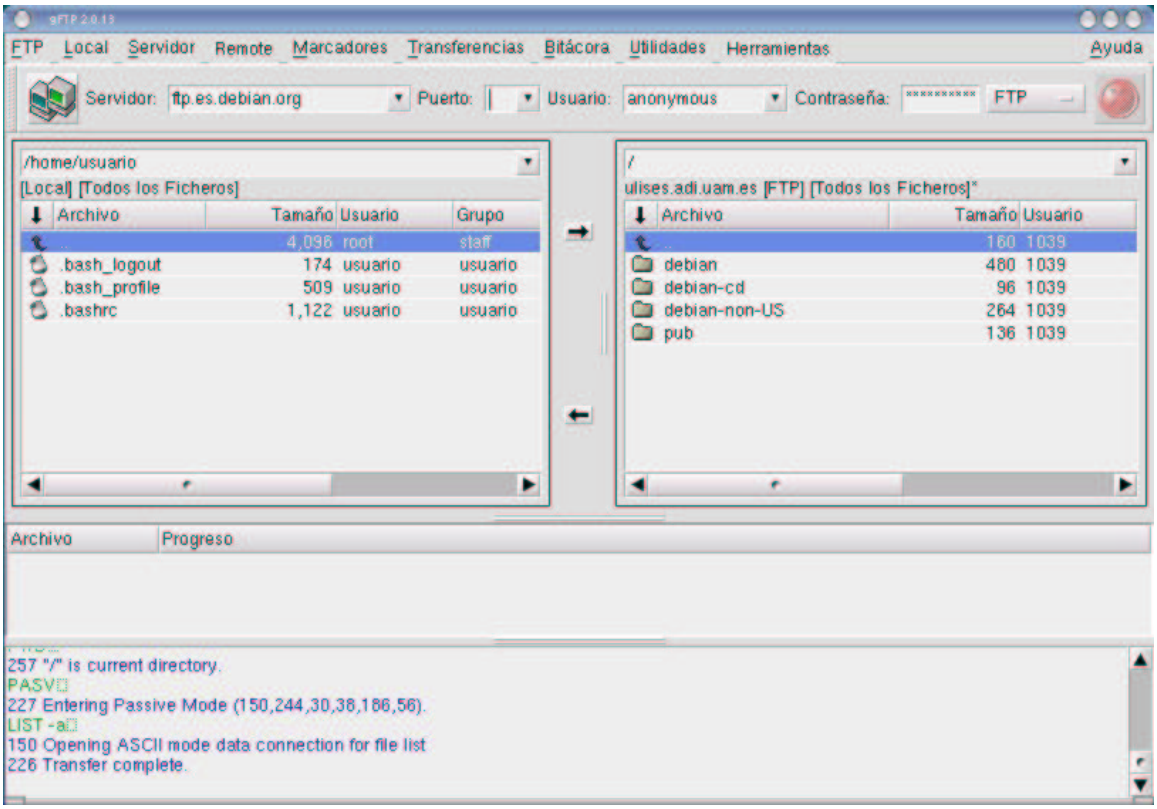


Figura 5.6: Ventana de gFTP en una conexión a `ftp.es.debian.org`

Tras completar los datos podemos pulsar en el botón de la izquierda para iniciar la sesión. gFTP se encarga de enviar los comandos necesarios para iniciar la conexión, evitándonos el engorroso problema de tener que conocerlos. El botón de la izquierda nos permite iniciar una conexión si ésta no ha sido iniciada, o terminarla si ya ha sido iniciada correctamente. Sin embargo, si queremos detener el intento de conexión en curso debemos recurrir al botón de la derecha. Dicho botón se utiliza para abortar cualquier tarea que gFTP esté realizando en el momento actual.

En la parte inferior de la ventana de gFTP podremos observar un registro de la conexión. En *rojo* veremos información propia de gFTP, en *verde* observaremos los comandos que gFTP envía a la máquina remota, y en *azul* podremos ver la respuestas de la máquina remota a dichos comandos. Sea cual sea la tarea que estemos realizando siempre quedará registrada en esa zona de la ventana. Por eso suele ser importante estar atento a ella para saber en todo momento lo que está sucediendo. Utilizando la barra de desplazamiento de la derecha podremos ver los mensajes antiguos, o que hayan pasado demasiado rápido.

El centro de la ventana de gFTP está dividido en dos áreas que podemos utilizar para navegar por el árbol de directorios. El área de la izquierda podemos utilizarla para movernos por el árbol de directorios de la máquina local. Mientras que el área de la derecha, siempre y cuando gFTP esté conectado, podemos utilizarla para movernos por el árbol de directorios de la máquina remota. En ambos lados podemos utilizar el botón derecho del ratón para desplegar un menú de contexto con un amplio conjunto de opciones. Con ellas podemos crear directorios, renombrar archivos, ver y editar archivos, modificar sus atributos, borrar, etc.

Mientras que la doble pulsación sobre una directorio nos permite ver su contenido, una doble pulsación sobre un archivo inicia una transferencia para que el archivo sea enviado al otro lado de la conexión. El mismo efecto conseguimos si seleccionamos uno o más archivos (p. ej. pulsando con el botón izquierdo del ratón sobre el archivo deseado mientras mantenemos pulsada la tecla `Ctrl` o `Shift`) y los arrastramos al área del otro lado. También podemos conseguir el mismo efecto si pulsamos sobre los botones situados en el centro de las dos áreas. Dichos botones nos permiten iniciar la transferencia de archivos en el sentido que más nos convenga.

Justo entre el área donde se registran las tareas realizadas por gFTP y la que utilizamos para explorar los árboles de directorios, tenemos la cola de transferencias. Todas las transferencias iniciadas o pendientes se muestran en dicha área, así como información relacionada con ellas. Utilizando el menú de contexto que se despliega con el botón derecho de nuestro ratón podemos detener o iniciar una transferencia, o alterar el orden de la cola para decidir qué transferencia se iniciará después de que se complete la actual. Por defecto, las transferencias desde máquinas remotas diferentes se realizan en paralelo, mientras

que las transferencias desde una misma máquina remota que se van realizando secuencialmente.

5.3. Acceso remoto (SSH)

En muchas ocasiones resulta interesante acceder a una máquina remota y trabajar como si estuviéramos físicamente frente a ella. Es decir, hablamos de poder ejecutar comandos en dicha máquina sin tener que trasladarnos a escribirlos en su teclado. Este tipo de servicios existe desde los orígenes de Internet pero siempre ha entrañado sus riesgos dado que es necesaria la validación remota del usuario. Para entender de lo que estamos hablando sólo es necesario que nos aproximemos al servicio de TELNET (cuyo cliente suele estar disponible bajo el comando del mismo nombre).

TELNET fue el primer servicio de acceso remoto diseñado para Internet. Básicamente el comando `telnet` toma nuestras pulsaciones de teclado y las transmite por la red hasta el servidor TELNET de la máquina remota a la que estamos conectados. Éste toma la salida a terminal de la aplicaciones que vayamos ejecutando y las envía a nuestro cliente en la máquina local. Ese mecanismo tan sencillo resuelve el problema del acceso remoto; excepto por un pequeño problema. Toda la comunicación se realiza en texto plano. Es decir, el texto se envía tal cual, sin mecanismo alguno de compresión y/o encriptación que altere las cadenas de texto. Todo lo que escribimos (por ejemplo, comandos del intérprete de comandos) y recibimos (por ejemplo, correo, informes, cartas) puede ser visto por quienes intercepten nuestros paquetes. Esto es especialmente crítico durante el proceso de conexión, momento en el que debemos enviar nuestro nombre de usuario y contraseña para realizar la validación remota a la que aludimos anteriormente. Es evidente que en esa etapa nuestra password, y con ella el acceso al servidor bajo nuestro nombre de usuario, queda al descubierto.

A efectos prácticos la mayor parte de los protocolos de Internet se fundamentan en TELNET. Servicios como la World Wide Web (HTTP), el correo electrónico (SMTP, POP, IMAP), las transferencias de ficheros (FTP), la administración de la red (SMNP), etc. utilizan protocolos cuyas especificaciones indican claramente su vínculo con TELNET. Si ya resulta grave que en todas esas situaciones nuestros datos queden al descubierto, más lo es aun cuando hablamos de permitir o denegar el acceso a un servicio tan crítico como es el acceso remoto. Éste suele poner las cosas más fáciles que ningún otro para quien desee atacar el sistema.

Dos son las cosas que debemos recordar de todo lo anterior:

- **En condiciones normales la mayor parte de los datos que enviamos y/o recibimos de Internet están al descubierto.** Es decir, una vez interceptados pueden ser leídos directamente sin requerir ningún proceso intermedio.
- **No debemos utilizar TELNET bajo ningún concepto.** El comando `telnet` puede ser una herramienta muy práctica en algunos casos, pero peligrosa si la usamos para acceso remoto.

La cuestión ahora es cómo podemos resolver estos problemas. La respuesta es utilizando Secure Shell (SSH). Este programa trabaja de forma similar a TELNET solo que encriptando tanto la información que es transmitida a la máquina remota como la que es enviada por ésta. El resultado es que aunque la comunicación pueda ser interceptada los datos resultarán ininteligibles. En realidad SSH es un paquete de comandos relacionados con las transacciones seguras de información en la red, que utiliza diferentes mecanismos para garantizar esa seguridad. De todos ellos el comando `ssh` esconde el programa de acceso remoto del cual la forma más sencilla de utilizarlo es:

```
$ ssh usuario@máquina_remota
```

Por ejemplo si el usuario `cila` quiere acceder al servidor `gulic.org` ejecutaría lo siguiente:

```
$ ssh cila@gulic.org
```

La primera vez que accedamos a una máquina el programa nos mostrará su *huella dactilar* y pedirá que confirmemos que queremos establecer la conexión. Esto permite utilizar políticas de seguridad en las que podamos verificar la autenticidad de la máquina a la que nos conectamos, evitando que pueda haber sido suplantada por otra. Si estamos seguros de la autenticidad del sistema remoto debemos contestar que sí. La huella dactilar es almacenada y vinculada a la dirección de la maquina remota, por lo que nunca más recibiremos una mensaje de verificación como el anterior. Exceptuando el caso en que la máquina haya cambiado, y con ello su huella dactilar, situación en la que probablemente estemos ante una posible suplantación. Si todo va bien `ssh` nos pedirá la password del usuario. En caso de ser autentificados dispondremos de acceso remoto sobre el sistema. Si al ejecutar el comando no especificamos el nombre de usuario (`ssh gulic.org`) el programa utilizará por defecto nuestro nombre en la máquina local.

A la hora de trabajar con `ssh` debemos tener algunas cosas en cuenta. En caso de duda podemos recurrir a las páginas del manual (`man ssh`) donde encontraremos detallada información así como referencias a las otras herramientas del paquete SSH. Si acaso destacar que debido a que el carácter `~` tiene un significado especial para el `ssh`, si queremos escribirlo en la máquina remota tendremos que pulsar `~~` en nuestro teclado.

Aunque `ssh` garantiza el acceso remoto seguro no proporciona por sí solo la transferencia segura de archivos. Para ello se utiliza el comando `scp` que tiene la siguiente forma:

```
$ scp usuario@máquina_origen:archivo_origen \
> usuario@máquina_destino:archivo_destino
```

El cual copia `archivo_origen` desde la `máquina_origen` hasta el `archivo_destino` en la `máquina_destino`. Si no se especifica alguno de los nombres de máquina, el `scp` asume que estamos hablando del sistema local. El siguiente comando copia el archivo `mi_archivo` desde la máquina local hasta `gulic.org`:

```
$ scp mi_archivo gulic.org:
```

Mientras que el siguiente comando hace lo contrario:

```
$ scp gulic.org:mi_archivo .
```

Si al especificar la ruta de archivo en la máquina remota lo hacemos de forma relativa (o sea sin usar `/`) la ruta será relativa a nuestro directorio personal en la maquina remota. Esto significa que el comando anterior copia `mi_archivo` desde nuestro directorio personal en `gulic.org`. Sin embargo, el siguiente comando copia el fichero `/tmp/mi_archivo` (ruta absoluta) en el mismo servidor.

```
$ scp gulic.org:/tmp/mi_archivo .
```

Otro comando interesante es `sftp` que nos proporciona los mismos servicios que `ftp` solo que de forma segura. Sin embargo aún no está disponible en todos los sistemas.

Una de las preguntas más habituales de los usuarios de Linux es si pueden acceder a su servidor SSH en Linux desde una máquina local en otro sistema operativo. La respuesta es que SSH es un protocolo estándar por lo que todo depende de la disponibilidad de un cliente para su plataforma. En la actualidad existen versiones libres para la mayor parte de sistemas operativos. De entre ellas destacaremos *PUTTY* como uno de los mejores clientes TELNET/SSH para sistemas Microsoft® Windows®.

5.4. Correo electrónico

5.4.1. Mutt

Mutt es un programa cliente de correo electrónico, lo que en inglés se denomina un MUA (Mail User Agent, agente de correo del usuario). Es un programa "de consola", lo que significa que no necesita un entorno de ventanas para ejecutarse. Al igual que otros programas basados en pulsasiones de teclas, Mutt resulta ser poco intuitivo al principio. Afortunadamente, se encuentra traducido al castellano y eso ayuda bastante.

Vamos a usar Mutt para familirizarnos un poco él, verás que es simple. Abrimos una ventana de emulador de terminal y ejecutan el comando `mutt`.

```
$ mutt
```

Si nos fijamos en la primera línea de la pantalla vemos que aparecen listadas una serie de teclas con sus acciones asociadas, `q:Salir`, para salir, `d:Sup` para suprimir un mensaje, etc. Vemos un ejemplo de uso para hacernos una idea de las funciones básicas.

Ejecutamos el comando `mutt` y vemos en el terminal un programa casi todas las líneas vacías, salvo la primera y las dos últimas. Probablemente en el momento de abrir Mutt por primera vez no veamos nada interestante, pero aquí tienen un ejemplo de una lista de mensajes vista desde Mutt.

```
q:Salir  d:Sup.  u:Recuperar s:Guardar m:Nuevo r:Responder g:Grupo
 43 Oct 27 Teresa Gonzalez ( 0)  *>Re: [l-gulic] CILA LLENO
 44 Oct 27 Lucas Gonzalez  ( 0)   >cvs
 45 Oct 27 Miguel Ángel Vi ( 0) Bienvenido al calendario de la
 46 Oct 28 Teresa Gonzalez ( 0) Re: [l-gulic] Cambios en el CVS
 47 Oct 28 Pedro Gonzalez  ( 0)  *>
 48 Oct 28 Carlos de la Cr ( 0) La pu~etera introduccion :-)
 49 Oct 28 carlos de       ( 0) maldito texto sobre java
 50 Oct 28 frodo@fmat.ull. ( 0) MUY IMPORTANTE!!
 51 Oct 28 frodo@fmat.ull. ( 0) ahora te llega?
 52 Oct 28 frodo@fmat.ull. ( 0) <como no te llegue! ..grrr
 53 Oct 28 Administrador d ( 0) Re: instala esto


 ---Mutt: /var/spool/mail/miguev [Msgs:53 425K]---(threads/date)-------
```

Vamos a enviar un email a alguien que esté con nosotros en el aula, de esa forma cada uno enviamos un correo y recibimos otro. Para ello pulsamos la tecla `m` y veremos como en la última línea nos pregunta por el destinatario del mensaje (`To:`). Introducimos ahí la dirección de email a la que enviaremos el mensaje:

```
To: frodo@fmat.ull.es
```

Seguidamante Mutt nos preguntará por el asunto del mensaje (`Subject:`). Es importante poner un asunto al mensaje, para que el destinatario pueda tener una idea de qué es ese mensaje antes de abrirlo. En un tiempo en que el contagio de virus por correo eletrónico es preocupantemente frecuente, resulta muy molesto recibir un mensaje de email sin asunto.

```
Subject: <Feliz cumpleaños!
```

Una vez que Mutt ya sabe el destinatario del mensaje y el asunto, ejecuta el editor que tengamos definido en el fichero de configuración `~/.muttrc`. Editamos el mensaje que queramos y salimos del editor **guardando el mensaje**, importante esto último ya que si salimos del editor sin guardar el mensaje Mutt cancelará el envío. Una vez que salimos del editor Mutt está preparado para enviar el mensaje, pero nos ofrece la posibilidad de hacer aún varias cosas.

```
y:Mandar  q:Abortar  t:To  c:CC  s:Subj  a:Adjuntar archivo  d:Descrip
    From: Miguel Ángel Vilela <miguev@fmat.ull.es>
      To: frodo@fmat.ull.es
      Cc:
     Bcc:
 Subject: Hola pringao
Reply-To: Miguel Ángel Vilela <miguev@fmat.ull.es>
     Fcc:
     Mix: <no chain defined>
     PGP: En claro

-- Archivos adjuntos
- I     1 /tmp/mutt-euler-19795-2          [text/plain, 8bit, iso-8859-1


-- Mutt: Crear mensaje
```

Como podemos apreciar en el ejemplo, tenemos varias opciones con sus teclas asociadas en la primera línea. Para cambiar el destinatario pulsaríamos `t`, para enviar una copia a alguien pulsaríamos `c`, para editar el mensaje de nuevo pulsaríamos `e`, etc. Para enviar el mensaje pulsamos `y`. Entonces Mutt nos devolverá a la primera pantalla, pero mostrando en la primera línea información acerca del envío del mensaje. Debería aparecer:

```
Mensaje enviado.
```

El resto del manejo básico de Mutt es bastante intuitivo y no presenta dificultades. Tan sólo hay que acostumbrarse a trabajar con pulsaciones de teclas en lugar de manejar el ratón. Si en cualquier momento deseamos información más detallada acerca de las opciones disponibles, pulsamos `?`.

```
i:Salir  -:PágAnt  <Space>:PróxPág  ?:Ayuda
^B          M |urlview\n           call urlview to extract URLs out of
^D          delete-thread          suprimir todos los mensajes en este
^E          edit-type              editar el tipo de archivo adjunto
^F          forget-passphrase      borrar contraseña PGP de la memoria
<Tab>       next-new               saltar al próximo mensaje nuevo
<Return>    display-message        mostrar el mensaje
^K          extract-keys           extraer claves PGP públicas
^N          next-thread            saltar al próximo hilo
^P          previous-thread        saltar al hilo anterior
^R          read-thread            marcar el hilo actual como leído
^T          untag-pattern          quitar marca de los mensajes que coi
^U          undelete-thread        restaurar todos los mensajes del hil
<Esc><Tab>  previous-new           saltar al mensaje nuevo anterior
<Esc>C      decode-copy            crear copia decodificada (text/plain
<Esc>V      collapse-all           colapsar/expander todos los hilos
<Esc>b      M /~b                  search in message bodies
<Esc>c      change-folder-readonly abrir otro buzón en modo de sólo lec
<Esc>d      delete-subthread       suprimir todos los mensajes en este
<Esc>e      resend-message         usar el mensaje actual como base par
+                                  nuevo
Ayuda para index                                          -- (15%)
```

5.4.2. Fetchmail

Fetchmail es una aplicación que nos permite descargar de nuestro servidor de correo nuestros e-mails, puede como cliente y como servicio. La funcion del Fetchmail es conectarse al servidor de correo, bajarse los e-mails y luego pasarle los mensajes de correo electrónico a nuestro servidor de correo SMTP instalado en nuestra máquina (Debian por ejemplo instala por defecto el exim, aunque existen otros como qmail y postfix) y luego el servidor lo envía a los buzones de los usuarios.

Para la configuración del Fetchmail se utiliza el fichero `~/.fetchmailrc`. Vamos a utilizar un ejemplo sencillo.

```
poll pop.gulic.org proto POP3
   user "faraox@gulic.org" is there with password "coche" is faraox here
```

La primera línea llama, `poll`, al servidor `pop.gulic.org` e indica su protocolo con la opción `proto`, seguidamente del protocolo, `POP3`. En la siguiente línea se define el uruario y contraseña y las opciones del usuario. Con la opción `user is there` definimos el nombre de usuario, `faraox@gulic.org` y luego se define con `with password` nuestra contraseña `XXXX`, la opción `is here` nos indica el usuario al que debe ir el correo, `faraox`.

Una utilidad interesante si estamos manejando una máquina con varios usuarios es la utilización de el Fetchmail como demonio. Creamos el fichero de configuración y lo movemos a /etc/fetchmailrc. En la cabecera usamos la opción `set daemon` seguida de el número en segundo de el intervalo de tiempo con el que queremos que se ejecute.

dns Chequea las direcciones dns(por defecto).

no dns No chequea direcciones dns(recomendable, aumenta la velocidad).

timeout Especificamos el tiempo de inactividad del servidor, en segundos.

checkalias Hace una comparación de IP.

no checkalias No hace comparación de IP(por defecto).

folder Especifica la carpeta remota donde se consultará el correo.

mda Especifica nuestro programa de filtrado de correo (mailfilter,etc).

keep No borra los mensajes del servidor.

preconnect Comando para ser ejecutado antes de la conexión.

postconnect Comando para ser ejecutado depues de la conexión.

limit Especifica el tamaño máximo de los mensajes.

Otra utilidad interesante es `fetchmailconf`. Es un programa gráfico que nos permite configurar de forma intuitiva nuestro `.fetchmailrc`.

TEMA 6

Documentacion y ayuda

El primer contacto con un entorno nuevo como Linux siempre se suele hacer junto a alguien que te oriente (al que de ahora en adelante me referiré con el término *gurú de turno*). Cuando ese gurú vuelve a sus tareas cotidianas normalmente te quedas solo y te toca buscarte la vida. En este capítulo veremos diferentes formas con las que conseguir el conocimiento suficiente para poderse buscar la vida solo y hasta convertirte en un gurú también, pues eso es lo que él ha hecho para saber lo que sabe. Veremos que una gran parte ya lo tienes almacenado en tu disco duro tras instalar Linux, y te diremos cómo conseguirlo, mientras que el que te falte lo podrás encontrar en la red, y te diremos dónde buscarlo.

6.1. Pasos para encontrar ayuda

Estos pasos son para cuando no sabes como funciona un programa. Si realmente lo que pasa es que no entiendes nada, deberías pasar al siguiente punto, Recursos, donde se detallan guías y tutoriales. Estos pasos que aquí se detallan están en orden de dificultad creciente.

6.1.1. Información en línea de comandos

Cada comando en Linux siempre está documentado, ya sea de una forma u otra. Cuando conocemos un comando y no sabemos lo que hace nuestra primera opción puede ser llamar al comando con el argumento `--help` o `-h`, con lo que en la mayoría de los casos se nos mostrará un pequeño resumen de su uso y de la forma de llamarlo, con lo que nuestras dudas comenzarán a disiparse. Este es un estándar de facto bastante extendido.

6.1.2. Páginas del manual

Si aún así seguimos con dudas, o si queremos ver una descripción más pormenorizada de lo que hace cada uno de los parámetros de que dispone el comando, probaremos si el comando dispone de *página de manual* tecleando `man comando`. En caso afirmativo, nos aparecerá una descripción más detallada del comando, comenzando por un resumen, la descripción de la sintaxis, del funcionamiento, de las opciones y por último información de bugs, ficheros relacionados y referencias cruzadas. Desde los primeros UNIX las páginas de manual han sido el principal medio de documentación sobre comandos concretos y todas coinciden en los mismos contenidos mencionados anteriormente.

Algo que debemos tener en cuenta es que debido también a la herencia UNIX, el primer idioma en que se escriben todos estos documentos técnicos suele ser el inglés, por ser el idioma de mayor difusión en el mundo de la informática. Cuando el documento es un documento muy usado, entonces voluntarios lo traducen a su idioma nativo. El caso es que las páginas de manual de los comandos más usados seguro que están ya traducidos al español. Si nada en el sistema, ni siquiera aplicaciones, aparecen en español, entonces es posible que haya algún problema con las variables de entorno del idioma (para más información consultar `man setlocale`). Si hay aplicaciones que sí aparecen en español, entonces puede ser que estas páginas de manual no estén instaladas y tengas que hacerlo (depende de la distribución, en Debian el paquete se llama `manpages-es`).

Relacionados con el comando `man` existen dos comandos. El primero es que `apropos`, que busca la palabra que le decimos en los resúmenes de las páginas de manual instaladas y nos muestra las páginas que coinciden. El segundo es de menor utilidad, se llama `whatis`, y simplemente muestra la línea de descripción del comando, no la ayuda completa.

Con la llegada de los entornos gráficos, las páginas del manual se han modernizado también y se han integrado en los entornos más populares. En el caso del entorno de escritorio KDE, abriendo un Konqueror y tecleando en la barra de dirección `man:` obtenemos un listado de todas las páginas de manual organizadas en categorías, o bien seguido de una barra y el nombre del comando nos aparece directamente su página, por ejemplo, `man:/ls`. En caso de tener varios idiomas también aparecen allí. En el caso del escritorio GNOME 1.2, todo lo tenemos en la aplicación Gnome-help-browser (sistema integral de ayuda).

6.1.3. Páginas info

Las páginas del manual para escribirse utilizan un preprocesador llamado `troff`. Con la llegada de nuevos y más versatiles preprocesadores, las páginas del manual tuvieron la posibilidad de vincularse unas con otras, es decir, disponer de un índice del que se va a otras páginas y sobre estas, a su vez, puede irse a otras, etc: organizarlas como una obra de consulta más que como páginas aisladas. El primer fruto de esto fueron las páginas `info`, que son un sistema de ayuda navegable. Actualmente los preprocesadores como `sgmltools` permiten producir formatos de salida info, HTML, y otros, a partir de un único texto de ayuda escrito por el programador de la aplicación, y cualquier ahorro de tiempo es una ventaja.

El visualizador de páginas info, igual que man, ha evolucionado con los entornos gráficos. El visualizador con interfaz en modo texto se ejecuta tecleando con `info` o bien `info comando` si ya sabemos lo que deseamos ver. Una vez dentro, con tabulador vamos recorriendo cada uno de los enlaces. Cuando queremos entrar en uno pulsamos `Enter`. La estructura es jerárquica, en forma de árbol, y podemos movernos a los nodos de arriba (`u`), siguiente (`n`) y anterior (`p`).

En modo gráfico, en el escritorio KDE, al igual que las páginas del manual, simplemente tecleamos en la barra de direcciones de Konqueror `info:` para verlo todo, o bien seguido de una barra y el nombre de un tema concreto, como por ejemplo, **info:/libc**. Para GNOME 1.2, la aplicación Gnome-help-browser (Sistema integral de ayuda) nos permite visualizarlo también.

6.1.4. Documentación DVI/PS/PDF/HTML

El siguiente paso en documentación se dió con LaTeXy con la evolución del `sgmltools` hacia DocBook. Con ambos sistemas, escribir manuales y documentos se facilita enormemente, y a partir de un único documento se pueden conseguir versiones en diferentes formatos:

- **DVI (DeVice Independent file)**: Es el formato de salida del procesador LaTeX. Sólo se usa en documentos sin imágenes. Se visualiza, entre otros, con el programa `xdvi`.

- **PS (PostScript)**: Es un formato de impresión para impresoras de calidad profesional y resulta muy interesante pues se ve perfecto tanto en impresoras profesionales como en impresoras normales. La visualización en pantalla aparece tal cual va a salir por impresora. Se puede visualizar con `gs`, `gv`, `gnome-gv` (`ggv`, para GNOME) y `kghostview` (para KDE).

- **PDF (Portable Document Format)**: Es un formato reciente similar al PostScript, con la desventaja de que no se imprime directamente en las impresoras, sino que es necesario instalar un visor. Como visualizadores libres tenemos el `xpdf` y todos los anteriores visores de PostScript. También existe el *Acrobat Reader* (`acroread`) que no es libre pero sí gratuito.

- **HTML (HyperText Meta Language)**: El formato de archivos de la WWW resulta interesante para manuales, pues se puede publicar en Internet y al mismo tiempo lo puedes tener como paquete instalado en un directorio de documentación de tu disco duro para poder verlo sin conexión. Suele contener vínculos que te permiten navegar por dentro de todas las relaciones en el documento. Se puede ver con cualquier navegador de las docenas de ellos que existen para Linux, tanto en modo gráfico como en modo texto. Aparte, los sistemas de ayuda de Gnome y KDE, así como `devhelp` (un sistema de ayuda para programadores), aunque todos tengan una interfaz integrada con el escritorio, resulta que usan HTML, por lo que también se pueden ver en cualquier navegador.

- **Texto plano**: Este formato puede parecer obsoleto, pero sigue usándose pues no hace falta ningun visor para verlo. En especial, es interesante cuando no tienes entorno gráfico, o cuando no tienes navegadores con los que visualizar el resto de formatos, o simplemente por ahorrar espacio, pues al carecer de todos los códigos de formato ocupa notablemente menos. Se visualiza con un simple editor, o bien con `less` o `zless` si tiene sufijo `.gz` (compresión `gzip`).

6.1.5. Documentación del paquete

Normalmente usamos Linux después de haber instalado una de las distribuciones existentes. Sea cual sea la distribución, una cosa en la que se coincide es organizar las aplicaciones en paquetes individuales. Cuando alguien instala uno de esos paquetes, se instalan los ejecutables y resto de archivos necesarios para el funcionamiento de la aplicación, y también las páginas de manual y de info y también suele ser común tener un directorio con el resto de documentación del paquete.

Recordemos que el software en Linux es código abierto, eso significa que el programador distribuye libremente el código fuente de las aplicaciones, con el deseo de que sea útil para más gente. Y para que esto sea posible para el máximo número de personas, el programador siempre suele acompañar junto al código fuente de su programa instrucciones de instalación y de uso, así como manuales que no estén en formato man o info. Cuando la gente de la distribución crea el ejecutable a partir del código fuente, éste ya no es necesario, pero como los archivos de ayuda del programador tienen información útil pasan a colocarse en un directorio, que suele ser `/usr/share/doc/nombredelpaquete`. Cuando no conseguimos ayuda de otras formas, no debemos olvidarnos de mirar aquí.

En la distribución Debian existe un paquete llamado `dhelp` que busca documentos útiles para el usuario por todos los directorios de documentación de paquete y te los indexa y muestra de forma organizada para un fácil acceso a todos ellos.

6.1.6. Recursos en Internet de la distribución

Cada distribución de Linux construye los ejecutables a partir de las fuentes a su manera, y ordena los archivos de configuración y archivos de ayudas según su criterio. Posiblemente las ayudas estén ahí y no seamos capaces de encontrarlas. Simplemente pidiendo ayuda a usuarios más veteranos de nuestra misma distribución consigamos resolver nuestro problema o duda. Para esto, solo tenemos que buscar foros en Internet sobre nuestra distribución, o bien sobre Linux en general. Si lo que queremos realizar es algo común, muy probablemente otra persona lo haya preguntado y ya ha sido contestado, así que buscando previamente en los archivos de las listas nuestra pregunta nos ahorraremos preguntarlo y evitaremos hacer a la gente preguntas que ya han sido tratadas y contestadas. A veces, cuando los usuarios veteranos están cansados de responder siempre a las mismas preguntas, se crean los *FAQ - Frequently Asked Questions* (traducido por *PUF - Preguntas de Uso Frecuente*), que son colecciones de preguntas/respuestas muy típicas y útiles para cualquiera.

Por ejemplo, para la distribución Debian, el sitio donde ponerse en contacto con usuarios veteranos es vía lista de correo (una dirección de correo electrónico a la que se pueden suscribir muchas personas) `debian-user-spanish@lists.debian.org` o bien en el siguiente foro: `http://www.esdebian.org/`. Para obtener FAQ's y otros documentos en: `http://www.debian.org/doc/index.es.html`

6.1.7. Recursos en Internet del programa

En caso que nuestras pretensiones de conocer el programa sean mayores, o que las anteriores búsquedas hayan sido infructuosas, siempre nos queda la opción de buscar la página web de la aplicación. Como las aplicaciones libres viven de su difusión, el programador siempre intenta disponer de una página web donde se anuncie su programa, las nuevas versiones, su dirección de correo donde comunicar bugs, enlaces con ayudas y tutoriales, listas de correo donde ser notificado sobre las actualizaciones o donde los usuarios de su programa pueden discutir, ayudarse unos a los otros o proponer mejoras al programa. Mirando en el directorio de documentación del programa o tecleando el nombre de la aplicación en un buscador nos puede llevar a esta página.

6.1.8. Código fuente

Una lectura del código fuente puede a veces sacarnos de ciertos problemas. A lo mejor no está a tu alcance, pero cuando los pasos anteriores son infructuosos, éste es el último paso que te queda, y el que te convertirá en un gurú. Por ejemplo, si nuestra aplicación no funciona y acaba con un mensaje de error extraño, y ni en los buscadores ni otros sitios obtenemos respuesta, siempre podemos buscar la cadena del mensaje en el código fuente y una vez lo encontremos, mirando el entorno del mensaje, aunque no conozcamos el lenguaje de programación, a lo mejor podemos intuir que debe estar pasando para que nuestro programa no funcione. También, si el programa tiene aún partes en desarrollo o partes que no han sido probadas, el programador puede indicarlo poniendo un texto con una advertencia en medio del código. Si vemos que el programa falla en algo inacabado, ya vemos que no es problema nuestro. Este es el poder de la fuente abierta. *¡Usa la fuente Luke!* Esto es lo que distingue a los verdaderos *Jedi* de Linux.

6.2. Recursos

6.2.1. Linux Documentation Proyect

Este proyecto centraliza mucha información. Por su grado de elaboración, se pueden subdividir en:

- Guides (Manuales/Tutoriales/Cursos): Explicación exhaustiva sobre un conjunto de temas.
- Howtos (Cómos): Explicación exhaustiva sobre un tema concreto.
- Faqs (Pufs): Preguntas y respuestas sueltas sobre un determinado tema.

Si no están instalados ya en nuestra distribución, los sitios donde podemos conseguirlos son los siguientes:

- Original, en inglés: // `http://www.tldp.org/docs.html`
- En español (proyecto Lucas): // http://es.tldp.org/

No todos los documentos están traducidos, y la fuente más oficial es en inglés, así que si no encuentras lo que buscas en español, intenta buscarlo en inglés.

En la distribución Debian existen los siguientes paquetes con información en castellano que puedes instalar:

`ldp-es-lipp` Linux Instalación y Primeros Pasos, 239 páginas, se encuentra en `/usr/share/doc/ldp-es/lipp/lipp-1.1.ps.gz`

`ldp-es-garl` Guía de Administración de Redes con Linux, 352 páginas, se encuentra en `/usr/share/doc/ldp-es/garl/nag.ps.gz`

`ldp-es-gulp` Linux Programming Guide, 151 páginas, se encuentra en `/usr/share/doc/ldp-es/gulp/lpg.ps.gz`

`ldp-es-glpu` Guia de Linux para el Usuario, 169 páginas (bash, editores, xwindows, etc), se encuentra en `/usr/share/doc/ldp-es/glup/guide.ps.gz`

`linux-tutorial-es` Tutorial de Linux, se encuentra en `/usr/share/doc/linux-tutorial-es/index.html`

`lucas-novato` De novato a novato, 81 páginas (instalaciones debian 1.3 y 2.0, comandos básicos, trucos...), se encuentra en `/usr/share/doc/lucas/novato/novato-a-novato.ps.gz`

`doc-linux-es` HOWTOS traducidos al castellano, 117 COMOS diferentes para multitud de temas, se encuentran en `/usr/share/doc/HOWTO/es/HOWTO/`

`doc-es-misc` FAQ de `es.comp.os.linux`, 142 páginas, se encuentra en `/usr/share/doc/doc-es-misc/linux-faq.es.ps.gz`

6.2.2. Buscadores: Google

Cada proyecto dispone de su web donde ver información de actualizaciones, ultimas versiones, etc. Para encontrarla puedes usar cualquiera de los buscadores de Internet, por ejemplo, `www.google.com`. En navegadores como Mozilla, las búsquedas en Internet están integradas y facilitadas con un botón que dice Buscar en el propio navegador. En Konqueror, para buscar en Google solo tienes que escribir en la barra de direcciones `google:` y las palabras que quieras buscar separadas por espacios.

6.2.3. Listas de correo (*mailing list*)

Una lista de correo es una dirección de correo ficticia a la que te suscribes y que si envías un mensaje a esta dirección es recibido por todas las personas que se hayan suscrito. Las listas de correo se crean con una temática en mente, y hay multitudes para casi cualquier tema, y Linux no va a ser menos en este aspecto. Como ejemplo, las siguientes (la lista podría ser interminable):

- Conjunto de listas sobre el proyecto Debian, donde los usuarios y los desarrolladores se organizan:
  `http://lists.debian.org/`
- Listas para el proyecto de escritorio Gnome:
  `http://mail.gnome.org/`
- Listas para el proyecto de escritorio KDE:
  `http://lists.kde.org/`

6.2.4. Grupos de news (red USENET)

Los grupos de news o USENET son lo mismo que las listas de correo pero usando un protocolo específico llamado *NNTP (News Network Transfer Protocol)* Para verlas necesitas usar un cliente especial, por ejemplo: Mozilla News, luego `knews` o `knode` para KDE, `sylpheed` o `pan` para GNOME, `lynx` y `trn` para consola, y muchos más. Aquí tienes más información:

- Buscador dentro de grupos de noticias:
  `http://groups.google.com/`
- Los FAQS de muchos grupos de news españoles:
  `http://www.faqs.es.org/`
- Página web de la jerarquia de grupos de news `es.comp.os.linux.*`:
  `http://www.escomposlinux.org/`

6.2.5. Páginas de bugs

Cuando llevas un tiempo usando un programa y te das cuenta que haciendo una secuencia de acciones consigues colgarlo, y cada vez que repites esta secuencia lo cuelgas, resulta que has conseguido un bug del programa (fallo, errata). Es raro encontrar estos fallos de forma tan determinista ya que normalmente los fallos suelen ser aleatorios lo que dificulta su localización y obliga a usar herramientas de programador para descubrirlo (depuradores, core-dumps, trazas, etc), pero si lo encuentras le habrás hecho un gran favor al programador. Ahora solo tienes que decirle lo que has descubierto. Busca en la página de su programa el listado de bugs, comprueba que el tuyo ya no esté (lo siento ;-), y si no está, comunícalo. Para ello suele existir un formulario web donde escribes tu dirección de correo, el bug y se le asigna un número. Luego, por ese número, podrás volver a la web y encontrar los comentarios del programador o de otros usuarios con respecto a ese bug. En caso de que no disponga de este sistema, simplemente suscribiéndote a su lista de correo o enviándole un dirección de correo te servirá.

Ejemplos de sitios solo para notificar y hacer seguimiento de bugs son los siguientes:

- Para la distribución Debian:
  `http://www.debian.org/Bugs/`
- Para el navegador Mozilla:
  `http://bugzilla.mozilla.org/`
- Para el entorno de escritorio GNOME:
  `http://bugzilla.gnome.org/`
- Para el entorno de escritorio KDE:
  `http://bugs.kde.org/`

MÓDULO II

Instalación de GNU/Linux

TEMA 7

Instalación de GNU/Linux

7.1. Introducción

En este capítulo repasaremos de forma genérica, para que nos sirva de guía, una instalación de cualquier distribución de Linux. Tenemos que decir que lo que nos planteamos conseguir no es tarea sencilla porque todas las distribuciones tienen sus pequeñas diferencias aunque generalmente todas pasan en sus procesos de instalación por unas fases que son comunes que aquí vamos a detallar. El orden en el que aparecen las diferentes etapas comentadas en este capítulo no tiene por que ser de forma estricta el orden a seguir por todas las distribuciones, pero es uno de los más usuales en los procesos de instalación.

¡Atención! Antes de comenzar con la instalación te recomendamos que leas la sección referente a las particiones que se puede encontrar en este mismo capítulo.

7.2. Arrancando

Normalmente cuando nos disponemos a instalar una distribución Linux lo hacemos desde CD-ROM. Si nuestra placa base nos permite arrancar desde CD-ROM no tendremos problema ya que simplemente colocamos el CD de nuestra distribución preferida en el CD-ROM, le indicamos a la BIOS que lo primero que intente arrancar sea el CD-ROM y listo, tendremos nuestro sistema de instalación de nuestra distribución esperando a que le demos la orden de continuar.

Figura 7.1: Cargador de arranque de Mandrake Linux

Si nuestra placa no soporta el arranque desde el CD siempre se pueden hacer los disketes para arrancar desde la disketera. Todos los PC's que disponen de disketera pueden arrancar desde ella. Para arrancar el sistema de instalación desde discos de 3,5"tendremos que buscar en el contenido del CD de instalación de la distribución las imágenes de los discos de arranque. Normalmente están en un directorio que se llama "images", aunque esto no es una norma fija. Cuando los encontremos tendremos que copiar las imágenes a los disketes. Esto se hace con una herramienta que normalmente traen también todos los CDs de las distribuciones que se llama rawrite. Seguiremos las indicaciones que nos dé el rawrite para hacer los disketes de instalación y guardaremos con celo el trabajo realizado hasta ahora.

Una vez que tenemos esos discos en la mano tendremos que decirle a nuestra BIOS que intente arrancar en primer desde la disketera, reiniciamos y colocamos el diskete de instalación en la disketera. Tras unos momentos de incertidumbre tendremos finalmente el sistema de instalación de nuestra distribución listo para continuar.

7.3. Idioma y teclado

Normalmente después de la bienvenida tendremos que elegir el idioma para la instalación y la distribución del teclado que usaremos.

El idioma es a tu gusto, elige aquél con el que más cómodo estés y la distribución de teclado se recomienda que uses la distribución de tu teclado, ya que si no lo haces podrás encontrarte que pulsando la tecla ñ aparezca una p o algún otro carácter. Los teclados españoles son del tipo qwerty-ES.

Figura 7.2: Selección del idioma

7.4. Particiones

Hemos llegado a un punto delicado de la instalación, normalmente es peligroso porque podemos dañar la información almacenada en nuestros discos duros.

Lo primero que nos planteamos es si vamos a tener un ordenador con un único sistema operativo, en nuestro caso Linux o si serán varios sistemas operativos, cuyo caso más generalizado es Linux y Windows conviviendo en el mismo ordenador.

7.4.1. El caso más general: convivamos con Windows

Comenzamos por el caso más general, es decir, los dos sistemas operativos juntos. Suponemos que ya está instalado Windows y que sólo tenemos un disco duro. En ese caso tendremos que hacer sitio para la instalación de Linux en el disco duro que disponemos. Este paso es delicado, pero si se siguen las instrucciones aquí detalladas no tendremos problemas. Los pasos a seguir para no perder la información de la partición de nuestro Windows son los siguientes:

1. Desfragmentar la partición de Windows.
2. Decidir cuánto espacio vamos a dejar para la instalación de Linux. Esta decisión al final corre a cuenta del usuario. Se recomienda que para una instalación completa de las distribuciones, se reserve alrededor de 2 Gb de espacio en disco.

3. Reducir el tamaño de la partición de Windows. Para llevar a cabo esta tarea se tienen varias posibilidades. Podemos encontrar software[1] para Windows, que facilita la labor permitiendo redimensionar el tamaño de la partición de manera gráfica. Ésta es la opción que recomendamos para los usuarios con poca experiencia en este tipo de prácticas. Para usuarios con algo más de conocimientos tenemos utilidades menos amigables pero más fáciles de conseguir como son el fips sobre msdos, el fdisk, el parted de Linux y el diskdruid. Tened cuidado con estas dos herramientas porque, repetimos, pueden dañar el contenido de vuestros discos duros.

4. Establecer las particiones de Linux. Se necesitarán dos particiones en lugar de una como se piensa inicialmente. A continuación se detallará este paso, común a la convivencia y a la vida en soledad.

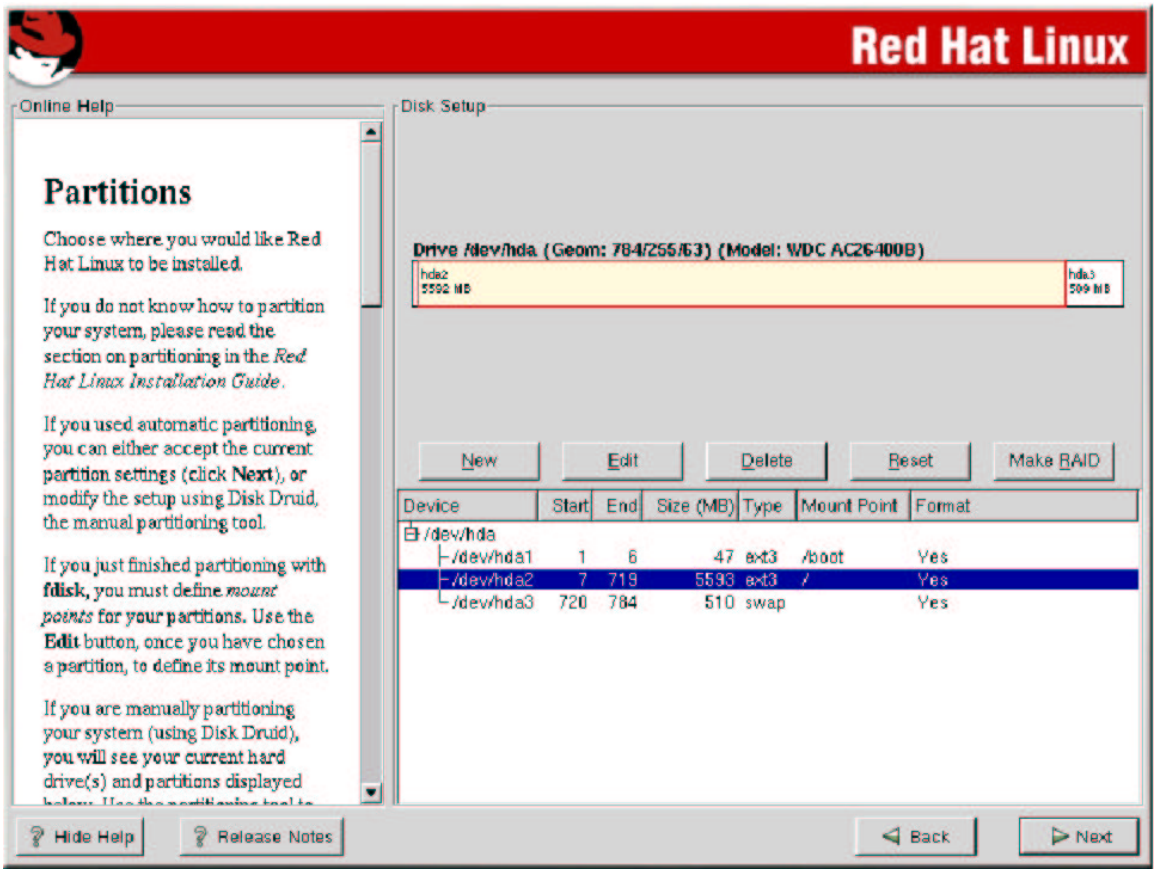


Figura 7.3: Preparación de las particiones

7.4.2. La vida en solitario

Trataremos el segundo supuesto: en nuestro disco duro sólo habrá un sistema operativo, Linux. Tendremos que hacer dos particiones, en este punto hay mucho que explicar. Empezaremos por comentar los tipos de particiones que normalmente se pueden encontrar en un PC doméstico. Hemos de tener claro lo que son particiones primarias, extendidas y unidades lógicas.

7.4.3. Tipos de particiones

Particiones primarias: es el tipo básico de partición. Podremos disponer como máximo de cuatro particiones primarias. Si hacemos uso de todas las particiones primarias ya no podremos crear más particiones de ningún otro tipo[2].

Particiones extendidas: en caso de querer más de cuatro particiones deberemos usar unidades lógicas. Para crear una unidad lógica tendremos que eliminar al menos una partición primaria para crear una partición extendida. Desde este momento podremos crear tantas unidades lógicas como deseemos dentro de esta partición extendida.

[1] Uno de los paquetes de software más usados en estos menesteres es el *PartitionMagic* de *PowerQuest*
[2] Como observación diremos que en Windows tendremos problemas a la hora de trabajar con más de una partición primaria en FAT16 o FAT32

Unidades lógicas: este tipo de particiones son las que se crean a partir de la partición extendida, nunca podremos crear unidades lógicas dentro de una partición primaria.

Una vez aclarados los tipos de particiones que podemos encontrarnos vamos a determinar la manera que tiene Linux de asignarles un nombre de dispositivo. En Linux todos los dispositivos son ficheros, desde la pantalla (`/dev/stdout`) hasta el teclado (`/dev/stdin`) pasando por los discos duros, CD-ROM, etc....

7.4.4. Nomenclatura de particiones

En primer lugar hemos de decir que los PC's domésticos suelen tener dos controladoras de disco duro (IDE1 e IDE2) y en cada una de estas controladoras podremos conectar dos dispositivos físicos (dispositivo maestro y dispositivo esclavo). A estas controladoras se puede conectar todo lo que tenga una interfaz hardware IDE, esto es, discos duros IDE, CD-ROM's IDE, DVD's IDE, unidades ZIP y DAT, y una larga lista de dispositivos.

La manera de nombrar los ficheros correspondientes a cada uno de estos elementos de hardware es la siguiente:

$$\text{Controladora IDE1} \begin{cases} \text{Maestro} \Rightarrow \texttt{/dev/hda} \\ \text{Esclavo} \Rightarrow \texttt{/dev/hdb} \end{cases}$$

$$\text{Controladora IDE2} \begin{cases} \text{Maestro} \Rightarrow \texttt{/dev/hdc} \\ \text{Esclavo} \Rightarrow \texttt{/dev/hdd} \end{cases}$$

Por si quedaran más dudas, como casi todos estamos acostumbrados a trabajar en Windows, si cada disco forma una única partición, podemos decir que `/dev/hda` es el equivalente a **C:**, `/dev/hdb` es el equivalente a **D:** y así sucesivamente...

Sigamos con la nomenclatura de las particiones utilizada por Linux. Las particiones primarias y las extendidas se nombran añadiendo al nombre del dispositivo, ya detallado, un número que indica el orden lógico de la partición. La primera partición de uno de los discos será `/dev/hdx1`, la segunda `/dev/hdx2` donde la **x** es lo que va a determinar a que disco nos estamos refiriendo según la tabla anterior.

Las unidades lógicas también se nombran utilizando el orden en el que se definen pero teniendo en cuenta que los cuatro primeros números están reservados para las particiones primarias, por lo tanto, las unidades lógicas empezarán a numerarse a partir del cinco.

Imaginemos por un momento que tenemos un disco duro, con una partición extendida, tres primarias, y en la partición extendida tenemos tres unidades lógicas. Haremos referencia a cada una de las particiones de esta manera (Suponemos que el disco está conectado como maestro a la primera controladora IDE).

partición extendida	⇒	`/dev/hda1`
unidad lógica 1	⇒	`/dev/hda5`
unidad lógica 2	⇒	`/dev/hda6`
unidad lógica 3	⇒	`/dev/hda7`
partición primaria 1	⇒	`/dev/hda2`
partición primaria 2	⇒	`/dev/hda3`
partición primaria 3	⇒	`/dev/hda4`

Bien, sabemos que esto es bastante denso al principio, cuando no se tiene práctica, pero con el uso nos daremos cuenta que tiene mucha lógica y que aprendiéndonos un par de reglas todo va sobre ruedas.

7.4.5. Esas dos particiones

En este punto ya es hora de introducir el significado de las dos particiones necesarias en un sistema Linux. La primera de ellas es la partición de swap. Esta partición lo utiliza el sistema operativo como memoria de intercambio. En el momento en el que se llena la memoria principal de nuestro ordenador, el sistema utiliza la memoria de intercambio para continuar ejecutando los programas sin problemas. Esta partición es de uso exclusivo del sistema, lo que significa que no la podremos utilizar para guardar nuestro datos. La buena noticia es que esta partición es muy pequeña, normalmente es suficiente con dejar para la partición de swap 256 Mb de disco, no obstante puede ser todo lo grande que se quiera, siempre que haya sitio para el resto del sistema. El tipo de sistema de ficheros que tendremos que asignar a esta partición no tiene duda, Linux Swap.

La segunda partición es donde vamos a colocar todos los ficheros del nuevo sistema operativo, en ella encontraremos desde nuestros directorios personales hasta los programas de usuario, pasando por el núcleo del sistema operativo, por lo tanto, esta partición tendremos que dejarle bastante sitio. Como ya hemos recomendado anteriormente 2Gb serán suficientes para una instalación normal de Linux. Ahora entremos en el fantástico mundo de los sistemas de ficheros, que como llevamos haciendo hasta el momento también daremos un breve repaso.

7.4.6. Tipos de sistemas de ficheros

Los sistemas de ficheros se pueden clasificar de varias maneras, nosotros lo haremos atendiendo a la capacidad de los sistemas de ficheros transaccionales (journaling). El journaling es una técnica que implementan algunos sistemas de ficheros muy parecida a la utilizada por las transacciones en las bases de datos. La técnica consiste en que todas las operaciones que se han de realizar sobre el sistema de ficheros se guardan en disco antes de realizarse, y luego se van llevando a cabo y se van eliminando de la lista a medida que se concluye su ejecución. La ventaja que nos dan los sistemas con journaling es que si nuestro sistema se cae (por ejemplo se va la luz) todas las tareas que el disco tendría que haber hecho para mantener la estructura lógica del sistema de ficheros está guardada y cuando restablecemos el sistema tenemos el disco en perfecto estado, eliminando de esta manera los famosos verificadores de la estructura del disco, el más conocido es scandisk.

Comencemos con los sistemas de ficheros que no implementan journaling, éstos son:

FAT16: sistema de fichero nativo de Windows95 (sin política de seguridad robusta).

FAT32: sistema de ficheros nativo de Windows98-WindowsMe (sin política de seguridad robusta).

ext2: sistema de ficheros nativo de Linux 2.x y 4.x (con una buena política de seguridad).

Sistemas de ficheros que implementan journaling:

ext3: Este sistema de ficheros es una evolución del ext2 que implementa journaling además de la política de seguridad basada en permisos del ext2.

ReiserFs: Sistema de ficheros con journaling que obtiene muy buenos resultados trabajando con ficheros de gran tamaño, también cuenta con una muy buena política de seguridad.

NTFS: Sistema de ficheros nativo de WindowsNT, Windows2000 y WindowsXP, tiene un sistema de seguridad basado en ficheros y en listas de control de acceso. Desgraciadamente este tipo de sistema de ficheros no es abierto, está sometido a continuos cambios por parte de su dueño, Microsoft, y por lo tanto no se tiene un buen soporte en Linux, de tal modo que sólo está soportado de manera segura la lectura de este tipo de sistema de fichero.

XFS: sistema de ficheros que implementan las máquinas Silicon Graphics.

JFS: sistema de ficheros que implementan los main frame IBM.

Con esta brevísima descripción de los tipos de sistema de ficheros podrás elegir tú mismo cuál quieres poner. Nuestra recomendación es la de poner ext2 ya que este sistema de ficheros está en todas las distribuciones, pero si tu distribución te da la posibilidad de utilizar un sistema de ficheros con journaling, escoge ext3 ya que en las comparativas obtiene mejores resultados en sistemas de propósito general que los otros tipos de sistema de fichero con journaling, y por supuesto, la elección entre journaling o no journaling es la primera. No solo por tener que estar esperando a que termine el proceso de fsck, sino también por la robustez y eficiencia de este tipo de sistemas de ficheros.

Vamos a dar por último la respuesta a la pregunta que muchos se estarán haciendo: ¿Podré utilizar la partición de Windows desde Linux y la de Linux desde Windows?. Hay varias respuestas bastante claras y tajantes. Sobre si se podrán utilizar las particiones de Linux desde Windows, la respuesta es no de escritura (existen problemas de licencias en el desarrollo) pero sí de lectura (mediante una utilidad llamada *explore2fs*). La otra respuesta depende del tipo de sistema de ficheros que tengamos en Windows. Para los sistemas de ficheros del tipo FAT16 o FAT32 se podrán utilizar sin problemas. Para los sistemas de ficheros con NTFS, lo más recomendable es utilizarlos en modo lectura, ya que el soporte de escritura no está completo debido a que Microsoft no revela los cambios que introduce en el esquema del sistema de ficheros, por tanto, es muy peligroso hacer operaciones de escritura ahí desde Linux. Esta limitación en ambos sentidos es producto de la ocultación de información sobre la que se basan los sistemas cerrados y propietarios y que impide que las aplicaciones abiertas y libres permitan realizarse.

7.5. Instalación del kernel y los módulos

En este momento ya le hemos dicho al sistema de instalación dónde va a tener que colocar todos los ficheros. Ahora tendremos que comenzar a tomar decisiones para indicar qué es lo que queremos instalar y qué hardware tenemos en nuestra máquina.

Lo primero que tendremos que hacer es elegir qué kernel poner. Normalmente podemos elegir entre unas pocas opciones. Llegados a este punto si no sabes cuál elegir, escoge la recomendada por la distribución. Ésta es siempre una buena elección en caso de duda. En cambio si sabes el kernel que quieres poner entre todas las opciones, selecciónalo y continuemos con la instalación.

Una vez decidido qué kernel vamos a usar tendremos que, en algunas instalaciones, decir qué hardware tenemos instalado y qué drivers queremos que use. Este paso se hace cada vez de manera más automática y no está recomendado para la gente que instala por primera vez Linux. Sin embargo, no pasa nada por mirar tocar e intentar cargar módulos para ver qué es lo que pasa. Quienes hayan instalado un sistema Linux previamente y conozcan el tipo de hardware que poseen y los módulos que hacen falta para que éste funcione correctamente es éste el momento de seleccionarlos y configurarlos para su correcta instalación.

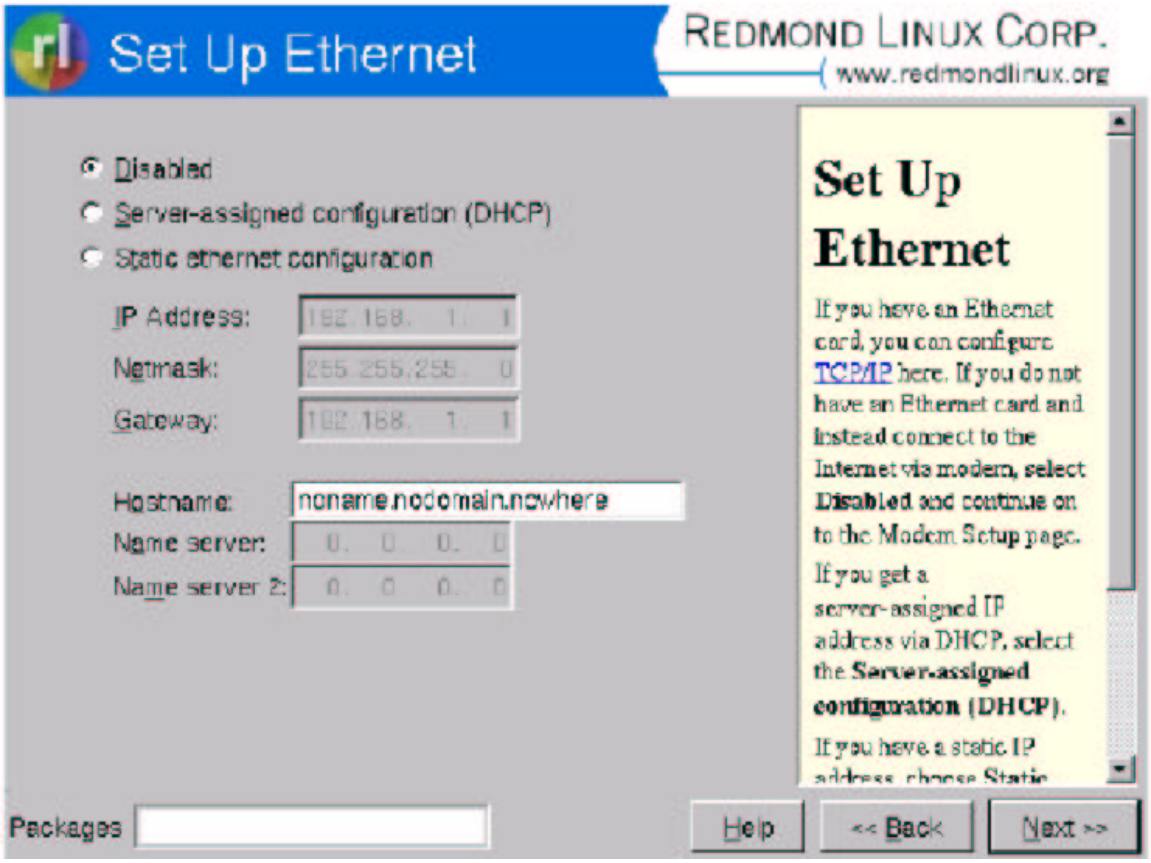


Figura 7.4: Configuración de la red

Si hemos instalado módulos de hardware que necesitan algún tipo de configuración, por ejemplo tarjetas de red, tras haber seleccionado el módulo y cargado con éxito en el kernel, tendremos que proporcionar la información necesaria al sistema para que se pueda configurar dicho hardware correctamente, que generalmente será algún parámetro adicional. En el caso de la tarjeta de red tendremos que decirle si coge la configuración por dhcp o si la especificamos nosotros a mano.

7.6. Estableciendo el arranque

Este es otro de los pasos delicados de la instalación, menos que el de particionar el disco duro pero también tendremos que tener cuidado con lo que hacemos. Vamos a seguir la estructura del apartado de las particiones, es decir, vamos a separar los sistemas donde conviven dos sistemas operativos y los sistemas donde sólo tenemos Linux.

Para aquellas máquinas con arranque dual Windows y Linux lo normal es que no se disponga de un sistema gestor del arranque antes de instalar Linux, por lo tanto tendremos que instalar el gestor de arranque de nuestra distribución (LILO o GRUB) en el master boot record MBR. Lo que lograremos de con esto es que nuestro cargador se ejecute antes de cualquiera de los sistemas operativos y nos permita decidir cual de los dos queremos arrancar. No basta con instalar el gestor de arranque en el MBR, sino que, tendremos que indicar que particiones queremos que arranque. Normalmente esto se hace de manera automática y simplemente tendremos que seleccionar las particiones a arrancar, pero si no sucede de manera automática haremos uso de nuestro conocimiento en nomenclatura de particiones para indicar que arranque por ejemplo la primera partición de nuestro IDE1 maestro (`/dev/hda1`).

En las máquinas donde no convivan los dos sistemas operativos tendremos que instalar LILO en el master boot record pero no tendremos que tener tanto cuidado a la hora de indicar que es lo que tiene que arrancar puesto que el sistema de instalación configurará todo él solo.

7.7. Instalar el sistema base

En este momento las distribuciones empiezan a diverger, en el sentido de que no en todas este paso se realiza de la misma manera. En algunas primero se instala el sistema base y luego se selecciona qué software extra se quiere instalar. En cambio hay otras distribuciones en las que el sistema base y el software adicional a instalar se seleccionan en un solo paso.

Si nuestra distribución decide instalar el sistema base simplemente haremos caso, en este paso normalmente no hay muchas cosas que decidir. Simplemente se instalan los programas básicos para que el sistema operativo pueda arrancar y poco más.

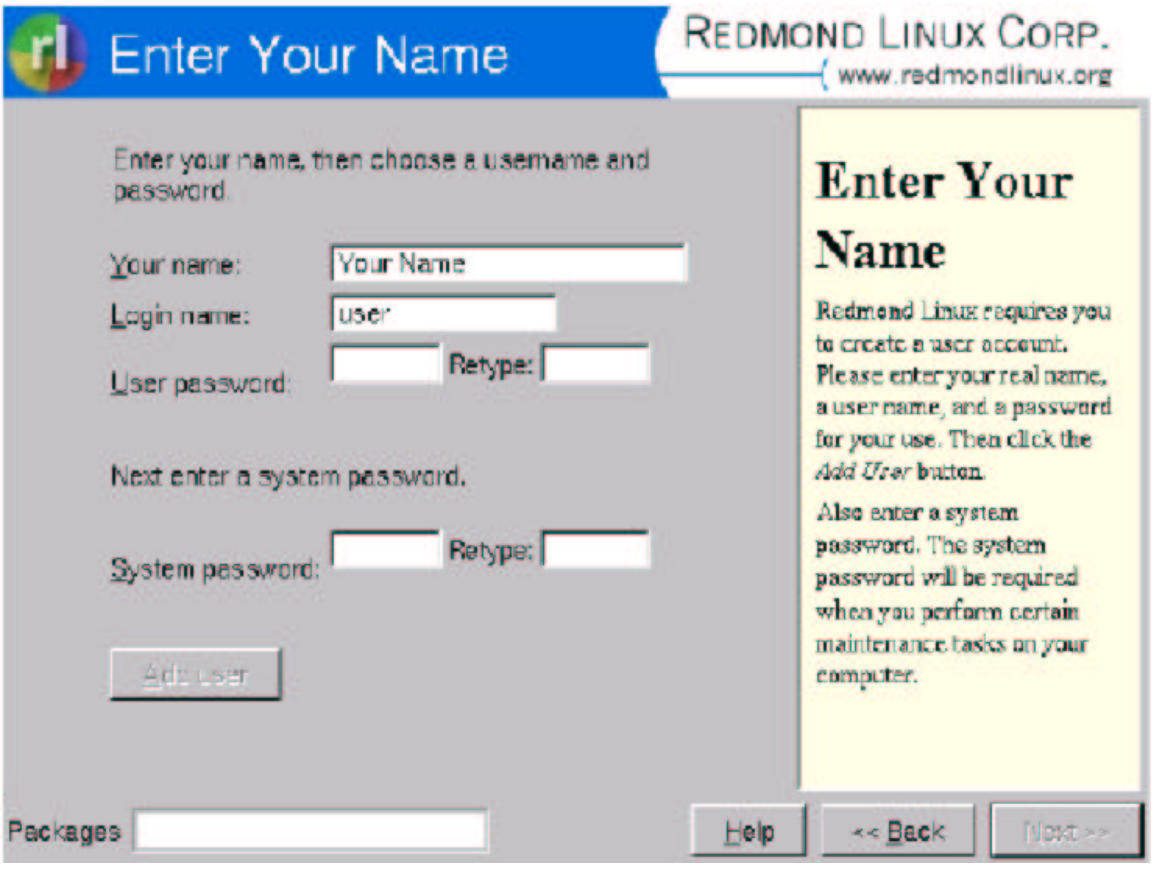


Figura 7.5: Creación de usuarios durante la instalación

7.8. Instalar el software

Hoy en día las distribuciones traen un sistema gráfico de selección de paquetes (software a instalar). Normalmente se encuentran organizados en categorías o ramas según el fin para el cual se usará ese software. A continuación vamos a describir

los más comunes:

Paquetes base Tal y como su propio nombre indica se trata del sistema base nombrado en el párrafo anterior. Entre otros incluye utilidades como **bash** (un tipo de consola, shell), **adduser** (utilidad para añadir y eliminar usuarios), **gzip** (compresor de ficheros) ...

Paquetes devel Esta sección de software está dedicada al desarrollo del mismo, es decir que en esta rama es donde podremos encontrar herramientas de programación tales como compiladores, librerías, depuradores, preprocesadores, etc. Algunos de ellos son: **g++** (compilador de C++), **g77** (compilador de Fortran77), **ddd** (depurador de código), **cvs** (sistema de control de revisiones) ...

Paquetes doc Aquí podemos encontrar todo tipo de documentación relacionada con los programas instalados o el uso de librerías. Sin duda lo más destacable de este apartado son las llamadas páginas man **(manpages)**.

Paquetes math Software dedicado a realizar tareas matemáticas. Podremos encontrar programas como **GNUplot** (generador de gráficas), **octave** (programa estrella de esta sección, sirve para realizar todo tipo de cálculos numéricos, simulaciones, representaciones etc; un clónico de MathLab), **gnumeric** (hoja de cálculo para GNOME) ...

Paquetes X11 Esta rama de paquetes proporciona la mayoría de los componentes del sistema X-Window desarrollados por *XFree86 Project*, junto a otros accesorios. Podremos encontrar el sistema base de las X, KDE, GNOME, administradores de sesiones como xdm, gdm, kdm, distintos tipos de fuentes, servidor de fuentes (xfs), etc.

Los paquetes nombrados previamente son solamente algunos de los básicos. Todas las distribuciones traen una multitud de programas adicionales los cuales puedes instalar a tu gusto y dependiendo de tus necesidades. Casi todas las aplicaciones se pueden encontrar en los CD's de tu distribución favorita, pero algunas de ellas pueden no estar en esos CD's.

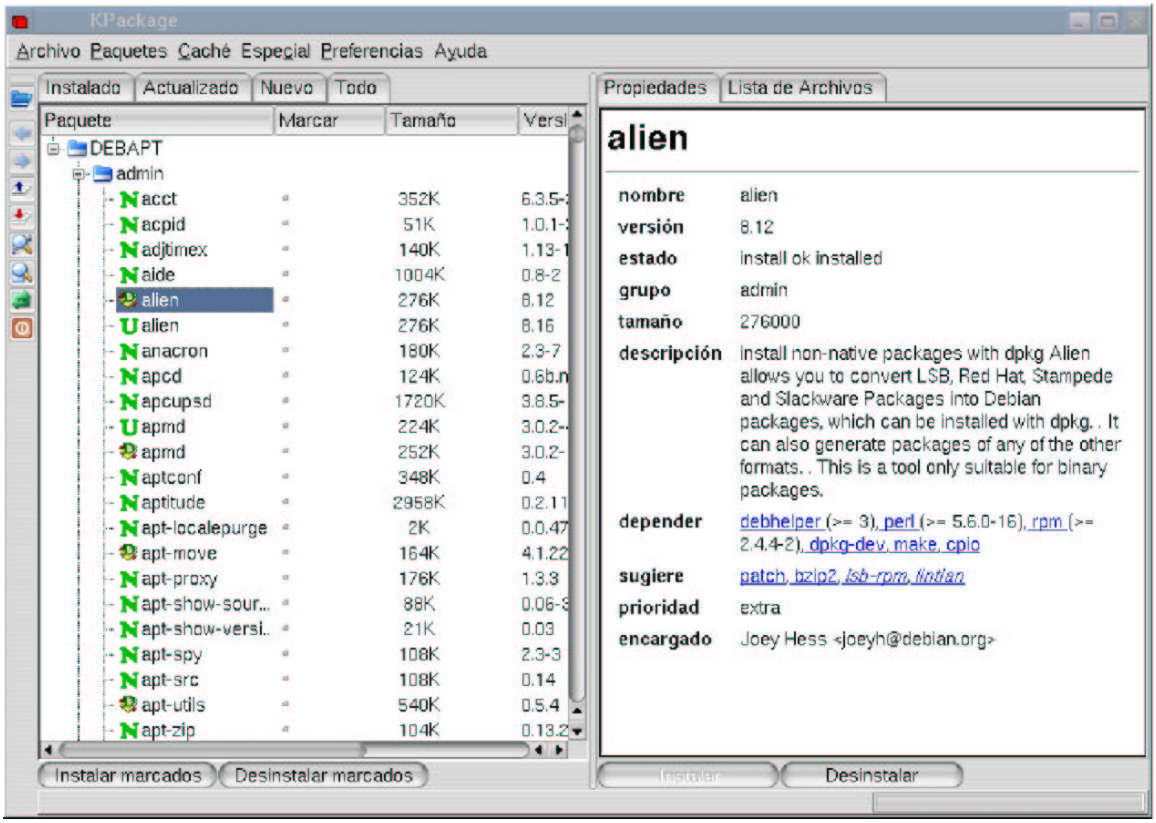


Figura 7.6: KPackage. Gestor de paquetes RPM y DEB para KDE

Para instalar esos programas bastará con descargar los paquetes de esa aplicación e instalarlos. Hay que anotar que existen paquetes binarios (compilados) para las distintas distribuciones. Los más famosos son los paquetes .RPM (los cuales usan Red Hat y Mandrake) y los paquetes DEB (usados en las distribuciones Debian). En la página *www.rpmfind.net* podremos encontrar (casi) todos los paquetes rpm existentes, con lo cual nos bastará con buscar el programa en cuestión, descargarlo e instalarlo ejecutando el comando `rpm -i nombre-del-paquete.rpm`. Los usuarios de Debian lo tienen bastante más fácil en

este aspecto gracias a las herramientas APT. En el caso de que un paquete no se pueda instalar mediante esas herramientas podemos descargar nuestro paquete .deb e instalarlo ejecutando el comando `dpkg -i nombre-del-paquete.deb`.

Existen más tipos de paquetes dependiendo de la multitud de distribuciones, pero algunas veces no podremos encontrar el software deseado en forma de paquete binario. En ese caso tendremos que descargar el código fuente del programa, compilarlo e instalarlo. Normalmente el código se encuentra en forma de paquetes .TGZ (paquetes comprimidos), con lo cual habrá que descomprimirlos antes de nada. La secuencia de pasos para realizar lo anterior es la siguiente:

```
$ tar zxvf nombre-del-paquete.tgz
$ cd directorio-del-programa
$ ./configure
$ make
$ make install
```

Esta secuencia es la más usual, pero (como siempre) no funciona en todos los casos, para los cuales tendremos que informarnos en la documentación del programa.

Por último cabe destacar que la mayoría de software hace uso de otros programas o librerías sin las cuales no se podrá instalar o compilar la aplicación en cuestión. La solución pasa por instalar esas dependencias previamente y a continuación el programa.

7.9. Configuración del servidor X

Dependiendo de la distribución que estamos instalando nos encontraremos con distintas utilidades para configurar el sistema X-Window. Todas ellas son muy similares entre sí pero con ligeras diferencias. Para no concretar el proceso de configuración según las distintas distribuciones usaremos la herramienta `xf86config` la cual está presente en todas ellas.

El programa `xf86config` es un programa de consola, en modo texto, que modifica los ficheros de configuración del servidor X, para lo cual necesitaremos los privilegios de superusuario. Para invocar el programa abrimos una consola, accedemos como root (usando el comando su si procede) y tecleamos `xf86config`. Nos encontraremos con una pantalla de descripción de la aplicación. Cabe destacar que la ejecución del programa se podrá abortar en cualquier momento presionando la combinación de teclas Ctrl-C. Así que no debemos temer a equivocarnos ya que al interrumpir esta herramienta no se afectan los ficheros de configuración existentes.

Para proceder con la configuración pulsaremos la tecla `Enter`, apareciendo la pantalla de selección del protocolo de ratón. Introducimos el número de la opción elegida (normalmente es la 4. PS/2 Mouse) seguida de un `Enter`. Se nos preguntará si queremos emular el tercer botón (para aquellos ratones con sólo dos botones). Si respondemos afirmativamente al pulsar el botón izquierdo y derecho simultáneamente obtendremos el mismo efecto como cuando pulsamos el tercer botón. Respondemos con y/n y apretamos `Enter` para poder llegar a introducir el nombre del dispositivo del ratón (por defecto /dev/mouse, y en mayoría de los casos nos servirá /dev/psaux). Posteriormente se nos pedirá que introduzcamos el número correspondiente a nuestro teclado de entre los listados en la pantalla (los del tipo generic en caso de duda). A continuación tendremos que elegir el país (España - 42). Seguidamente podremos poner un alias para el idioma anteriormente elegido. Pulsando a `Enter` se pondrá la opción por defecto, y el programa nos preguntará si queremos activar las opciones adicionales del XKB (en caso de duda `n` seguido de `Enter`).

A continuación accederemos al apartado de configuración de nuestro monitor. Observando las opciones mostradas en la pantalla y eligiendo las adecuadas avanzaremos dos apartados y en el siguiente se nos pedirá que escribamos un identificador para el monitor. Escribe una cadena con su nombre, descripción o lo que quieras y pulsa la tecla `Enter` para poder elegir nuestra tarjeta de vídeo.

Si leemos la pantalla averiguamos que al pulsar `y + Enter` nos aparecerá un listado de tarjetas soportadas (al pulsar `Enter` continúa la lista) del cual tendremos que introducir el número correspondiente a nuestro modelo de tarjeta. Seguidamente se nos pedirá el tamaño de la memoria de la tarjeta de vídeo (4096K son 4M, 32768K son 16M ...). Al igual que en el caso del monitor se nos pedirá una cadena para identificar la tarjeta de vídeo. Elegiremos la profundidad de color que deseamos que sea por defecto y estaremos en el último paso que es el de guardar el fichero de configuración. Si estamos seguros de haber elegido las opciones adecuadas responderemos afirmativamente o en el caso contrario abortaremos el programa y volveremos desde principio.

El proceso anterior a simple vista parece demasiado complicado, pero es sencillo con un poco de paciencia. Destacar lo dicho al principio, que cada distribución tiene su propia herramienta de configuración del servidor X, las cuales resultan bastante más sencillas de usar. Algunos fabricantes de tarjetas de vídeo como NVIDIA (www.nvidia.com) proporcionan drivers de sus tarjetas para Linux, así como los drivers OpenGL y documentación sobre su instalación.

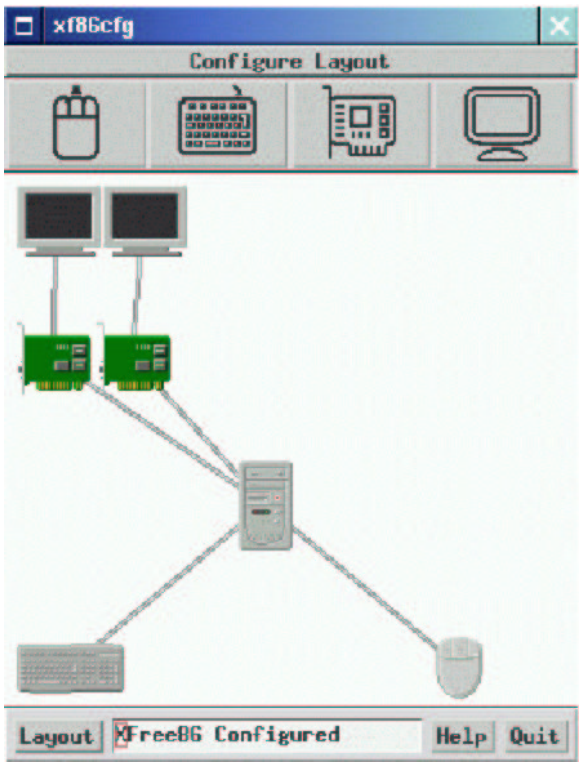


Figura 7.7: xf86cfg. Utilidad para configurar el servidor X

7.10. Configuración del acceso a internet

Este apartado lo vamos a dividir en tres subbloques dependiendo del tipo de nuestra conexión. Vamos a hablar de como configurar el acceso a internet mediante un módem (externo o interno), una línea ADSL con router ADSL, y una conexión mediante un cable - módem.

7.10.1. Acceso mediante modem

Casi todos los modems externos están soportados por el kernel de Linux; en cambio hay muy pocos módem internos para los cuales hay soporte en nuestro S.O. favorito. Actualmente existe soporte para los siguientes:

- Modems que hagan uso de los chipsets Lucent Apollo (ISA) y Lucent Mars (PCI)
- Modems Intel V90 HaM e Intel 536ep
- Modem IBM Mwave (usado en el Thinkpad 600E y posteriores)
- Modems Conexant que hagan uso de los chipsets Conexant HCF y HSF
- Modems ESS ISA

Toda la documentación sobre la instalación y los drivers para los modems anteriores se pueden encontrar en la página www.linmodems.org.

Antes de empezar con la configuración de nuestro módem nombrar como en casos anteriores que existen herramientas específicas de distintas distribuciones, escritorios o aplicaciones (como es el caso de *kppp* de KDE el cual es intuitivo y bastante sencillo de configurar). Otra vez volveremos a usar una aplicación incluida en todas las distribuciones llamada **pppd** y la utilidad de configuración del programa anterior **pppconfig**. También nombrar que antes de empezar con el proceso debemos tener instalado el pppd y activado el soporte para el protocolo PPP (*Point to Point Protocol*) en el kernel, apartado *Network devices*, el cual suele estar activado al instalar nuestra distribución.

Gracias a una herramienta como *pppconfig* la tarea de configuración es muy sencilla, basta con ejecutar el programa e ir introduciendo los datos requeridos por el mismo. Esta utilidad modifica los ficheros de configuración, así que antes de

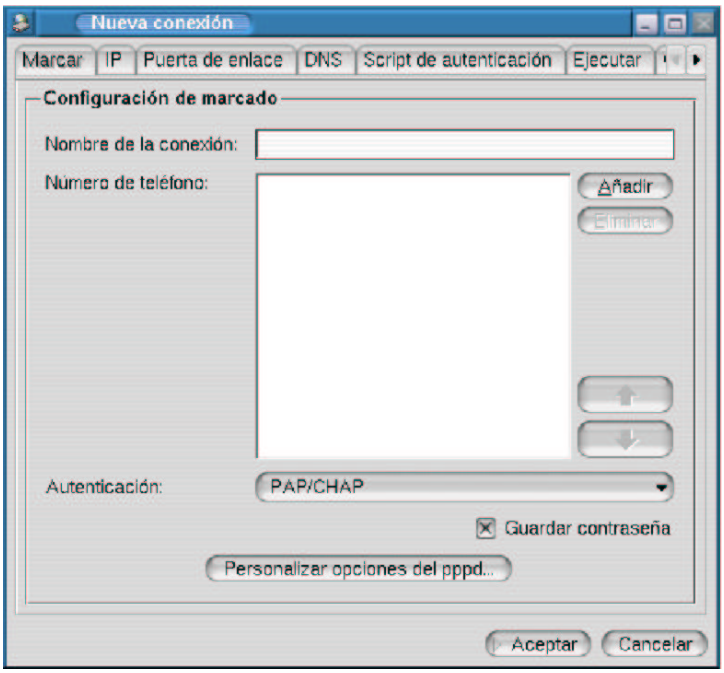


Figura 7.8: KPPP. Interfaz GUI de pppd para KDE

empezar necesitamos acceder como root y a continuación escribir en la línea de comandos `pppconfig`. Seleccionamos la opción marcada (crear una conexión nueva) y posteriormente introducimos un identificador para nuestra conexión (como ejemplo usaremos ñueva-conexion'). Elegimos nuestro tipo de DNS (casi siempre estático, dependiendo de nuestro proveedor de acceso a internet), seleccionamos <OK> y seguidamente introducimos la dirección del servidor DNS primario y secundario si tenemos (ambas direcciones son proporcionadas por el proveedor de acceso).

En este momento habrá que seleccionar el método de autentificación. PAP suele funcionar y si no es así prueben CHAT o CHAP. Introducimos el nombre de usuario, la contraseña, la velocidad de nuestro módem (deja la opción por defecto en caso de duda - 115200), método de marcación (tonos o pulsos) y el número de teléfono del acceso a internet.

Enciende el módem si es externo porque a continuación el programa intentará autodetectar el puerto de nuestro módem. En el caso de que no lo consiga tendremos que introducirlo nosotros. En algunos casos estará en /dev/modem. Si lo anterior tampoco funciona debemos saber que los puertos COM son dispositivos que se encuentran en /dev y se relacionan mediante la siguiente tabla:

Dispositivo	Puerto
/dev/ttyS0	COM1
/dev/ttyS1	COM2
/dev/ttyS2	COM3
/dev/ttyS3	COM4

Normalmente los modems externos usan el COM1 o el COM2, mientras los internos usan el COM3 o el COM4. Anotar que /dev/modem es simplemente un enlace a alguno de los /dev/ttySx. Introduciremos el dispositivo correspondiente (/dev/ttySx) y habrémos acabado con el proceso. Guardamos los cambios y abandonamos al asistente pppconfig.

Los ficheros de configuración por defecto se guardan en el directorio /etc/ppp. Buscamos la ruta del fichero con el nombre ñueva-conexión'(u otro introducido como identificador) y ejecutamos el siguiente comando:

```
$ pppd file /ruta/del/fichero/nueva-conexion
```

Lo cual normalmente corresponde a lo siguiente:

```
$ pppd file /etc/ppp/peers/nueva-conexion
```

Podremos observar que es lo que está sucediendo en el fichero `/var/log/messages` con el comando:

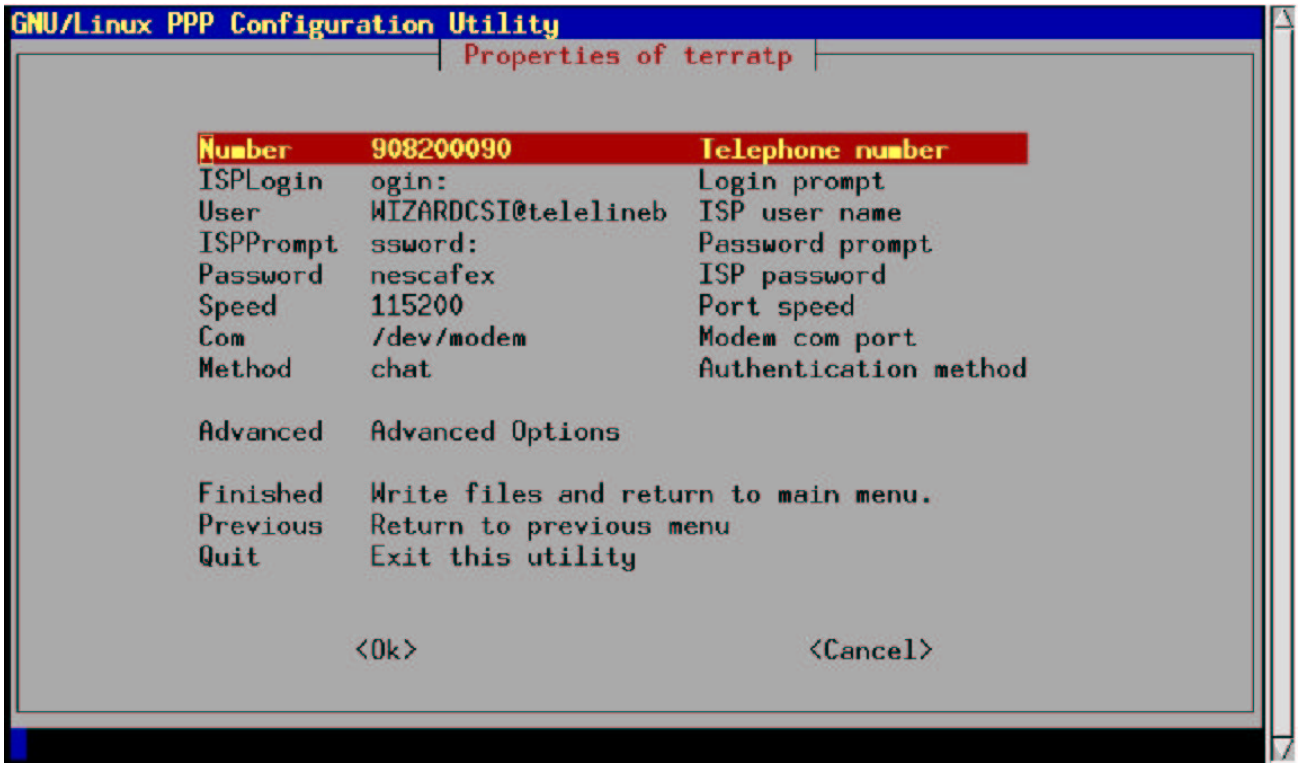


Figura 7.9: PPPConfig

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

Podemos finalizar nuestra conexión matando el proceso pppd con un comando kill (`killall pppd`).

Por último recordar que existen muchos otros asistentes de configuración algunos de los cuales pueden simplificar el proceso de configuración.

7.10.2. Acceso mediante ADSL y cable modem

La configuración de acceso usando una línea ADSL que vamos a explicar en este apartado es usando un Router ADSL el cual debe de estar previamente configurado. También necesitaremos una tarjeta Ethernet que funcione en Linux.

Para poder realizar el acceso tenemos que configurar la interfaz de red. Normalmente usaremos el programa **ifconfig** con una orden similar a la siguiente (root):

```
$ ifconfig eth0 192.168.0.2 netmask 255.255.255.0 up
```

Con el comando anterior estamos configurando la interfaz Ethernet eth0, asignándole una dirección IP 192.168.0.2 y la máscara de subred 255.255.255.0. En internet se pueden encontrar varios tutoriales y manuales de configuración de red, así que no nos vamos a extender demasiado en este punto. El paso siguiente es establecer un camino por defecto (**gateway**). Lo haremos con este comando:

```
$ route add default gw 192.168.0.1 eth0
```

donde 192.168.0.1 es la dirección IP de nuestro router ADSL.

Para no tener que repetir este proceso cada vez que iniciamos el ordenador podemos poner la información sobre nuestra interfaz de red y el gateway en un fichero de configuración que normalmente están en el directorio /etc y tienen nombres similares a network, networks ...; en la distribución Debian se encuentra en */etc/network/interfaces* y cuyo aspecto es parecido al siguiente:

```
auto eth0
iface eth0 inet static
   address 192.168.1.50
   netmask 255.255.255.0
   network 192.168.1.0
   broadcast 192.168.1.255
   gateway 192.168.1.1
```

En otras distribuciones existen utilidades de configuración de red como netcfg (RedHat) o netconfig (Slackware).

En un acceso a internet a través de un cable módem la autentificación de usuario y la dirección IP se obtiene de manera dinámica a través del servicio DHCP. Además de lo anterior se obtendrá la máscara de subred, dirección de gateway y las direcciones DNS; es decir todo se configura por si mismo. Para su uso tendremos que tener instalado un cliente DHCP, por ejemplo dhcp-client. En el fichero *interfaces* bastará poner:

```
auto eth0
iface eth0 inet dhcp
```

para poder usar este servicio. En los distintos asistentes de configuración seleccionaremos el uso de DHCP y con esa operación tendremos configurada la interfaz de red y por lo tanto el acceso a internet a través de un cable módem.

7.11. Configuración de la impresora: CUPS

En Linux podemos utilizar varios sistemas de impresión y nos sería imposible explicarte como se configuran todos ellos, aquí nos vamos a centrar en uno en concreto, CUPS, *Common Unix Printing System*.

Una de las ventajas que tiene este sistema frente a los otros es su facilidad de configuración ya que podemos configurar completamente nuestras impresoras desde un navegador.

El cups está diseñado para que podamos utilizar una misma impresora o un grupo de éstas desde cualquier PC que esté en la red, es decir, que un amigo siempre que tu le des permiso y estés conectado a internet podrá utilizar tu impresora para sacar a tiempo ese trabajo que tiene que entregar mañana y no le ha dado tiempo de ir a comprar tinta para la impresora o se ha quedado sin papel o mil cosas que suelen pasar.

Para configurar el sistema simplemente tendremos que abrir un navegador y en la barra de dirección colocaremos: `http://localhost:631`, en ese momento, si nosotros no tenemos permiso para configurar el sistema de impresión nos saldrá un cuadro de diálogo en el que tendremos que introducir el nombre de usuario y el password de un usuario del sistema con permiso para configurar el CUPS, por ejemplo root. Una vez obtenido acceso a la configuración de CUPS en el navegador aparecerá una página web como esta:

Para comenzar con la configuración pulsaremos sobre `Do Administration Tasks` en ese momento nos aparecerán tres categorías sobre las que podemos operar: clases, jobs, printers.

Las clases son grupos de impresoras con unas determinadas características que se unen para hacer más fácil la gestión de los documentos. Por ejemplo, si en una oficina se tienen 3 impresoras láser, 2 chorro de tinta a color y 5 matriciales, podríamos hacer tres grupos: grupo láser, donde meteríamos las impresoras láser; grupo tinta donde colocaríamos las impresoras a chorro de tinta y grupo matricial donde estarían las impresoras matriciales. Con estos grupos confeccionados cuando quisiéramos imprimir un documento simplemente tendríamos que elegir uno de los tres grupos y nuestro documento se imprimirá en la impresora que antes quede libre de trabajo del grupo seleccionado.

Los jobs o trabajos son los trabajos que hay en este instante en cola para imprimir, podremos eliminarlos, detener su impresión, etc. Los trabajos que ya han sido terminados también podremos reanudarlos y conocer varios datos sobre ellos.

Las impresoras son los dispositivos de impresión que tenemos en nuestro sistema podremos añadir impresoras nuevas o gestionar las que tenemos. Como nosotros vamos a instalar nuestra impresora por primera vez pulsaremos sobre `Add Printer`.

Tendremos que ir facilitando los datos que se nos solicitan, lo primero es asignarle un nombre, decir donde se encuentra y añadir una breve descripción al nombre.

Lo siguiente es seleccionar donde tendremos conectada nuestra impresora, normalmente tendremos conectada la impresora al puerto paralelo (`Parallel Port #1`)

En la siguiente página tendremos que seleccionar el fabricante de nuestra impresora.

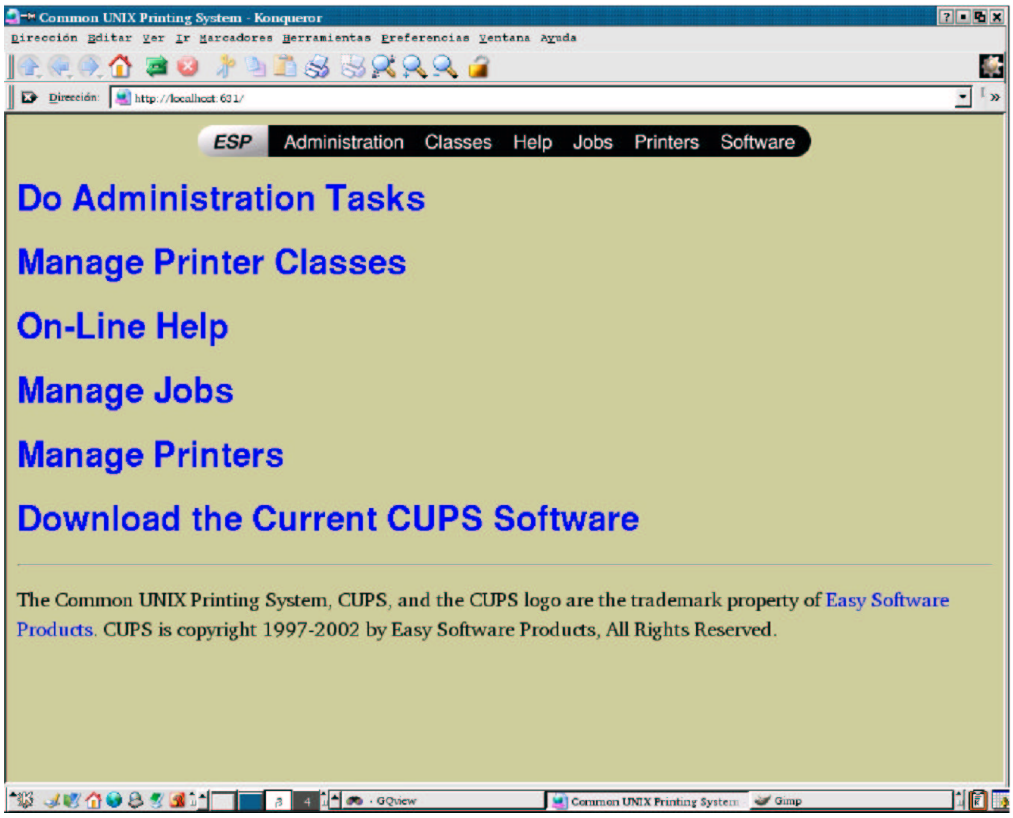


Figura 7.10: Página web que CUPS sirve en el puerto 631

Figura 7.11: Configuración de CUPS

A continuación elegiremos el modelo de impresora.

y listo, nuestra impresora ya está instalada, simplemente faltan los últimos retoques. Pulsaremos sobre el nombre que le hemos dado a nuestra impresora y aparece la siguiente pantalla:

Ahora pulsamos sobre `Configure Printer` y seleccionamos el tipo de papel que queremos usar, la calidad de impresión y ese tipo de cosas. Ya tenemos nuestra impresora lista para imprimir cualquier tipo de documento. Si queremos comprobar como imprime nuestra impresora podemos pulsar en la pantalla anterior a `print test page` e imprimirá la página de prueba de CUPS.

TEMA 8

Administración básica

8.1. Paquetes de instalación

Un paquete es un archivo que contiene varios ficheros que permiten la instalación de un programa. En linux existen varios tipos de paquetes:

RPM: es el tipo de paquetes más popular. Creada por Red Hat y que utilizan la mayoría de las las distribuciones.

DEB: creada por el grupo Debian.

TARBALLS: éstos son ficheros comprimidos que contienen el código fuente del programa y pueden instalarse en cualquier distribución.

Una parte importante y muy delicada de un sistema de paquetes son las dependencias. Cada paquete necesita que estén instalados en el sistemas otros paquetes que son necesarios para su funcionamiento, los paquetes necesarios para su instalación se llaman dependencias. Si esto no funciona, no se puese asegurar que el sistema funcione con estabilidad y fiabilidad al sistema GNU/Linux.

8.1.1. Sistema de paquetes RPM

Este sistema de paquetes fue creado por la compañía Red Hat. De ahí viene el nombre RPM, **Red Hat Package Manager**. Con el tiempo otras distribuciones han ido adoptando este sistema de empaquetado de ficheros. Las principales distribuciones que lo adotan son: Red Hat, SuSE y Mandrake.

Un paquete de software construido con RPM es un conjunto de ficheros e información asociada a ello, como un nombre, una versión y una descripción. que terminan con la extensión `.rpm`. A diferencia de el sistema de Debian, cuando queremos instalar un paquete rpm, hemos de bajárnoslo nosotros mismos y efectuar las operaciones de instalación. La aplicación que se utiliza para la instalar estos paquetes es `rpm`. Ésta es su forma de utilización:

8.1.1.1. Instalación de paquetes RPM

RPM se utiliza principalmente para instalar de paquetes software. La sintaxis general de una orden de instalación `rpm` es:

```
rpm [opciones] [paquetes]
```

donde `opciones` puede ser alguna de las siguientes:

`-v`: Muestra por pantalla el trabajo que está realizando RPM.

`-h o --hash`: Imprime en pantalla el carácter # muchas veces seguidas mientras se está instalando el paquete para marcar el proceso de instalación.

`--percent`: Muestra en pantalla el porcentaje realizado mientras se extraen los ficheros de un paquete.

`--test`: Realiza el proceso de instalación de un paquete, pero no instala nada. Se utiliza más bien para detectar problemas que pudieran surgir en la instalación.

`--nodeps`: No se realizan verificaciones de dependencias antes de la instalación de un paquete.

`--force`: Fuerza la instalación de un paquete aunque surjan problemas de conflictos.

Cuando se le pasan opciones a RPM, independientemente del modo, todas las opciones indicadas por una sola letra se pueden unir formando un único bloque. Así, si se hace:

```
rpm -i -v -h vim-4.5-2.i386.rpm
```

sería equivalente a

```
rpm -ivh vim-4.5-2.i386.rpm
```

Observemos que este paquete sigue el convenio de nomenclatura:

```
nombre-versión-emisión.arquitectura.rpm
```

Puede ocurrir que se intenta instalar el paquete "dosemu"

```
# rpm -ivh dosemu-0.66.2-1.i386.rpm

failed dependences: kernel >= 2.2.16 is needed by dosemu-0.66.2-1
dosemu = 0.64.1 is needed by xdosemu-0.64.1-1
```

Estos mensajes indican dos cosas: Es necesario actualizar a la versión 2.2.16 del kernel y, si se instala, una versión más moderna de `dosemu`, también habrá queinstalar una versión más moderna de `xdosemu`. Aunque no suele ser una buena idea ignorar estos errores, en caso de que se quiera instalar sin atender a estos mensajes, sólo hay que usar la opción `--nodeps`.

8.1.1.2. Actualización de paquetes RPM

El modo de actualización de RPM proporcina un método sencillo renovar paquetes ya instalados a versiones más modernas. Su uso es similar al de la instalación:

```
rpm -U [opciones] [paquetes]
```

8.1.1.3. Desinstalación de paquetes RPM

El modo de desinstalación de RPM proporciona un método limpio para eliminar los archivos que pertenecen a un determinado paquete y que se encuentran en diferentes lugares.

Muchos paquetes instalan ficheros en `/etc`, `/usr` y `/lib`, por lo que puede ser complicado eliminarlos, pero con RPM, se puede desinstalar un paquete completo con la orden:

```
rpm -e [opciones] [paquetes]
```

Hay que tener en cuenta que para desinstalar sólo hay que indicar el nombre de la aplicación, no el del paquete, como se hace en el proceso de instalación.

8.1.1.4. Pidiendo ayuda a RPM

Como es obvio, RPM tiene muchísimas opciones y también puede ocurrir que no nos acordemos de alguna. Entonces podemos pedirle ayuda a RPM:

```
rpm -h
```

Con esto tenemos para saber manejar el RPM de forma básica. El resto son horas y ganas.

8.1.2. Sistema de paquetes Deb

Todos los paquetes deb tienen el siguiente formato: `nombre-del-paquete_version(1.3.34-5).deb`

La distribución Debian tiene diversas utilidades para la instalación de paquetes, entre ellas, APT, que permite la instalación de paquetes de forma fácil y rápida, advirtiendo de las dependencias y recomendando paquetes. El sistema APT engloba varios comandos como apt-get, apt-cache, apt-cdrom,...

8.1.2.1. El fichero /etc/apt/sources.list

Este fichero es imprescindible para la instalación de paquetes con APT. En él se guardan las direcciones de donde APT se descarga los paquetes. Los medios por los que se pueden descargar los paquetes son varios: file(podemos elegir un directorio albitrario de donde bajarnos los paquetes, esto es útil para mirrors locales o carpetas NTFS),de un cdrom,de un servidor web(http), de un ftp, por rsh/ssh.

Vamos a ver un ejemplo:

```
deb http://http.us.debian.org/debian woody main contrib non-free
deb http://non-us.debian.org/debian-non-US woody/non-US main contrib non-free
deb-src http://http.us.debian.org/debian woody main contrib non-free
```

La diferencia entre `deb` y `deb-src` es que el primero indica la descarga de paquetes .deb, que son ficheros binarios, es decir, preparados para ejecutarse, mientras que con el segundo podemos descargarnos el código fuente del paquete(usando el comando apt-get source).

La siguiente parte de la línea es el `URI`, es decir, el tipo de sistema para la descarga, recordemos que existen varios(file,cdrom,ftp,http,rsh/ssh). En este caso es de un servidor web.

Seguidamente escribimos la `localización` del mirror de paquetes, este caso tenemos varias líneas con diferente localización, esto se debe a que en los Estados Unidoas es ilegal utilizar aplicaciones de encriptación, así que para bajar esos programas, existen líneas especiales que contienen la palabra non-US. Después de la localización, separado por un espacio, se escribe la `versión de debian`, es válido tanto el alias de la version como en qué estado se encuentra (stable, unstable,testing).

Por último se escriben las `secciones de software` que usaremos (main, contrib, non-free). Debian organiza los paquetes en varias carpetas segun su licencia.

- La sección `main` agrupa los paquetes en los que su licencia cumple con los criterios de la DGFS("Guías de Debian del Software libre").
- La sección `contrib` agrupa paquetes que tiene una licencia libre pero que sin embargo dependen de otros paquetes que no cumple con las normas del DGFS.
- Y por último, la sección `non-free` contienen paquetes que son de libre distribución pero que sin embargo no cumplen las directrices de la DGFS (no distribuye el código, no se permite redistribuir el código,etc).

8.1.2.2. Apt-get

El comando **apt-get** se utiliza para la manipulación de paquetes deb. Permite la instalación de paquetes, borrado, ...

`apt-get install paquete1 paquete2 ...` Instala paquetes.

`apt-get remove paquete1 paquete2 ...` Borra paquetes.

`apt-get source paquete1 paquete2 ...` Descarga el código fuente de los paquetes.

`apt-get update` Actualiza la lista de paquetes disponibles para instalar.

`apt-get upgrade` Instala las nuevas versiones de los diferentes paquetes disponibles.

`apt-get dist-upgrade` Función adicional de la opción upgrade que modifica las dependencias por la de las nuevas versiones de los paquete.

`apt-get build-dep paquete1 paquete2 ...` Instala los paquetes necesarios para la compilación del código fuente de los paquetes.

`apt-get clean` Elimina los ficheros que se encuentran en /var/cache/apt/archives y /var/cache/apt/archives/partial. Ahí se encuentran los paquetes que hemos descargado para instalar.

`-d,--download-only` Sólo descarga el paquete, no lo instala.

`-f,--fix-broken` Esta opción es importante, intenta arreglar problemas de dependencias que tengamos en el sistema.

`-s,--simulate` Nos muestra los resultados de la instalación de un paquete.

`-b,--build` Compila el paquete de código fuente que hayamos bajado.

8.1.2.3. Apt-cache

El comando apt-cache trabaja con la caché de los paquetes. Este comando no manipula el estado del sistema, así que lo pueden usar usuarios normales. Es de gran utilidad ya que nos muestra información valiosa sobre los paquetes.

Algunas opciones más importantes:

apt-cache show paquete1: Este comando muestra la cabecera de los paquetes. Muestra el desarrollador, las dependencias, una breve descripción del mismo, su tamaño, el nombre del fichero donde se encuentra, entre otros.

apt-cache search texto: Muestra una lista de todos los paquete y una breve descripción relacionado con el texto que hemos buscado.

apt-cache depends paquete: Muestra las dependencias de dicho paquete.

apt-cache stats: Muestra la estadística de el cache.

8.1.2.4. El fichero `/etc/apt/apt.conf`

El fichero apt.conf sirve para la configuración por defecto de APT. En el fichero podemos, por ejemplo, darle las órdenes al APT para el uso de un proxy. Podemos encontrar un ejemplo del fichero en

`/usr/share/doc/apt/examples/configure-index.gz`

8.1.2.5. Apt-cdrom

El comando apt-cdrom permite añadir nuevos CD-ROM's al sources.list. Para añadir un cdrom la orden es `apt-cdrom add`

8.2. La gestión de los procesos.

Un proceso es cualquier instrucción o programa que en ese momento se está ejecutando en nuestro sistema. Todo proceso tiene un PID (Process IDentifier), es decir, un número que le identifica y le diferencia de todos los demás. Una caracterísa importante es que todo proceso tiene un estado: corriendo, durmiendo, zombie o parado.

8.2.1. El comando kill

El comando kill nos permite interactuar con cualquier proceso mandando señales (signal). Cuando ejecutamos `kill pid` lo que hacemos es mandar la señal de TERM(terminar) con lo cual se termina ese proceso. Podemos usar cualquier otro tipo de señal, para ello utilizamos `kill signal pid`. Podemos conseguir una lista de señales usando `kill -l`. Una señal útil para alunas ocasiones es `-9`, esta señal fuerza a terminar cualquier proceso. Como su nombre indica, estamos matando el proceso.

También podemos utilizar el comando `killall` con el que podemos mandar señales a un proceso utilizando el nombre, en vez del PID.

Entre los procesos diferenciamos los que se están ejecuntando en 1^{er} o 2^{o} plano. Los que se ejecutan en primer plano son los que interactúan con el usuario en ese momento, mientras que los procesos en segundo plano se ejecutan pero están ocultos, y muy posiblemente el usuario no tenga constancia de que se esté ejecutando.

Sólo puede haber un proceso en primer plano por consola. Eso nos deja las manos atadas si no estamos en el entorno gráfico. Para poder ejecutar varios comandos, lo que podemos hacer es ejecutar los comandos en segundo plano. Para ello solo tenemos que añadir `&` al final del comando. Vamos a poner un ejemplo:

```
$ls -R / > /dev/null &
```

En el anterior ejemplo listamos todos los ficheros de todos los directorios del sistema. Enviamos la salida a /dev/null para que su salida no nos moleste. El carácter `&` manda el proceso a segundo plano.

El comando `jobs` nos muestra los procesos que se están ejecutando en segundo plano:

```
$ls
[1]+  Running                 ls --color -R / >/dev/null &
```

Aquí estamos ejecutando el comando anterior. El elemento `[1]` nos indica el número del proceso que se están ejecutando en segundo plano y cuál es su estado. En este caso `Running`(corriendo). Seguidamente nos muestra cuál es el proceso Podemos utilizar también el comando `fg` para mandar un proceso al primer plano y el comando `bg` para mandar el proceso al segundo plano.

```
$fg
ls --color -R / >/dev/null
```

`fg` manda el proceso al primer plano y nos muestra el programa que ha mandado. Si tenemos varios procesos en segundo plano añadimos el número del proceso.

El comando `bg` se utiliza cuando tenemos, por ejemplo, procesos suspendidos. Estos procesos son programas que están "parados", es decir, no consumen ni CPU ni memoria, y que podemos volver a poner en archa en cualquier momento. Para suspender un proceso utilizamos la combinación de teclas `C-z`, al igual que para interrumpir un proceso utilizamos `C-c`.

```
$jobs
[1]+  Stopped                 ls --color -R / >/dev/null
```

Esta tarea está parada(Stopped).

8.2.2. El comando ps

El comando `ps` permite mostrar todos los procesos que están corriendo en nuestro sistema. Veamos una parte de una salida del comando ps:

```
$ps -aux
faraox@menut:~/doc/glup_0.6-1.1-html-1.1$ ps xau
USER       PID %CPU %MEM    VSZ  RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2   1272  436 ?        S    16:00   0:04 init [2]
root         2  0.0  0.0      0    0 ?        SW   16:00   0:01 [keventd]
root         3  0.0  0.0      0    0 ?        SW   16:00   0:00 [kapmd]
faraox    1363  0.0  0.8   2740 1564 pts/2    S    18:57   0:00 -bash
```

Los parámetros xau nos permiten ver todos los procesos que se están ejecutando. El parámetro `a` muestra lo que se está ejecutando en las tty conocidas, el parámetro `x` añade los procesos que no se conece la tty en la que se están ejecutando y `u` muestra los usuarios que están ejecutando esos procesos.

Algunas partes de la salida le serán conocidas. La columna "USER" nos dice que usuario está ejecutando el proceso,"PID" es su número de proceso, "%CPU" es el porcentaje de CPU que está utilizando al igual que "%MEM" es el porcentaje de memoria. También incluye la cantidad de memoria en kylobytes que ha utilizado dicho proceso, se muestra en la columan "RSS".La columna "TTY" muestra la consola desde la que se está ejecutando. "STAT" nos muestra el estado del proceso:S(drmiendo), R(corriendo), T(parado), Z(zombie). Las opciones "W" y "N" son especiales para procesos del kernel. La columna "START" muestra la hora a la que empezó el proceso, y la columna "TIME" muestra el tiempo de CPU que ha usado el proceso desde que se inició y "COMMAND" muestra el nombre del comando que se está ejecutando.

8.2.3. El comando top

El comando top es una utilidad que permite la monitorización de los procesos de la CPU. También muestra el estado de la memoria. Es una mezcla del comando uptime, free y ps.

```
20:07:54 up  4:07,  5 users,  load average: 0.07, 0.05, 0.05
60 processes: 58 sleeping, 1 running, 0 zombie, 1 stopped
CPU states:   0.4% user,   0.6% system,   0.0% nice,  99.0% idle
Mem:    182900K total,   172404K used,    10496K free,    35064K buffers
Swap:    96352K total,    14284K used,    82068K free,    43228K cached

PID USER     PRI  NI  SIZE  RSS SHARE STAT %CPU %MEM   TIME COMMAND
1565 faraox    14   0  1040 1040   820 R     0.5  0.5   0:00 top
300 root       9 -10 24736 9.9M  1524 S <   0.1  5.5   2:47 XFree86
1541 faraox    10   0  3148 3148  2184 S     0.1  1.7   0:00 Eterm
1 root       8   0   480  436   416 S     0.0  0.2   0:04 init
```

espacio	Actualiza la pantalla
C-l	Borra y reescribe la pantalla
k	Mata un proceso(kill)
r	Cambia la prioridad de cualquier proceso
s	Cambia el intervalo de refresco(por defecto es cada 5 segundos)
o	Cambia el orden de los elementos
N	Ordenar procesos por PID
P	Mostrar procesos por el uso de la CPU(por defecto)
M	Mostrar procesos por el uso de memoria
T	Mostrar procesos por tiempo
W	Guarda la configuración en /.toprc

Tabla 8.1: Teclas para la interactuación con el top

8.2.4. Nice: prioridad en procesos

Una CPU tiene que compartir su tiempo de cálculo con varios procesos. `Nice` es un programa que permite cambiar la prioridad de un proceso en nuestro sistema. La prioridad tiene un rango desde -20 a 20. Un usuario normal tiene prioridad 0 cuando ejecuta cualquier comando desde la consola. Con el comando nice, ese usuario normal sólo puede bajar la prioridad del proceso, nunca subirlo, sólo root es capaz de ello. El esquema de utilización sería: nice -n prioridad comando o también podemos utilizar el comando `rnice` para cambiar la prioridad de un proceso: `renice prioridad PID`

8.3. Los servicios en Debian

Los servicios o demonios son procesos que se ejecutan automáticamente al arrancar el sistema o al llamarlos y que esperan cualquier petición. Es el caso por ejemplo de Apache, Qmail, SSH, etc. Los servicios normalmente se inician al iniciar el sistema, pero podemos iniciarlo(start), pararlo(stop) o reiniciándolo, sería así: `/etc/init.d/servicio opción`.

Para que el servicio no se inicie cuando arrancamos el sistema, podemos usar el comando update-rc.d. Este comando crea enlaces a los diferentes directorios de runlevels. Para que no se inicie automáticamente: `update-rc.d -f servicio remove`. Para añadir un servicio, simplemente creamos el script de nuestro servicio en Bourne Shell Script y lo copiamos al directorio /etc/init.d/. Si queremos que se inicie automáticamente al arrancar escribimos: `update-rc.d servicio defaults`

8.4. Los archivos de registro

Los ficheros de registro (logging files) se almacenan los resultados e información útil de algunos programas. Éstos son muy importantes para conocer el estado de nuestra máquina. Los ficheros de registro más importantes se encuentran en la carpeta `/var/log/`. Normalmente, estos ficheros se pueden visualizar con cualquier editor de texto ya que están guardados en un fichero ASCII.

syslog: El fichero syslog es el resultado de los mensajes del demonio Syslogd. Este demonio se encarga de la organización de los mensajes del kernel, así que en el fichero syslog podemos enontrar toda la información sobre lo que ocurre en nuestra máquina.

messages: Este fichero es igual al syslog, pero muestra información más simple. En el podemos controlar, por ejemplo, la información del demonio pppd.

~/.bash_history: Estos ficheros muestran los comandos que han sido ejecutados por los usuarios desde la consola.

wtmp: En el hay un listado de todas las conexiones que ha tenido la máquina mientras ha permanecido encendida. Se puede leer con el comando `last`.

Normalmente cada demonio tiene un fichero de registro al igual que otros muchos programas. Sólo hay que mirar la documentación. Los ficheros de registro de nuestro servidor de correo se encuentran en /var/log/mail.log, así como los del Apache se encuentran en /var/log/apache,etc.

8.5. Otras utilidades para la administración

uptime

El comando `uptime` nos indica el tiempo que ha estado corriendo la máquina.

```
$ uptime
17:56:33 up  1:55,  8 users,  load average: 0.05, 0.01, 0.01
```

El primer elemento es la hora actual. El siguiente elemento, seguido de la palabra "up" es el tiempo que la máquina está encendida. Seguidamente nombra el número de usuarios que se encuentran en el sistema. Por último muestra la carga media de la máquina, en tres tiempos, 1,5 y 15 minutos.

w

Muestra los usuarios que se encuentran en el sistema. Veamos un ejemplo:

```
$w
18:00:59 up  2:00,  8 users,  load average: 0.01, 0.02, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
faraox   tty1     -                16:01    1:58m  2.69s  0.00s  w
```

La primera línea que muestra el comando `w` es la salida de el comando `uptime`. Por orden, la información que muestra es el nombre de usuario, la consola desde donde ha entrado, desde donde se conecta, la hora de entrada, el tiempo que ha permanecido inactivo (idle), el tiempo usado por todos los procesos de esa consola(tty), el tiempo usado por los procesos actuales y lo que está haciendo el usuario.

free

El comando `free` muestra información sobre el estado de la memoria del sistema. Muestran tanto el estado de la memoria física como de la swap. También muestra el búffer utilizado por el kernel. Una salida del comando `free`

```
$free
     total       used       free     shared    buffers     cached
Mem:     182900     173300      9600          0      11796      66588
-/+ buffers/cache:      94916      87984
Swap:    96352        448      95904
```

Este comando lee la información del fichero /proc/meninfo.

dmesg

Este comando muestra los mensajes del kernel durante el inicio del sistema.

8.6. Grupos y usuarios

Como es sabido, Linux es un sistema multiusuario. Eso no sólo significa que cada uno puede tener su cuenta y trabajar en él sino además puede hacerlo al mismo tiempo que otro usuario que esté trabajando en ese instante.

Una de las competencias del administrador es gestionar la parte de usuarios y englobarlos dentro de grupos de trabajo.

8.6.1. Gestión de usuarios

Para la gestión de usuarios hay dos operaciones: Adición y eliminación.

8.6.1.1. Añadiendo nuevos usuarios

En Debian, para añadir un usuario, se utiliza el comando `adduser nombre`. En ese momento, no sólo se creará la cuenta del usuario sino también su directorio de trabajo, un nuevo grupo de trabajo que se llamará igual que el usuario y añadirá una serie de ficheros de configuración al directorio de trabajo del nuevo usuario:

```
root@cila:/home# adduser pepe
Adding user pepe...
Adding new group pepe (1000).
Adding new user pepe (1000) with group pepe.
Creating home directory /home/pepe.
Copying files from /etc/skel
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for pepe
Enter the new value, or press return for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct?
[y/n] y
```

En ese momento, el usuario ya puede trabajar en el sistema.

8.6.1.2. Eliminando usuarios

Para eliminar la cuenta de un usuario, emplearemos el comando `deluser nombre`. La pega de este comando es que no elimina automáticamente el directorio de trabajo del usuario.

```
root@cila:/home# deluser pepe
Removing user pepe...
done.
```

Una vez realizado este proceso, es responsabilidad del administrador decidir si elimina el directorio de trabajo del antiguo usuario.

Si tratáramos de eliminar un usuario que no existe en el sistema, recibiremos el siguiente aviso:

```
root@cila:/home# deluser pepe
/usr/sbin/deluser: ‘pepe’ does not exist.
```

8.6.2. Gestión de grupos

Cuando creamos un usuario, siempre lo vamos a incluir en algún grupo de trabajo, ya sea el suyo propio o bien, en uno común.

8.6.2.1. Añadiendo nuevos grupos

La forma de hacerlo es bien fácil:

```
root@cila:/home# addgroup usuarios
Adding group usuarios (105)...
Done.
```

El número "105" nos indica que ése el identificador numérico que se le asigna al nuevo grupo en el momento de su creación.

8.6.2.2. Eliminando grupos

De forma similar, la eliminación de un grupo se hace de esta forma:

```
root@cila:/home# delgroup usuarios
Removing group usuarios...
done.
```

¿Qué puede pasar si tratamos de eliminar un grupo inexistente? El sistema nos avisará con el siguiente mensaje:

```
root@cila:/home# delgroup usuarios
/usr/sbin/delgroup: 'usuarios' does not exist.
```

8.6.2.3. Añadiendo y eliminando usuarios de los grupos

Para añadir un usuario **pepe** a un grupo **usuarios** haremos:

```
root@cila:/home# adduser pepe usuarios
Adding user pepe to group usuarios...
Done.
```

Y para eliminarlo de ese grupo:

```
root@cila:/home# deluser pepe usuarios
Removing user pepe from group usuarios...
done.
```

Con esto queda por terminada esta introducción a la administración en Linux.

TEMA 9

Programación en Bash

En capítulos anteriores hemos visto algunos aspectos del intérprete de comandos (o shell en la literatura inglesa). Como en muchos otros casos la variedad de intérpretes de comandos existente es muy amplia. Sin embargo existe uno que ha destacado sobre los demás, o al menos que ha sabido ocupar el puesto de estándar de facto en el mundo GNU/Linux. Estamos hablando del Bourne Shell, más conocido como `bash`, que suele ser el intérprete de comandos instalado por defecto en nuestro sistema.

En general esto no desmerece en nada las posibilidades de otros intérpretes de comandos como pueden ser el `tsh` o el `ash`. Todo lo contrario, los intérpretes de comandos del mundo UNIX presentan una potencia sin igual, en especial si los comparamos con sus equivalentes en Microsoft® Windows® (o sea el viejo `COMMAND.COM` o el nuevo `CMD.EXE`). Esto es natural si tenemos en cuenta que por la consola de los sistemas UNIX han pasado millones de profesionales que han contribuido con sus comentarios o con su esfuerzo a que haya ido ganando en potencia a lo largo de los años. Cada usuario puede tener su propio intérprete de comando, pero por sencillez, y puesto que es el intérprete por defecto en muchos sistema GNU/Linux, no centraremos exclusivamente en el `bash`.

En realidad ya hemos pasado por un capítulo donde aprendimos los principios básicos del uso del intérprete de comandos, ahora se trata de utilizarlo para generar pequeños programas que nos ayuden en el trabajo diario. El `bash` no sólo permite la ejecución de las aplicaciones instaladas en el sistema; sino que proporciona una serie de comandos internos así como estructuras sintácticas de control de flujo semejantes a las existentes en muchos lenguajes de programación (p.ej: `for`, `case`, `while`, `until`).

9.1. Ficheros de comandos

Todas las características que veremos pueden ser utilizados interactivamente introduciendo los comandos directamente desde la consola. Esto es práctico para realizar tareas sencillas. Sin embargo, para desarrollar programas extensos o rutinas ampliamente utilizadas suele ser más interesante escribir nuestro programa en un archivo a modo de *script*. En este último caso podemos invocar nuestro programa escribiendo el comando:

```
$ bash mi_programa
```

De esa manera se iniciará la ejecución de una nueva copia del `bash` que abrirá el script y lo ejecutará. En los sistemas Linux suele haber un comando llamado `sh`. Dicho comando suele corresponderse con el intérprete de comandos por defecto de nuestro sistema. Por lo tanto, podemos sustituir `bash` por `sh` si estamos seguros de que el `bash` es nuestro intérprete por defecto o de que nuestro script utiliza características estándar entre los diferentes intérpretes disponibles. Si queremos garantizar que la interpretación de nuestro script la realice el `bash` es conveniente indicarlo explícitamente en lugar de utilizar el `sh`. Resumiendo, podemos invocar nuestro programa de la siguiente manera:

```
$ sh mi_programa
```

La verdad es que resulta mucho más profesional y sencillo que nuestro script se ejecute de forma semejante a la de cualquier otro programa. Es decir, escribiendo directamente su nombre en el intérprete de comandos. Para ello sólo es necesario que la primera línea de nuestro script sea así:

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

O sustituimos `bash` por `sh` si se dan las condiciones comentadas anteriormente. El último paso es habilitar los permisos de ejecución y ya podemos utilizar nuestro script.

```
$ chmod u+x mi_programa
$ ./mi_programa
```

Cada línea de nuestro script debe contener un comando a ejecutar por el intérprete. Si deseamos poner varios comandos en una misma línea debemos usar `;` para separarlos. Por lo tanto la siguiente secuencia de comandos:

```
whoami
pwd
date
```

Es equivalente a:

```
whoami; pwd; date
```

A la hora de mostrar texto por pantalla se utiliza el comando `echo`. Veamos el siguiente script:

Fichero `whoami.sh`

```
#!/bin/sh
# whoami.sh .- Ejemplo sencillo de programación en BASH

echo Usted es:
whoami
echo Su directorio actual es:
pwd
echo La fecha de hoy:
date
```

Ejemplo 9.1: Este código se limita a mostrar la información referente al usuario en la máquina local, el directorio de trabajo y la fecha actual del sistema.

Varios son los elementos nuevos que podemos ver en este programa, aparte del uso del `echo`:

1. Cuando el intérprete encuentra un `#` ignora todo el contenido de la línea a partir de ese punto.
2. El comando `who` muestra información sobre los usuarios autentificados en el sistema. Al añadir las opciones `am I` estamos indicando que queremos que sólo muestre información referente a nosotros como usuarios.
3. El comando `pwd` informa del directorio actual de trabajo.
4. El comando `date` muestra la fecha y hora actual del sistema.

Existen una serie de variables que el propio intérprete define en el momento de ejecutar el script y que contienen información sobre la línea de comandos que se le ha pasado. Una breve descripción de esas variables la podemos ver en el Cuadro 9.1. Veamos un ejemplo sencillo:

Variable	Contenido
`$0`	Nombre del fichero de comandos
`$1-$9`	Argumentos del 1º al 9º. El primero es $1, el segundo $2, etc.
`$*`	Línea de comandos completa exceptuando $0

Tabla 9.1: Variables de la línea de comandos

Fichero `yorecuerdo.sh`

```
#!/bin/sh
# yorecuerdo.sh .- Argumentos de la línea de comandos.

echo El comando es $0
echo El primer argumento es $1
echo El tercer argumento es $3
echo Todos los argumentos son $*
```

Ejemplo 9.2: Ejemplo de acceso a los argumentos de la línea de comandos especificados al iniciar la ejecución de un script.

Al ejecutarlo obtenemos:

```
$ ./yorecuerdo A B C
El comando es ./yorecuerdo
El primer argumento es A
El tercer argumento es C
Todos los argumentos son A B C
```

9.2. Variables de entorno

El intérprete de comandos permite la definición de variables que puedan ser utilizadas en nuestros script. Algunas de esas variables tienen un significado particular en nuestro sistema. Para conocer las variables actualmente definidas podemos utilizar el comando `set`. Si lo usásemos seguramente veríamos variables como `$HOME`, que define nuestro directorio personal de usuario (la ruta ∼/ tiene el mismo significado), o `$PATH`, que contiene la lista de directorios donde el intérprete buscará los ejecutables cuando le indiquemos el nombre de alguno.

Definir nuevas variables es tan sencillo como realizar una asignación:

```
$ EDAD=24;
```

Mientras que acceder a su valor se hace precediendo al nombre de la variable con el caracter `$`.

```
$ echo Mi edad es $EDAD
```

Toda variable definida en un `bash` es local a esa copia del `bash` y por tanto invisible para el resto de programa. Si tenemos en cuenta que cada vez que ejecutamos un script se inicia un `bash` nuevo, resulta evidente que todas las variables definidas en el script serán destruidas cuando éste termine. Además si un script ejecuta otro script, uno no podrá ver las variables del otro. Para que esté garantizado que una variable pueda ser vista fuera del `bash` donde fue definida es necesario *exportarla*.

```
$ EDAD=65
$ export EDAD
```

A continuación vamos a añadir una nueva ruta al `$PATH`:

```
$ PATH=$HOME/bin:$PATH
```

O a modificar nuestro prompt:

```
$ PS1='Le obedezco amo: '
```

Como se puede apreciar hemos puesto la frase entre comillas. Si no lo hubiéramos hecho así obtendríamos un error a causa de los espacios. Esto nos lleva a intentar conocer algunos caracteres que en el entono del `bash` tienen un significado especial.

9.3. Metacaracteres

A esos caracteres se los de nomina metacaracteres:

9.3.1. Sustitución: * ?

El primero puede ser sustituido por un número indeterminado de cualquier combinación de caracteres. El segundo sólo representa a *un* carácter cualquiera. Se explican por sí mismos si vemos el siguiente código de ejemplo en el que se listan todos los archivos del directorio actual:

```
$ echo Introduzca un *
```

O todos los que empiecen por `a` y terminen por `b`:

```
$ echo Introduzca un a*b
```

O sencillamente todos los que empiecen por `a` y terminen por `b` pero que sólo tengan tres letras:

```
$ echo Introduzca un a?b
```

9.3.2. Redirección: > >> < |

Podemos volcar la salida de un comando a un archivo, borrándolo y creándolo de nuevo si éste existe.

```
$ ls > lista_archivos
```

O bien si preferimos añadir la salida del comando a un fichero ya existente:

```
$ ls -al >> lista_archivos
```

O pasar dicha salida como entrada a otro comando:

```
$ ls | less
```

O usar como entrada de un comando el contenido de un archivo:

```
$ less < lista_archivos
```

9.3.3. Ejecutar en segundo plano: &

Si al final de un comando añadimos `&`, éste se ejecutará en segundo plano. Es decir, el `bash` no esperará a que el comando termine, permitiéndonos seguir ejecutando nuevos programas mientras este se ejecuta en paralelo. Evidentemente resulta muy práctico cuando suponemos que un comando va a llevar un tiempo de de ejecución prolongado y nosotros deseamos poder seguir utilizando el sistema.

9.3.4. Separación de comando: ;

Como ya hemos visto, permite indicar varios comandos en una misma línea.

9.3.5. Continuación de línea: \

Permite dividir una línea en varias si por algún motivo no podemos escribirla de una sola vez.

```
$ echo Quiero vivir en \
> canarias
```

9.3.6. Valor de una variable: $

Como ya hemos visto, debe preceder al nombre de una variable para que el intérprete sepa que debe sustituir su valor.

9.3.7. Otros: ( ) [ ] ‘

Los corchetes se utilizan en la evaluación de expresiones condicionales y los paréntesis en la evaluación de expresiones artiméticas. Veremos unos ejemplos más adelante.

En cuanto al último metacarácter, al ejecutar un comando entre las comillas invertidas (tildes francesas) se sustituye la cadena por la salida de dicho comando. Por ejemplo, si ejecutamos:

```
$ echo date
date
```

Pero si ejecutamos:

```
$ echo ‘date‘
mar oct 30 23:51:08 WET 2001
```

Si deseamos escribir estos metacaracteres de forma literal deben ir precedidos de \.

```
$ echo \* \\ \[
* \ [
```

También podemos evitar la sustitución si los escribimos entre comillas simples o dobles:

```
$ echo "Enviar $100?"
Enviar $100?

$ echo "‘minombre‘ es dulce"
‘minombre‘ es dulce
```

La diferencia entre las comillas simples y las dobles es que ambas eliminan el significado de todos lo metacaracteres con la excepción, sólo en el caso de las comillas dobles, de $ y ‘.

9.4. Ficheros de comandos interactivos

Aparte de ejecutar una secuencia predeterminada de comandos nuestros scripts pueden ser interactivos. Es decir solicitar y leer datos desde la consola de usuario. Para ello se utiliza el comando `read` seguido por uno o más nombres de variable.

```
$ read NAME1 NAME2 NAME3
$ echo $NAME1
$ echo $NAME2
$ echo $NAME3
```

El comando `read` lee desde la entrada estándar hasta encontrar un espacio y almacena lo leído en la primera variable indicada. A continuación sigue leyendo hasta el siguiente espacio y almacena la nueva lectura en la segunda variable. Así sucesivamente, excepto que la última variable siempre almacena desde el último espacio alcanzado con variable anterior hasta el final de la línea.

9.5. Control de flujo del programa

Al igual que en muchos otros lenguajes de programación, disponemos de sentencias de control de flujo del programa. Algunos ejemplos son:

if `if condicion1; then`
`elif condicion2; then comando;`
`[ ... ]`
`else comando;`
`fi`

while `while condicion; do comando; done`

until `until condicion; do comando; done`

Existen múltiples alternativas a la hora de establecer una condición para el control del flujo. Una de ellas es la verificación del código de error devuelto por un comando o programa al final su ejecución. Por definición el código de error de un comando que ha terminado de forma satisfactoria es 0, lo cual es considerado por el intérprete como cumplimiento de la condición. En caso contrario el comando devolverá un código distinto de 0 y el intérprete considerará que la condición no se cumple. Para conocer en qué situaciones se devuelven los diferentes códigos de error se hace necesario consultar la ayuda de cada comando en particular. Por ejemplo en el siguiente código se utiliza el programa `grep` para buscar un nombre de usuario en el archivo de contraseñas del sistema (`/etc/passwd`).

Fichero `finduser.sh`

```
#!/bin/sh
# finduser.sh .- Utilización de la sentencia "if".
# Copiado de http://www.linuxgazette.com/issue25/dearman.html

# Carga la variable $ME con el primer argumento de la línea de comandos.
ME=$1

# Buscar a $ME en el archivo de contraseñas del sistema.
if grep $ME /etc/passwd > /dev/null
then
  # Si $ME está en el archivo, mostrar la siguiente línea
  echo "$ME es usuario de este sistema"
fi
```

Ejemplo 9.3: Ejemplo del uso de la sentencia `if` para el control del flujo de un programa diseñado para buscar un nombre de usuario en el archivo de contraseñas del sistema.

Si `grep` encuentra al usuario en `/etc/passwd` saldrá de forma correcta, por lo que se cumple la condición y se ejecuta el comando que está dentro del cuerpo del `if`.

Otras sentencias son `case`, `select` y `for`. Una forma interesante de esta última es:

`for variable in lista; do comando; done`

Por ejemplo:

```
$ for a in pato gallo perro
> do
>   echo yo tenía un $a
> done
```

Veamos algunos casos más interesantes. Para ello vamos a empezar por el siguiente programa:

Fichero `minusculas1.sh`

```
#!/bin/sh
# minusculas1.sh .- Utilización de la sentencia "for".
# Copiado de http://www.linuxgazette.com/issue25/dearman.html

DIR=$1

for a in `ls $DIR`
do
  fname=`echo $a | tr A-Z a-z`
  mv $DIR/$a $DIR/$fname
done;
exit 0
```

Ejemplo 9.4: Ejemplo del uso de la sentencia `for` para el control del flujo de un programa diseñado para renombrar todos los archivos de un directorio. Esto se hace de forma que se sustituyen los caracteres en mayúsculas por sus equivalentes en minúsculas.

Resulta sencillo apreciar que debemos pasar como primer argumento de nuestro programa un nombre de directorio. Ese nombre se almacena en `$DIR` para su uso posterior. En la sentencia del `for` se ejecuta el comando `ls $DIR` que listará el contenido del citado directorio. Al rodear el comando por comillas invertidas la salida del mismo no va a la consola sino que se almacena para su uso por el `for`. Éste toma esa salida como una lista de elementos por los que tiene que pasar la variable `$a` en cada iteración. Por tanto, en cada interación `$a` contedrán el nombre de uno de los archivos del directorio `$DIR`.

Dentro del bucle se usa `echo` para que envíe el valor de `$a` a la salida estándar. Pero como usamos el metacarácter | en realidad pasa a la entrada del siguiente comando, es decir a `tr`. Éste realiza una traducción de dicha entrada sustituyendo los caractéres en mayúscula por los correspondientes en minúscula (o sea de `A-Z` a `a-z`) y envía el resultado a la salida. Nuevamente el uso de las comillas invertidas provoca que la salida se almacene en la variable `$fname`. A continuación se utiliza el comando `mv` para renombrar el archivo `$a` a `$fname`. Al finalizar el bucle habremos renombrado todos los archivos de `$DIR` pasando los caracteres de mayúsculas a minúsculas.

El programa finaliza con `exit 0`. Ese comando no es necesario para finalizar un programa, pues éste termina cuando se llega al final del archivo, pero resulta interesante para fijar el código de error a la salida. En este caso se indica que la ejecución fue sin errores, pues valores distintos de 0 se suelen usar para indicar condiciones de error. Si no se utiliza `exit` el programa terminará con el código de error devuelto por el último comando ejecutado dentro del programa.

El script anterior no funcionará si no pasamos en la línea de comandos un nombre de directorio válido. Para comprobar el no cumplimiento de una expresión podemos utilizar `!`. Por ejemplo, vamos a añadir a nuestro programa la comprobación de que `$1` no esté vacío:

Fichero `minusculas2.sh`

```
#!/bin/sh
# minusculas2.sh .- Mejora del programa "minusculas1".
# Copiado de http://www.linuxgazette.com/issue25/dearman.html

if [ ! $1 ]
then
  echo "Uso: `basename $0` nombre_directorio"
  exit 1
fi

DIR=$1

for a in `ls $DIR`
do
  fname=`echo $a | tr A-Z a-z`
  mv $DIR/$a $DIR/$fname
done;
exit 0
```

Ejemplo 9.5: Mejora del programa `minusculas1` para que se compruebe que se haya pasado un nombre de directorio como argumento de la línea de comandos.

Para aquéllos que conozcan C/C++ todo esto resultará extremadamente sencillo. Hemos utilizado los corchetes para encerrar la expresión que debe ser evaluada de forma condicional. Al añadir ! indicamos que queremos comprobar el no cumplimiento de dicha expresión.

Operador	Comprobación
`-b fichero`	Comprueba si *fichero* es un dispositivo orientado a bloques
`-c fichero`	Comprueba si *fichero* es un dispositivo orientado a caracteres
`-d fichero`	Comprueba si *fichero* es un directorio
`-f fichero`	Comprueba si *fichero* es un fichero ordinario
`-r fichero`	Comprueba si *fichero* es legible por el proceso
`-w fichero`	Comprueba si *fichero* es escribible por el proceso
`-x fichero`	Comprueba si *fichero* es ejecutable

Tabla 9.2: Operadores de comprobación de `[ ]`

Aun así no somos capaces de comprobar que se haya pasado un nombre de directorio válido a nuestro programa. Para ello se utilizan una serie de operadores junto con los `[ ]`, de los cuales podemos ver los más empleados en el Cuadro 9.2.

Fichero `minusculas3.sh`

```
#!/bin/sh
# minusculas3.sh .- Mejora del programa "minusculas3".
# Copiado de http://www.linuxgazette.com/issue25/dearman.html

if [ ! $1 ]
then
  echo "Uso: `basename $0` nombre_directorio"
  exit 1
fi

if [ -d $1 ]
then
  DIR="/$1"
fi

if [ -f $1 ]
then
  DIR=""
fi

for a in `ls $DIR`
do
  fname=`echo $a | tr A-Z a-z`
  mv $DIR$a $DIR$fname
done;
exit 0
```

Ejemplo 9.6: Mejora del programa `minusculas2` para que compruebe que el argumento pasado sea un nombre de directorio válido.

Puesto que las modificaciones se explican por sí mismas no estaremos más en ellas. Un característica interesante de nuestro programa sería que comprobara la existencia del archivo de destino antes de ejecutar `mv`. En el estado actual si el archivo de destino ya existe el programa muestra un error. Esta modificación es sencilla a partir de nuestros conocimientos actuales.

Otra característica sería que admitiera poder pasarle más de un directorio a través de la línea de comandos. Este problema es ligeramente más complejo puesto que necesitamos conocer cuántos argumentos ha indicado el usuario. Para averiguarlo debemos usar una variable especial del intérprete, que éste prepara para nosotros. Se trata de `$#` que contiene el número de argumentos presentes en la línea de comandos. Conocido el número de argumentos debemos iterar sobre ellos utilizando la

Operador	Acción
`int1 -eq int2`	Verdadero si *int1* es igual a *int2*
`int1 -ge int2`	Verdadero si *int1* es mayor o igual a *int2*
`int1 -gt int2`	Verdadero si *int1* es mayor que *int2*
`int1 -le int2`	Verdadero si *int1* es menor o igual a *int2*
`int1 -lt int2`	Verdadero si *int1* es menor que *int2*
`int1 -ne int2`	Verdadero si *int1* no es igual a *int2*

Tabla 9.3: Operadores de comparación de enteros

sentencia `while`. También necesitamos saber como comparar números enteros, debido a la necesidad de conocer el número de veces que hemos iterado en el bucle, para así determinar cuándo salir. En el Cuadro 9.3 podemos ver algunos operadores de comparación de enteros.

Fichero `minusculas4.sh`

```
#!/bin/sh
# minusculas4.sh .- Mejora del programa "minusculas3".
# Copiado de http://www.linuxgazette.com/issue25/dearman.html

if [ ! $1 ]
then
  echo "Uso: `basename $0` nombre_directorio"
  exit 1
fi

while [ $# -ne 0 ]
do
  if [ -d $1 ]
  then
    DIR="/$1"
  fi

  if [ -f $1 ]
  then
    DIR=""
  fi

  for a in `ls $DIR`
  do
    fname=`echo $a | tr A-Z a-z`
    if [ $fname != $a ]
    then
      mv $DIR$a $DIR$fname
    fi
  done;

  shift
done
exit 0
```

Ejemplo 9.7: Mejora del programa `minusculas3` para que admita el paso de más de un directorio como argumento de la línea de comandos.

Quizás el único aspecto difícil de comprender es el uso del comando `shift`. La forma general de dicho comando es `shift [n]` donde, si no se especifica, `n` vale 1. Este comando desplaza los argumentos de forma que el `n+1` estará en `$1`, el `n+2` en `$2` y así sucesivamente. Los argumentos anteriormente en `$1`, `$2`, etc se pierden. Sabiendo eso comprender el programa anterior es realmente sencillo.

9.6. Funciones

El `bash` también permite crear funciones. En este caso el valor de retorno de la función será el valor de retorno del último comando ejecutado:

Fichero `funcsh.sh`

```
#!/bin/sh
# funcsh.sh .- Ejemplo del uso de funciones.
function mifunc ()
{
  echo La fecha de hoy:
  date
}

echo Veamos qué hace esta función...
mifunc
```

Ejemplo 9.8: Este programa define una función que muestra la fecha actual por la salida estándar, y realizar la llamada a dicha función.

En general el `bash` es un programa demasiado potente como para poder ser explicado en unas pocas líneas. Probablemente con lo que hemos comentado sea posible crear nuestros pequeños script para automatizar algunas tareas tediosas, pero en caso de querer profundizar más lo mejor es recurrir a la propia ayuda del intérprete (`man bash`). Un tema interesante a consulta es sobre el uso de los archivos `.bash_profile` y `.bashrc` de nuestro directorio personal, y sobre cómo usarlos para personalizar nuestro entorno de trabajo.

MÓDULO III

Edición de gráficos y documentos

TEMA 10

StarOffice

10.1. Introducción

StarOffice es un entorno de oficina que contiene todas las herramientas para desenvolverse en un ambiente de trabajo, así como también hogareño. Este set de herramientas tiene sus raices en StarOffice, un suite de oficina que desde mediados de los ochenta se desarrollaba en Alemania, a cargo de una compañía llamada StarDivision, cuyos derechos fueron comprados en 1999 por Sun Microsystems. La version 5.2 of StarOffice se lanzó en junio del 2000, y la versión 6.0 salió en mayo del 2002. Lo más extraordinario del desarrollo de StarOffice 6.0 es que se desarrolla mayoritariamente en el contexto del proyecto Open Source de StarOffice. A efectos de posibilitar dicho desarrollo, Sun hizo público el código fuente de la mayoría de los módulos de StarOffice (excepto por algunos módulos de proveedores externos, que no han entrado en el proyecto), y estableció el proyecto StarOffice. Esto no significa sin embargo que Sun deja en manos de voluntarios el desarrollo futuro del suite: la mayoría del trabajo todavía se lleva a cabo por desarrolladores de Sun. Sun también se encarga de los costes operacionales del proyecto StarOffice.

StarOffice es uno de los grandes competidores del paquete de oficina *Microsoft Office*, ya que el StarOffice es compatible con los formatos de archivos del MS-Office, además que existen versiones para GNU/Linux, MS-Windows, y otros sistemas operativos; y por último, una característica muy importante: es gratis.

StarOffice es una suite ofimática que está compuesta por diferentes módulos que se pueden cargar de forma independiente. Este capítulo no está orientado a ser un manual exhaustivo de StarOffice, sino más bien una introducción a este enorme paquete, donde se explicará el funcionamiento básico de los módulos mas importantes y se mencionará la existencia de los demás módulos.

En la actualidad Sun Microsystems ha liberado el código de StarOffice 5.2, lo que ha derivado en el desarrollo de OpenOffice.org (`http://www.openoffice.org`) mientras Sun Microsystems sigue vendiendo StarOffice 6.

10.2. StarOffice Writer: Procesador de textos

10.2.1. Introducción:

StarOffice Writer es un procesador de textos[1] avanzado, con el que se pueden crear diferentes tipos de documentos en los que se pueden insertar tablas, esquemas, gráficos, etc. Tambien cuenta con plantillas [2] que simplifican la creación de: cartas, folletos, artículos, informes, curriculum vitae, libros, etc. Además de las anteriores existen multitud de utilidades que facilitan el uso del procesador, como son el autocorrector, el autoformato, etc.

Un punto muy importante a favor de StarOffice Writer es la compatibilidad con los documentos guardados con formatos propietarios como Microsoft Word, no obstante siempre se podrán exportar documentos de cualquier procesador de texto guardandolo en formato de texto Enriquecido (RTF[3]). StarOffice Writer posee un sisema de ayuda que puede ser consultado de forma contextual (pulsando F1) mientras se edita el documento o buscando en los temas de ayuda. Se debe tener en

[1] Un procesador de textos es un programa con el que se generan diferentes documentos de texto como, cartas, informes o incluso el presente libro

[2] Una plantilla es el esqueleto de un modelo de documento en el que vamos insertando nuestro texto, esto se explica en el apartado 10.2.5

[3] RTF: formato de texto enriquecido, es un formato estandar que puede ser entendido por la practica totalidad de editores de texto.

cuenta que se puede acceder a todas las funcionalidades del editor a través de los diferentes menús tanto con el ratón como con diferentes combinaciones de teclas, denominadas teclas rápidas, que permitirán realizar cualquier tarea de una forma mucho más rápida y eficaz.

10.2.2. Como empezar:

Este es el aspecto que tiene la pantalla del StarOffice Writer, antes de comenzar a crear nuestro documento vamos a determinar para que sirven los principales menús que encontramos en la aplicación.

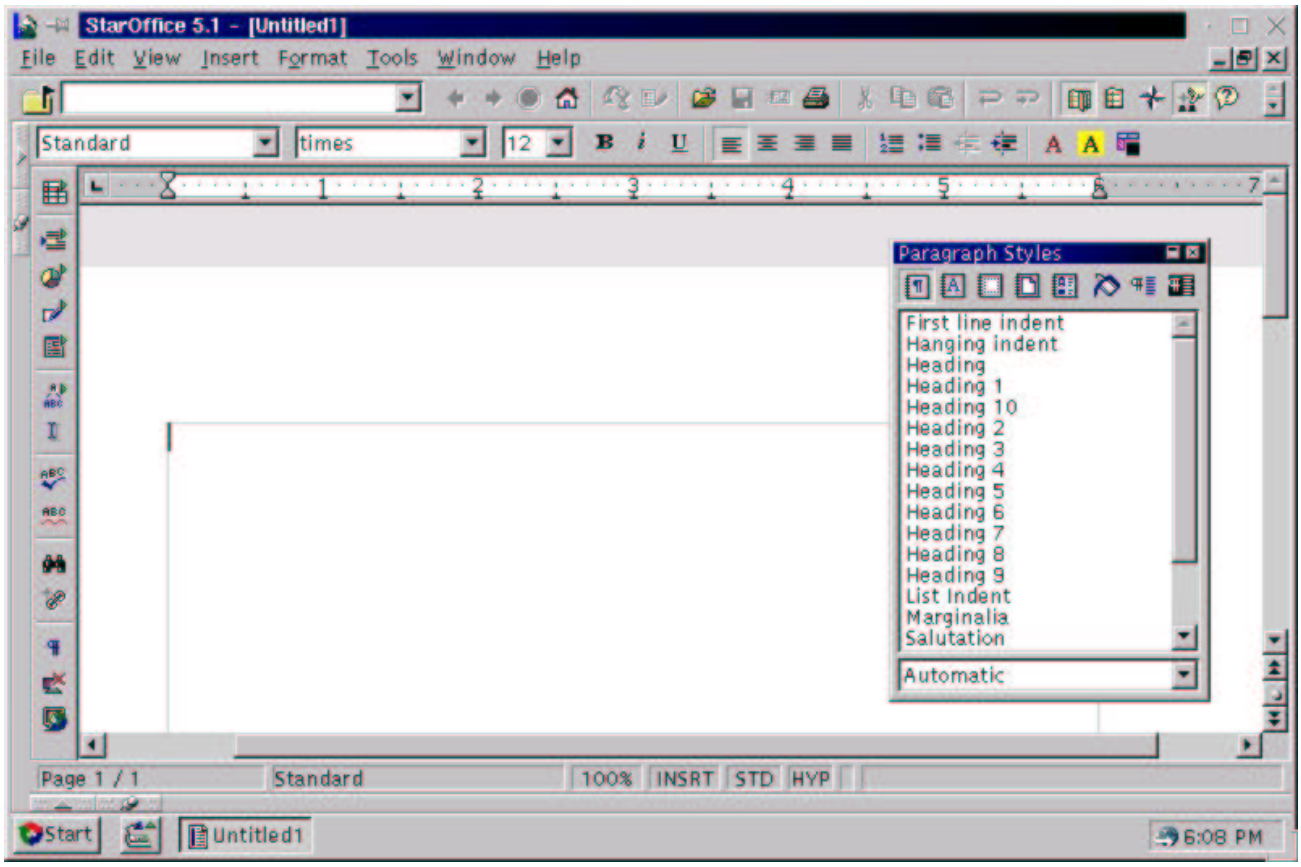


Figura 10.1: Primer vistazo al StarOffice Writer

10.2.2.1. Especificaciones del menú principal:

Archivo: Este menú se utiliza para abrir, cerrar, imprimir o crear asistentes para documentos, para cerrar StarOffice Writer hay pulsar en este menú la opción cerrar.

Editar: Sirve para modificar el texto, cortar, pegar, seleccionar, buscar y remplazar, etc.

Ver: Es posible ver el documento a distinta escala, tener acesso directo a los archivos del usuario a traves del beamer, ver la regla, la barra de estado, los submenús de las barras de herramientas.

Insertar: Con este menú se insertan campos, simbolos, encabezados, indices, notas, referencias, tablas, imagenes, objetos.

Formato: Modificación de los parrafos, caracteres, páginas, el estilo y aplicación del autoformato.

Herramientas: Aquí se encuentra el diccionario de sinonimos, la ortografia, la numeración de capitulos y páginas, actualizar el documento y las macros[4].

[4]Una macro es una utilidad que creamos para automatizar tareas repetitivas.

Ventana: Sirve para ir de una ventana a otra, es decir cuando se tiene abierto más de un documento se tiene la posilididad de desplazarse de uno a otro, verlos en cascada, azulejearlos que es verlos todos a la vez en la pantalla, ver dos documentos en horizontal o en vertical, cerrar todas la ventanas a la vez y ver cuantos documentos se tienen abiertos.

Ayuda: Para obtener ayuda sobre el contenido o sea sobre StarOffice Writer, tener el asistente abierto mientras escribimos, o la ayuda activa que al posicionar el cursor sobre un icono da una explicación de para que se utiliza (está sin traducir al castellano).

10.2.3. Uso de la ayuda:

10.2.4. El documento:

Una vez que se ha abierto StarOffice, para abrir una sesión de StarOffice Writer hay que abrir un documento ya sea nuevo o existente.

10.2.4.1. Como crear un nuevo documento:

Para crear un documento nuevo simplemente se hace doble click sobre el icono Nuevo Texto y aparecerá el nuevo documento. Tambien se puede abrir un nuevo documento seleccionando en el menú [`Archivo`] la opción [`Nuevo`] y elegir [`Texto`] .

10.2.4.2. Como abrir un documento existente:

Con StarOffice Writer existe la posibilidad de trabajar con documentos de formatos muy diferentes, como pueden ser los documentos de Microsoft Word, html, TXT, RTF, Winword, etc. Para abrir un documento existente se pulsa en [`Archivo`] y se elige [`Abrir`] y aparecerá un cuadro de dialogo en el que aparece la ubicación de los archivos en nuestro ordenador, se elige la ruta correcta y el nombre del archivo y se pulsa abrir o utilizamos las teclas rápidas `Ctrl-O`.

10.2.4.3. Guardar, guardar como:

La primera vez que se guarda un documento hay que darle un nombre y decidir su ubicación (donde se quiere guardar). Guardar como, se utiliza para guardar el documento con un nombre de archivo, por ejemplo: Manual.doc, y en una localización determinada en el sistema de archivos, por ejemplo: `/home/usuario/StarOffice Writer/personal/trabajos/Manual.doc`, para Guardar Como se pulsa sobre [`Archivo`] y después sobre [`Guardar como`], y se elige la ruta y el nombre del archivo en el cuadro de dialogo.

Guardar, se utiliza para conservar los cambios que se realizan en el documento mientras se escribe, para guardar se pulsa sobre [`Archivo`] y después sobre [`Guardar`] y todos los cambios serán guardados.

10.2.4.4. Edición y formateado de un documento:

Como se ha comentado anteriormente el texto puede formatearse mientras se escribe o una vez que se ha terminado la edición del documento. Si se quiere formatear el texto mientras se escribe se activa la orden, por ejemplo activar `negritas`, y una vez acabado se desactiva la opción. Para formatearlo una vez que terminado hay que seleccionar el texto o la parte del texto que queramos modificar, para señalarlo basta con colocar el puntero del ratón al comienzo del texto y pulsando el botón derecho arrastrar el ratón hasta el final de la selección, tambien se puede seleccionar mediante los cursores manteniendo la tecla de mayusculas pulsada.

Para ver el documento en una escala mayor o menor se utiliza....Para cambiar el tamaño y tipo de letra (una vez seleccionada) se pulsa `Formato`.....y aparecerá un cuadro de dialogo en el que se puede elegir el tamaño de la letra, el tipo de fuente, Negrita, cursiva, etc. Una vez seleccionados los cambios pulsamos `Aceptar` y se modificará el texto seleccionado. Para ver las marcas del texto de principio o fin de párrafo, tabulaciones, sangrías, etc., se pulsa en [`Ver`] y después se selecciona ..... (Caracteres no Imprimibles)

10.2.4.5. Trabajar con tablas:

Para insertar una tabla en un documento se selecciona [insertar] y se pulsa [tabla], aparecerá un cuadro de diálogo....

Para modificar el aspecto de la tabla en el documento se selecciona [Formato], [Tabla] y en el cuadro de diálogo se pulsa en la pestaña Tabla.....

Para cambiar el ancho de las columnas..... Para cambiar los bordes..... El grosor o estilo de las líneas.....color..... sombras.....Como introducir texto en las casillas....., formato del texto de la tabla .....

10.2.4.6. Imprimir un documento:

Para poder imprimir correctamente un documento se debe tener previamente configurada la impresora que se vaya a utilizar[5]. Antes de imprimir el documento es conveniente que veamos el aspecto que va ha tener una vez impreso, para previsualizarlo se pulsa en [Archivo] y después en [Vista preliminar], en la pantalla aparece una barra de herramientas donde existen una serie de botones con los que podremos por ejemplo ampliar el documento, mostrar varias páginas en pantalla, etc. Para ver el documento se puede pulsar el icono con forma de prismáticos.

Una vez visualizado el documento, para imprimir se selecciona [Archivo] y después [Imprimir], se puede pulsar el icono de la impresora, abriendose un cuadro de dialogo en el que se puede ver una serie de opciones de impresión, impresora, propiedades de la impresora, donde elegiremos el formato del papel, la orientación, el area de impresión, el número de copias, etc. También se puede utilizar la serie de teclas Ctrl-P

10.2.5. Uso de plantillas:

10.2.6. Teclas rapidas:

10.2.7. Salir de StarOffice Writer:

Para salir de StarOffice Writer, se pulsa Archivo y se elige Salir, pudiendose utilizar también las teclas rápidas Ctrl-Q. Si el documento no se ha guardado previamente aparecerá un cuadro de dialogo que preguntando si se quieren guardarlos cambios realizados en el documento, seleccionando Si los cambios serán almacenados y se cerrara la aplicación, si por el contrario elegimos No los cambios no son guardados y se perderá todas las modificaciones realizadas en el documento, finalmente si se pulsa Cancelar se retornará al documento.

10.3. StarCalc: Hoja de Cálculo

10.3.1. Introducción

Una hoja de cálculo[6] es una herramienta puesta al alcance del usuario para solucionar una cierta clase de problemas. Estos problemas van desde llevar el cálculo de la contabilidad del hogar hasta el ánalisis de datos estadísticos pasando por un sinfín de posibilidades. El uso de una hoja de cálculo queda supeditada a lo que el usuario quiera, pero, hay situaciones en las que se hace imprescindible el uso de una hoja de cálculo. Desde llevar la contabilidad de pequeñas empresas, en las que se requieran sumar grandes cantidades de cifras que la hoja de cálculo puede hacer automáticamente, utilizarlo para hacer regresiones estadísticas no excesivamente complejas, llevar el control de pedidos, de ventas, de .... Un largo etcétera cuyo principal punto en común es el de evitar cálculos manuales tediosos que puedan inducir a errores. Definido el problema al que se puede enfrentar el usuario ahora queda la elección del programa de hoja de cálculo. En el mercado hay una gran cantidad de paquetes que incorporan hojas de cálculo. Aquí se expondrá el uso de la StarCalc, la hoja de cálculo perteneciente al paquete ofimático StarOffice. Pero lo aquí explicado puede ser aplicado sin esfuerzo al uso de otras hojas de cálculo, como la Excel. El lector debe tener en cuenta que aquí no se explican todas las opciones y características que incorpora StaCalc, sólo se trata de una introducción al uso de la misma así como una serie de prácticas en las que se tratará de extender la parte teórica.

[5] La impresora se configura siendo Administrador y desde el escritorio de StarOffice (Desktop)
[6] También denominada Planilla de Cálculo

10.3.2. Primeros Pasos

10.3.2.1. Entorno de trabajo.

El entorno de trabajo de la StarCalc se hace familiar para aquel usuario que conozca el uso de las hojas de cálculo. Es un entorno totalmente gráfico e intuitivo desde el principio. En el que podemos observar una barra de menús donde se encuentran las operaciones que se pueden hacer con la StarCalc. Debajo se hallan una serie de botones que se usan para acceder más rápida y cómodamente a funciones y acciones que se encuentran en los menús.

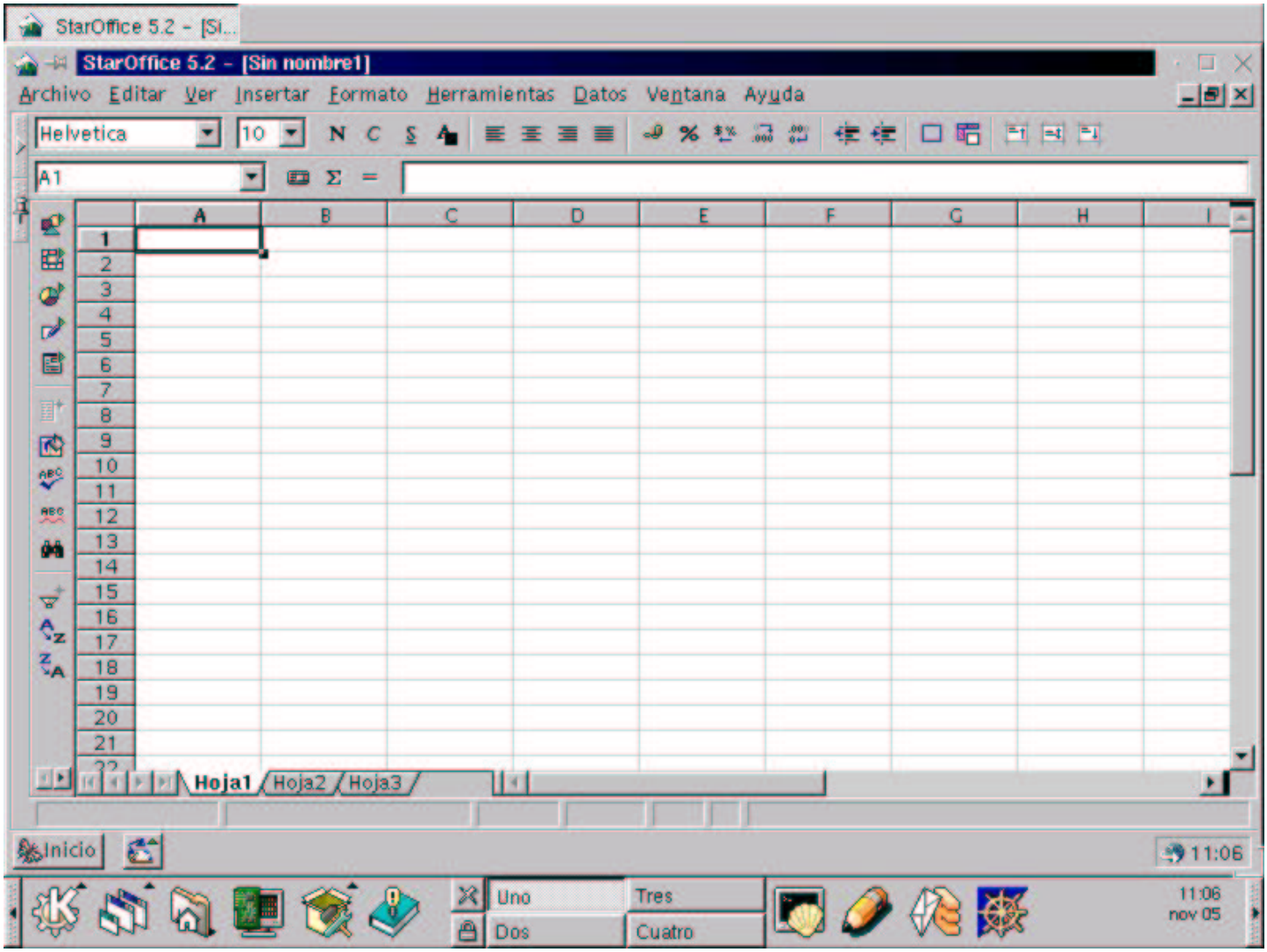


Figura 10.2: La hoja de cálculo StarCalc

La zona de menús, allí se posicionan el menú [`Archivo`], [`Editar`], [`Ver`], [`Insertar`], [`Formato`], [`Herramientas`], [`Datos`], [`Ventana`] y [`Ayuda`]. Dentro decada una de ellas se agrupan las diferentes macros que la StarCalc tiene predefinidas y que nos permiten realizar las tareas propias de la hoja de cálculo. Ver figura 10.3.

Figura 10.3: Los menús de la StarCalc

La siguiente zona está conformada por una serie de botones que permiten el acceso rápido a funciones definidas. Este área también se extiende a la parte izquierda de la pantalla, que pueden verse en las figuras 10.4 y 10.5.

Figura 10.4: Barra de botones superiores

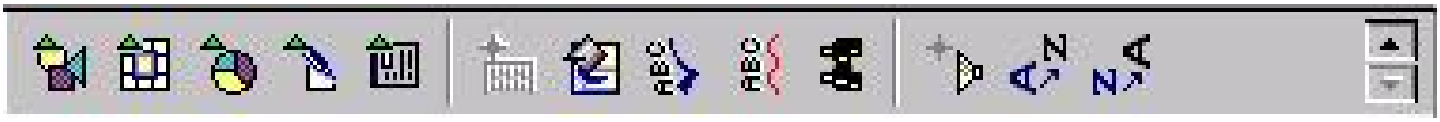


Figura 10.5: Barra de botones laterales

Luego se nos presenta la zona de trabajo, esto es, la parte destinada a la introducción, modificación, cálculo y presentación de datos, que puede observarse en la figura 10.6.

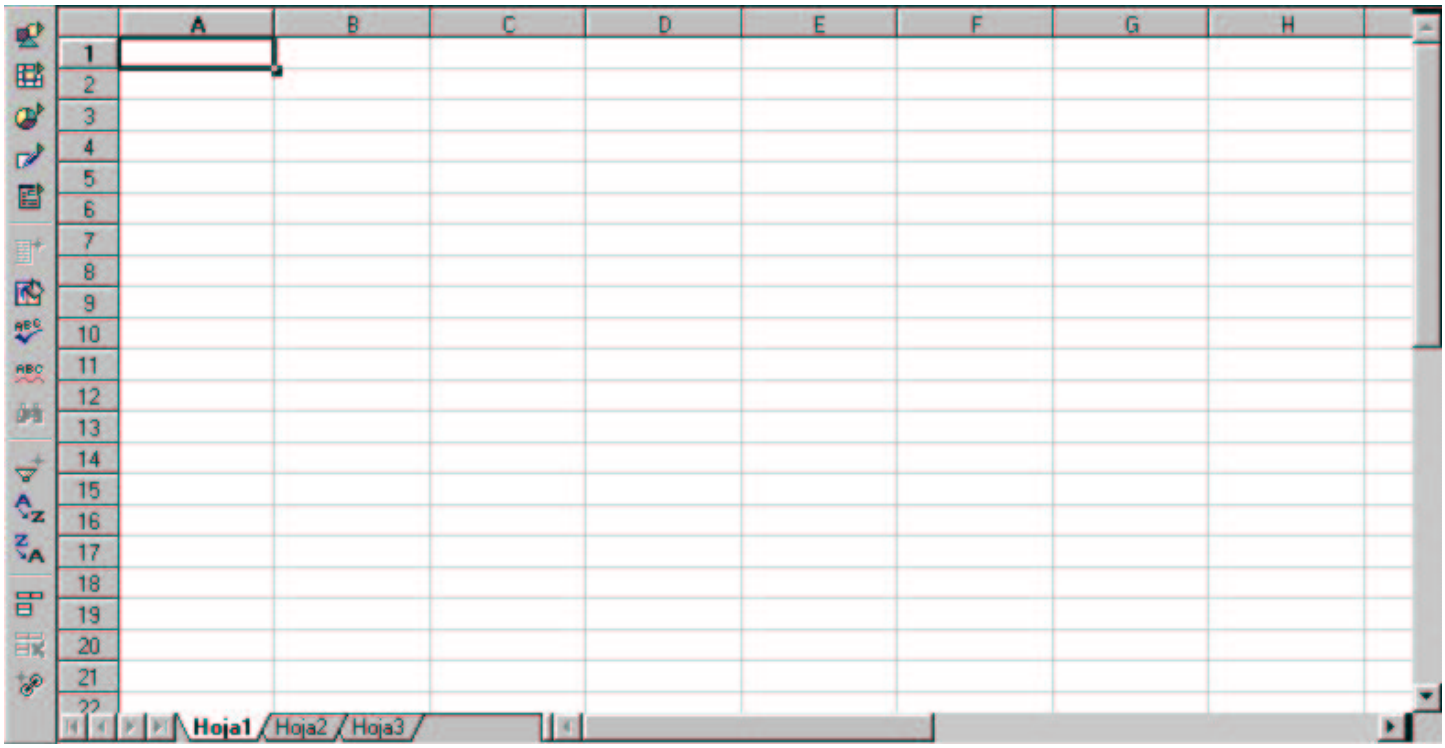


Figura 10.6: Zona de trabajo

La última zona está conformada por una línea en la que se presentan una serie de datos sobre características generales de la hoja en la que estemos trabajando.

10.3.2.2. Creación de nuevas hojas

Para crear una nueva hoja se puede recurrir a varias formas de hacerlo. Por un lado se puede hacer doble click en el icono de `Nueva Hoja de Cálculo` dentro del escritorio del entorno StarOffice. También se puede crear en el menú `Archivo`, submenú `Nuevo`, y elegir `Hoja de cálculo`. Otra forma es utilizar las plantillas, para ello se puede acceder al menú [`Archivo`], submenú [`Nuevo`], y [`Plantillas`], o también presionar `Ctrl+N`.

Selección de celda. Las celdas dentro de la hoja se pueden seleccionar bien usando el ratón o con el teclado. Con el ratón se presiona en la celda de inicio y, sin soltar, se extiende la selección moviendo el ratón hasta llegar a la celda final que se quiere seleccionar. Con el teclado, se posiciona con las teclas de dirección sobre la celda a seleccionar y se presiona la teclas

Mayúsculas. Sin soltar dicha tecla se seleccionan las celdas deseadas con las teclas de dirección.
Para seleccionar celdas que no sean contiguas se utilizará el ratón y la tecla Control. Se selecciona el primer rango de celdas o una sola, y a continuación se seleccionan las celdas o rangos deseados sin soltar la tecla Control. Para el caso en el que la selección de celdas quede fuera de las ventanas, hay que hacer notar que el movimiento es automático y no se necesita preocuparse hasta dónde llegar. Las columnas y filas se pueden selccionar enteras si se presiona sobre el rótulo de fila o columna.

10.3.2.3. Abrir hoja desde archivo

Para recuperar un trabajo desde el lugar donde esté almacenado acudiremos al menú [Archivo] y [Abrir] con lo que aparecerá el cuadro de opciones para recuperar el archivo, como se aprecia en la figura 10.7.

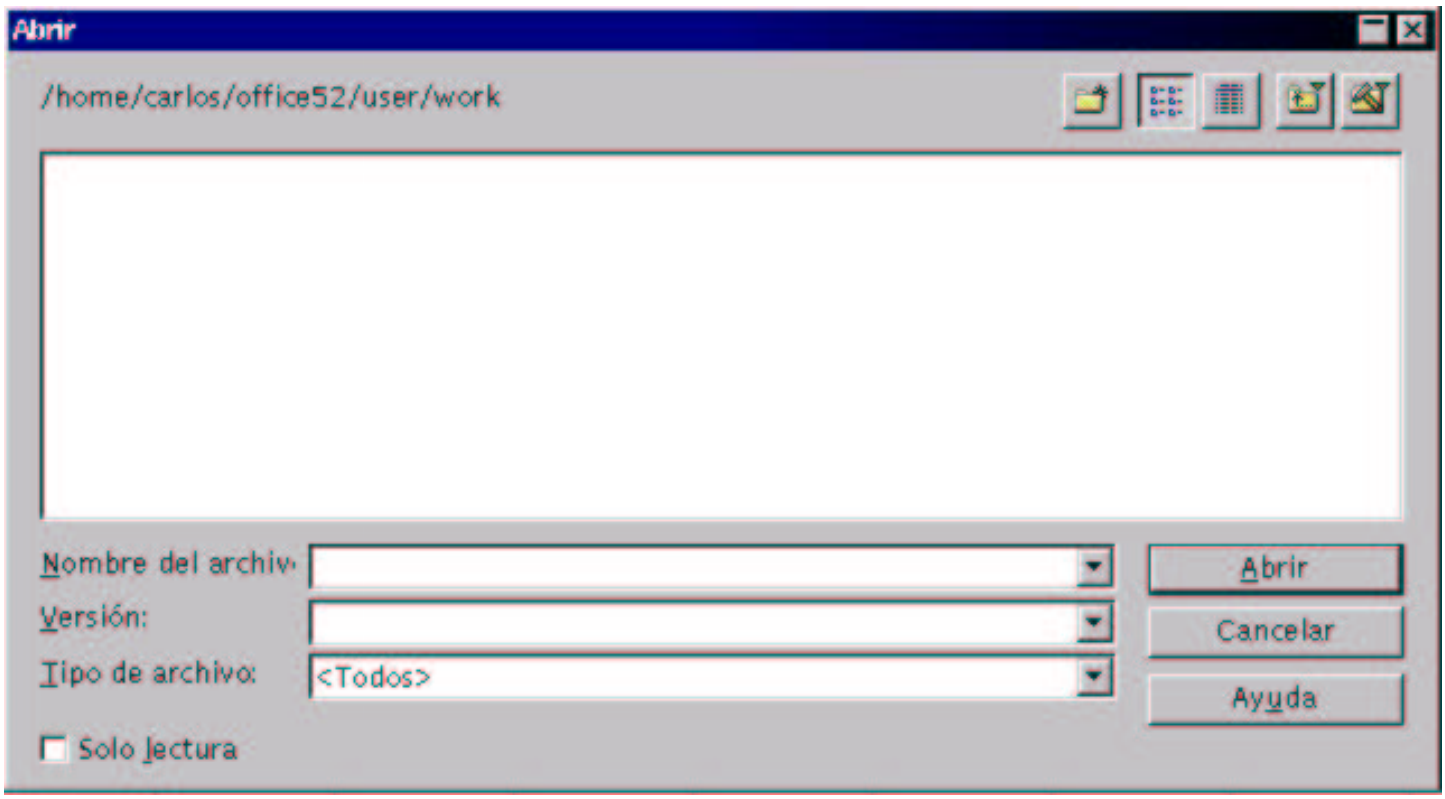


Figura 10.7: Recuperar archivo

Se puede acceder al cuadro de diálogo de abrir archivo presionando Ctrl+O.

10.3.2.4. Introducción de datos.

Se aceptan dos grandes tipos: constantes y fórmulas. Las constantes se dividen en: valores numéricos, de texto y de fecha y hora. Para los valores numéricos y de texto se aceptan: Números del 0 al 9 y ciertos carácteres especiales, tales como +-E e () . , $ % y /. Las restantes entradas se considerarán de texto. Si se quiere utilizar un valor numérico como texto se antepondrá el signo 'y luego se introduce el valor.

Para introducir datos nos situaremos en la celda adecuada presionando en ella y escribiremos aquello que queramos.

StarCalc puede ayudarnos a introducir datos, por ejemplo, en series de números podemos poner el primer número y seleccionar la celda. En la parte inferior derecha hay un punto negro, al seleccionarlo el puntero del ratón se convierte en una cruz, arrastrándolo hacia donde queramos poner los datos se extenderá la selección y la StarCalc sumará automáticamente los siguientes datos. En el caso de que los número no sean correlativos se pueden poner los dos primeros números de la serie y la StarCalc hará el resto.

Se puede hacer lo mismo con los nombres de los meses, si se introduce enero, y siguiendo el procedimiento anterior se conseguirá que vaya escribiendo todos los meses. La StarCalc muestra un rótulo al lado del puntero del ratón, de esta manera sabremos hasta dónde llegar sin tener que hacer cálculos mentales.

Protección de datos. Se pueden proteger libros enteros u hojas individuales, de manera que se permita acceso de sólo lectura o sin acceso. A través del menú [`Herramientas`] y [`Proteger`]. También se pueden proteger y desproteger celdas individuales.

10.3.2.5. Construcción de fórmulas.

Creación de fórmulas para la transformación de los datos.

Se selecciona una celda vacía y se introduce el signo igual precedido de la expresión aritmética a utilizar. Por ejemplo, para sumar 3 y 6 pondremos:

$$= 3 + 6$$

Precedencia de los operadores:

- Se procesan primero las expresiones entre paréntesis.
- Se ejecutan la multiplicación y la división antes de la suma y la resta.
- Los operadores del mismo nivel se calculan de izquierda a derecha.

Los paréntesis. Hay que incluir un paréntesis cerrado por cada uno que se abra.
Referencias a las celdas. Dentro de una fórmula nos podemos referir a una celda de diferentes maneras. Escribiendo A1 haremos referencia a la celda de la fila 1 columna A, que también puede ser seleccionada con el ratón. En este punto hay que diferenciar el tipo de referencia que se va a utilizar.
Referencias relativas. Se refieren a las celdas por sus posiciones en relacción a la celda que contiene la fórmula. De esta manera la celda A1 se convertirá en B2 si la fórmula se mueve una fila hacia debajo y una columna hacia la derecha.
Referencias absolutas. Se identifica a la celda por su posición fija, la fórmula hará referencia siempre a la misma celda aunque se mueva la fórmula. La manera de hacer esto es utilizando el signo $. Esto es, A1 hace referencia a la fila 1 columna A en todo momento. A$1 hará referncia a la fila 1 simpre y a la columna se calculará de forma relativa a la fórmula. El caso contrario sería $A que mantendría fija la columna. Nótese que se puede hacer referencia a cualquier celda dentro del libro de trabajo.

Uso de funciones. Las funciones tal y como las hemos visto pude que no tengan demasiado sentido si se trata, por ejemplo, de sumar una gran cantidad de celdas:

$$= A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8 + A9 + A10 + A11 + A12$$

por eso se puede utilizar la sintaxis que provee la StarCalc

$$= SUMA(A1 : A12)$$

Se puede utilizar cualquier función definida dentro de la StarCalc pasándole los parámetros adecuados. Sin embargo, no se pueden conocer o retener todas las funciones que tiene la hoja, por eso se puede utilizar insertar función que presentará un cuadro con las distintas funciones existentes, una pequeña explicación y los parámetros que se necesitan.

10.3.2.6. Formateo de la hoja.

Para poner el formato correcto de los datos podemos seleccionar las celdas a formatear y acudir al menú [`Formato`] [`Celda...`] y se nos presentará un cuadro con las diferentes opciones a configurar, desde el tipo de cuadro hasta el idioma, la fuente, el color, la justificación o la orientación de la escritura. Al finalizar la configuración presionamos `Aceptar`.

10.3.2.7. Impresión y presentación.

El uso de la impresión dentro de la StarCalc sigue el mismo esquema que en el resto de la suite StarOffice. Seleccionaremos el menú [`Archivo`] e [`Imprimir`], también podemos presionar las teclas `Ctrl+P`. En ambos casos se mostrará un cuadro de diálogo. En el caso de que no necesitemos ajustar las propiedades presionaremos `Aceptar` y StarCalc mandará el trabajo a la cola de impresión.

10.3.2.8. Salvando nuestro esfuerzo.

Para guardar los datos con los que estamos trabajando acudiremos a [`Archivo`] y [`Guardar`] o [`Guardar como`] si queremos guardarlo bajo otro nombre. En el caso de que el archivo sea nuevo se nos presentará el diálogo de guardar como para indicar a la StarCalc bajo que nombre y tipo de archivo deseamos que sea salvado nuestro trabajo.

10.3.3. Análisis de Datos

10.3.3.1. Funciones comunes.

Sintaxis de las funciones: Las funciones presentas dos partes diferenciadas, por un lado, el nombre de la función, por el otro, sus argumentos. Los argumentos de una función deben estar entre paréntesis inmediatamente después del nombre de la función y su cantidad dependerá del tipo de función que se use. También hay que destacar el hecho que dentro de los argumentos se pueden incluir tanto otras funciones, con sus propios argumentos, como valores numéricos, valores de texto o valores lógicos, así como matrices y valores de error. La forma más sencilla de insertar funciones es usándo el autopiloto de funciones que se encuentra en el menú [`Insertar`], [`Función`], o presionando las teclas `Ctrl+F2` o usando el botón que se encuentra en las barras de botones, dónde se introducen las fórmulas. Aquí no se van a exponer todas las funciones que existen e la StarCalc, para ver todas las funciones que la hoja puede ofrecer puede consultar el Autopiloto de Funciones, allí las encontrará agrupadas según el uso que se les quiera dar. Ver figura 10.8.

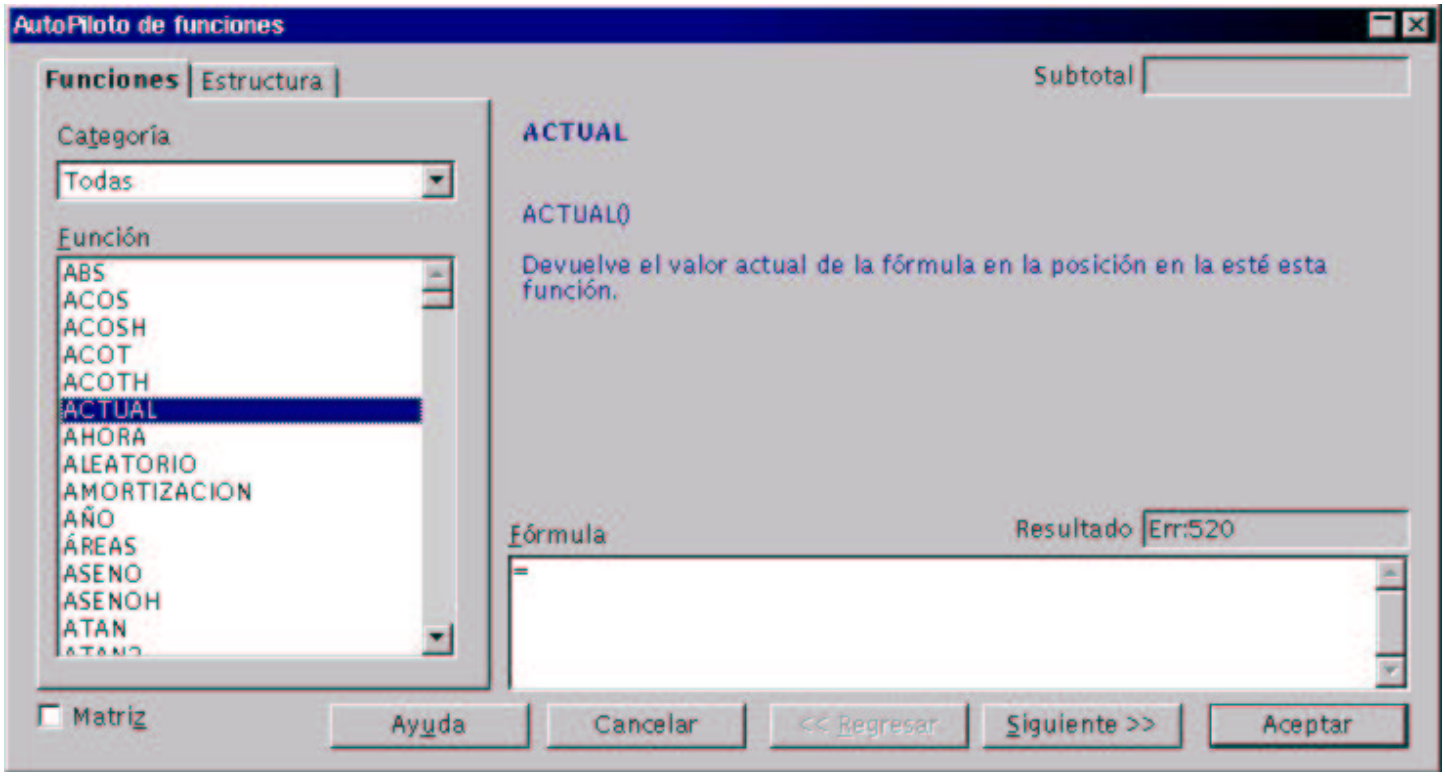


Figura 10.8: Autopiloto de funciones.

10.3.3.2. Fechas y horas.

Las fechas se introducen directamente en las celdas siguiendo alguno de los siguientes formatos que son perfectamente legibles por StarCalc: d/mm/aa; d-mmm-aa, d–mmm o mmm-aa. Para conseguir 5 de Noviembre el año 2000, teclearemos `5-nov-2000` y se nos mostrará tal cual la hemos introducido. En aquellos formatos en los que se omita alguna parte de la fecha se tratarán de distinta manera, así si falta el día se introduce automáticamente el día primero, si es el año se introduce el año en curso. Nótese que no puede faltar el mes en los formatos presentados. Para introducir series de fechas utilizaremos lo aprendido en la sección de introducción de datos, esto es, podemos extender la selección automáticamente con el ratón. Para la introducción de horas se pueden seguir los siguientes formatos: h:mm AM/PM; h:mm;ss AM/PM; h:mm; h:mm:ss; mm:ss.0; [h]:mm:ss. El lector interesado puede mirar el Autopiloto de funciones donde se encuentran algunas funciones para utilizar y transformar fechas.

10.3.3.3. Análisis financiero.

En una hoja de cálculo no podían faltar las funciones referidas a operaciones financieras, tales como el valor actual neto o la tasa interna de retorno. Estas funciones son muy utilizadas en entornos empresariales y para el usuario doméstico que de esta forma puede calcular el rendimiento de sus ahorros o si la inversión en la compra de una casa es factible. En esta sección también se debe hacer referencia a las funciones de depreciación, cálculo de la tasa de retorno y análisis de valores bursátiles. En este último caso, desarrollaremos un método de análisis de acciones y opciones en la parte teórica que no hace uso de las funciones que propone la StarCalc.

10.3.3.4. Análisis estadístico.

StarCalc utiliza funciones estadísticas predefinidas con las que puede calcular multitud de aspectos sobre poblaciones. Así tenemos funciones como media, modas, medianas, rango, histogramas, frecuencias covarianzas, coeficiente de correalción, desviación típica, distribuciones estándar, binomial, exponencial, normal, ji, poisson, regresiones lineales, entre otras. Cuya utilización será de utilidad para aquellas personas que requieran el uso de cálculos estadísticos.

10.3.4. Gráficos

Como dice el refrán, más vale una imagen que mil palabras. Por eso la StarCalc provee una serie de herramientas que permiten convertir los datos en gráficos para una mejor comprensión.

Creación de un nuevo gráfico. Lo primero es seleccionar los datos que se desean representar para luego ir a [`Diagrama`] en el menú [`Insertar`]. Se nos mostrará un cuadro pidiendo los datos, estos serán los que hemos seleccionado o cualquier rango que podemos ingresar manualmente. En este punto podremos seleccionar si queremos nuestro gráfico en la misma hoja o en una nueva hoja. Presionaremos `Avanzar` y se nos mostrará el cuadro de diálogo para seleccionar el tipo de gráfico que vamos a utilizar.

Una vez seleccionado el tipo de gráfico y la variante que vayamos a utilizar, se nos presenta un cuadro en el que podremos introducir los títulos que van a acompañar el gráfico.

Tras lo cual seleccionaremos `Crear` y se nos presentará el gráfico dentro de la hoja activa, o en una hoja nueva si es lo que hemos seleccionado previamente. Podremos trabajar directamente sobre el gráfico utilizando el botón derecho del ratón o el menú de formato que nos permitirá personalizar el gráfico.

Personalización de gráficos. Presionando con el botón derecho elegiremos la opción del menú [`Editar`]. De esta manera activaremos el gráfico y podremos utilizar las funciones de personalización sobre aquellas zonas del gráfico que deseemos cambiar. Para ello seleccionaremos la parte del gráfico a cambiar y seleccionaremos la opción [`Propiedades del Objeto`] que se nos muestra con el botón derecho del ratón o dentro del menú [`Formato`]. Ha de hacerce notar que se pueden utilizar la hilera de botones verticales que se encuentran en la parte izquierda de la pantalla y que nos permiten acceder a opciones del gráfico de forma más rápida.

10.3.5. Gestión de bases de datos y listas

La utilización de listas es muy común en las hojas de cálculo (clientes, teléfonos, etc.). Las lista deben cumplir una serie de características para que su utilización sea lo más efectiva posible.

- Cada columna debe contener el mismo tipo de información.

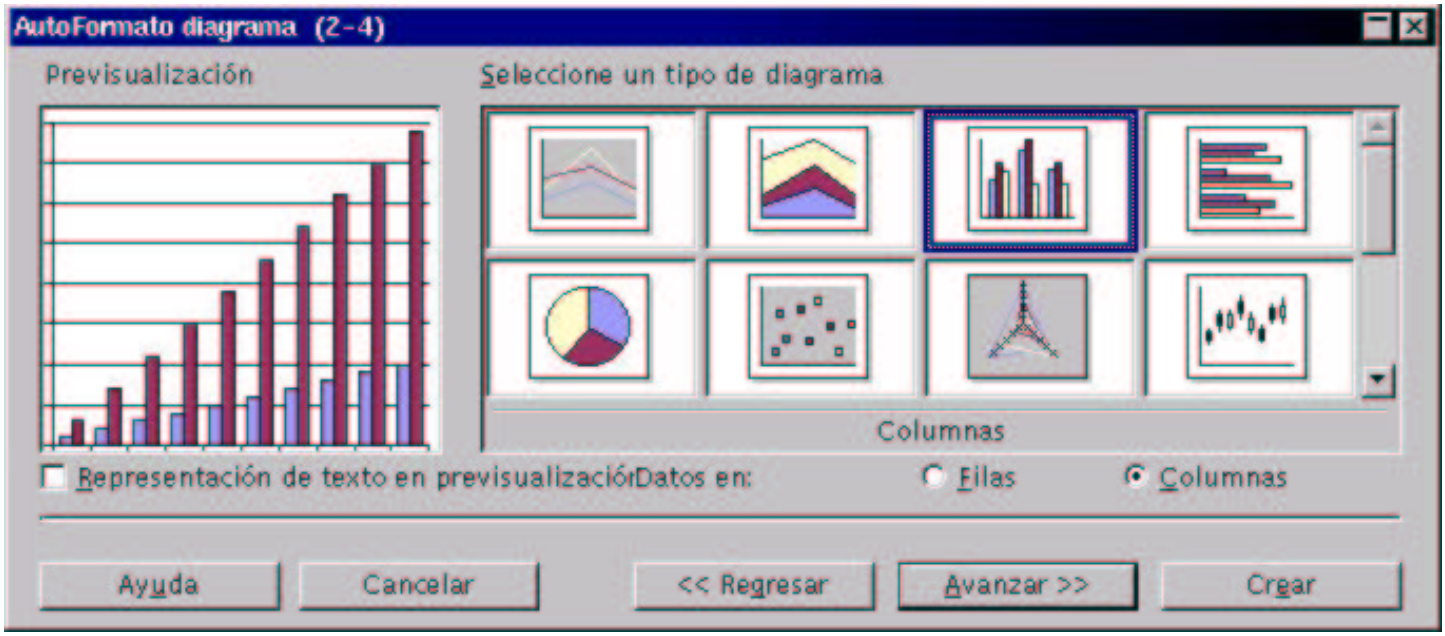


Figura 10.9: Selección del tipo de gráfico

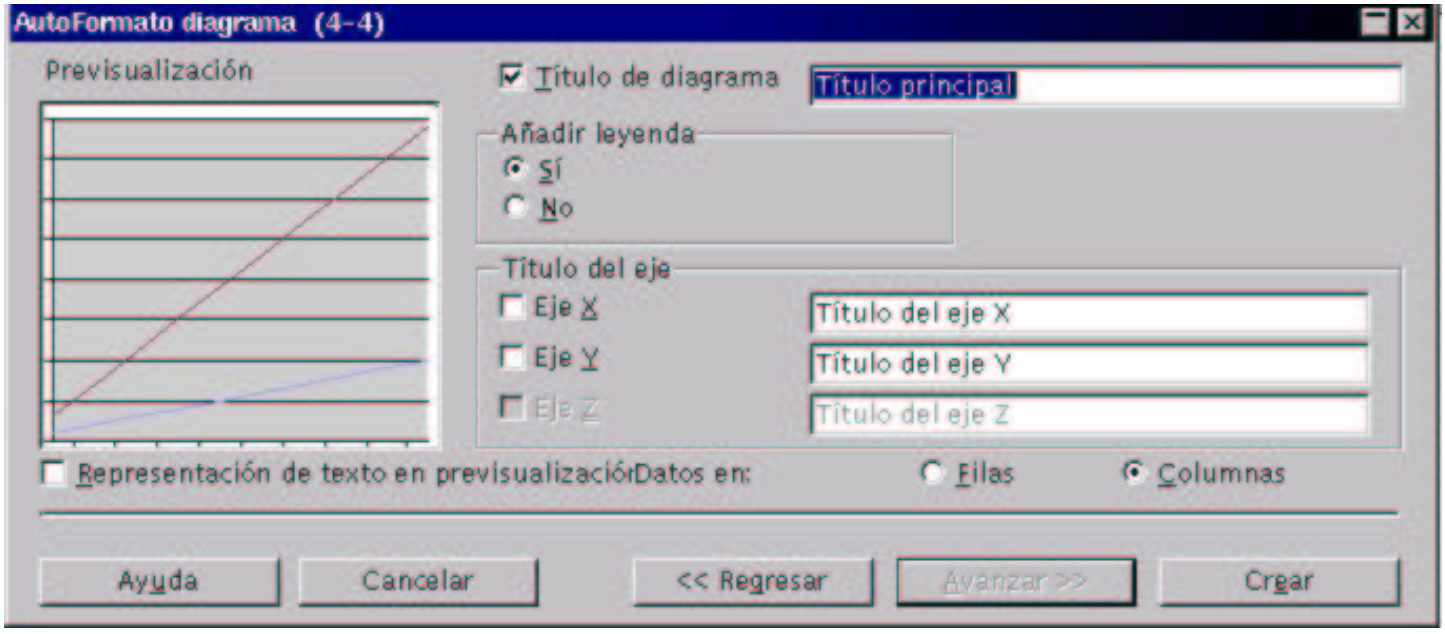


Figura 10.10: Titulos del gráfico

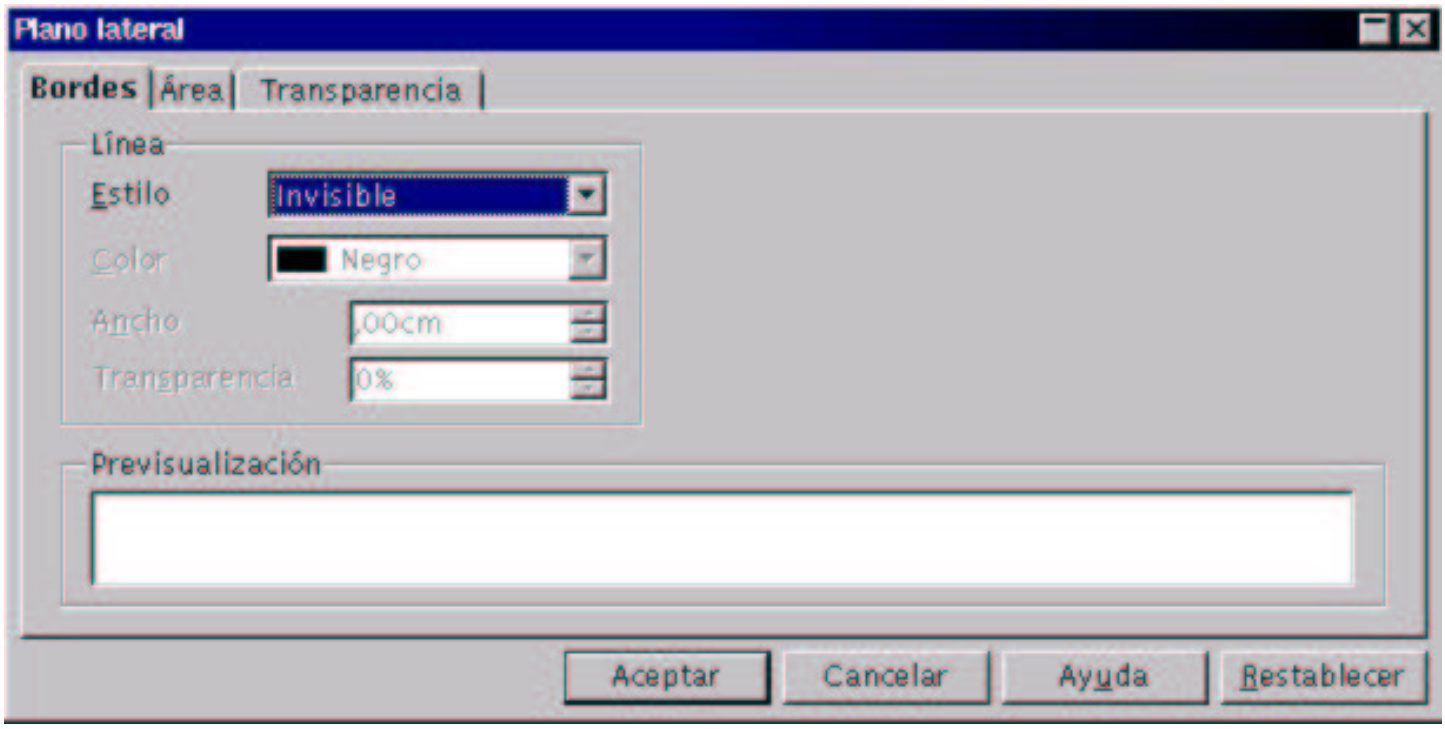


Figura 10.11: Personalización de gráficos

- La primera fila deben ser rótulos de descripción del contenido.
- No debería haber columnas o filas en blanco en la lista.
- Para el uso de filtros no debería haber otra información en las filas que ocupe la misma.

La utilización de listas es muy común en las hojas de cálculo (clientes, teléfonos, etc.). Las lista deben cumplir una serie de características para que su utilización sea lo más efectiva posible.

- Cada columna debe contener el mismo tipo de información.
- La primera fila deben ser rótulos de descripción del contenido.
- No debería haber columnas o filas en blanco en la lista.
- Para el uso de filtros no debería haber otra información en las filas que ocupe la misma.

Para crear una lista a partir de los datos de una hoja vamos al [`Piloto de Datos`] en el menú de [`Datos`]. Se nos mostrará el cuadro de seleccionar fuente.

Presionando aceptar se nos mostrará el cuadro de Pilot de Datos donde colocaremos por el método arrastar y soltar los datos que queramos que se muestren. En el botón de `Opciones` podremos acceder a una serie de características para afinar aún más la lista.

Ordenación de listas. Para ordenar listas sólo tenemos que seleccionar el área a ordenar y elegir [`Ordenar`] dentro del menú de [`Datos`] e indicar el método de ordenación y la columna por la que se ordenará, entre otras opciones de personalización. Se debe tener en cuenta que hemos de seleccionar la opción de Área contiene encabezamientos de columna para que no ordene los encabezados junto con los datos. Para esto seleccionaremos la solapa de Opciones dentro del cuadro `Ordenar` y marcaremos la opción `Área` contiene encabezamientos de columna, a continuación aceptaremos y se creará la lista.

Filtrado de listas. Con el uso de filtros dentro de las listas conseguiremos analizar aquellos datos que interesan. Por eso, al crear la lista se añade un botón denominado `Filtro` que permite definir los datos que se van a mostrar dependiendo de

aquellos datos que se busquen. *Bases de datos.* StarCalc puede trabajar conjuntamente con bases de datos profesionales. Se pueden importar tablas desde bases de datos que StarOffice tenga registradas para trabajar directamente con listas. En la versión 5.2 de StarOffice se instala por defecto el soporte para Adabas siempre y cuando dicha base de datos se encuentre en el sistema. El uso de bases de datos tiene su tratamiento en la sección de StarBase.

10.3.6. Tablas dinámicas

Es un tipo especial de tabla que resume la información de ciertos campos de una lista o base de datos. Las tablas dinámicas están vinculadas a los datos de los que proceden. Cuando dichos datos cambian, la tabla no se recalcula automáticamente, pero se puede actualizar en cualquier momento.
La creación de tablas dinámicas en StarCalc sigue el mismo procedimiento que el apartado anterior. Pero, esta vez, en lugar de seleccionar los datos en la hoja, utilizaremos fuentes externas. Activamos el [`Piloto de Datos`] en el menú [`Datos`]. Seleccionamos [`Fuente de datos registrada en StarOffice`][7] y presionamos `Aceptar`. Seleccionaremos aquellas fuentes que deseemos utilizar y persionaremos `Aceptar`. Los siguientes pasos son los del apartado anterior que el lector ya conoce.

10.3.7. Macros y Starbasic

Una macro es un conjunto de intrucciones que se ejecutan en conjunto y que permiten hacer tareas repetitivas y complejas. Para crear una macro dentro de StarBasic tenemos que acceder a [`Herramientas`] y [`Macro...`] se nos mostrá el cuadro Macro de la figura 10.12.

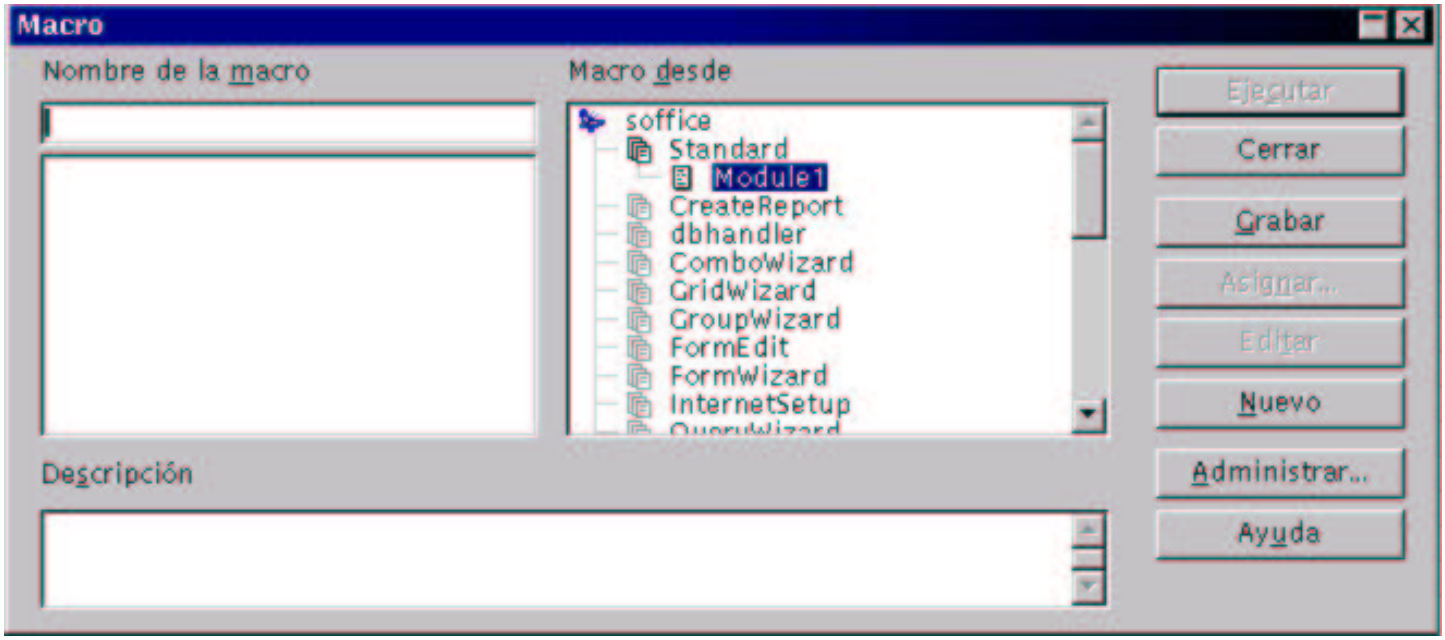


Figura 10.12: Cuadro de Macro

Lo primero será introducir el nombre de la macro a grabar y luego presionamos `Grabar`. Mientras estemos grabando la macro se almacenará cualquier acción que ejecutemos, tanto con el ratón como a través del teclado. Para finalizar presionaremos en el cuadro detener macro que se ve en la figura 10.13

Una vez grabada la macro podremos depurarla para ajustar algunos parámetros o para borrar aquellas partes en las que nos hayamos equivocado o queramos modificar. Las macros y sus modificaciones se hace utilizando el lenguaje que tiene la StarCalc y que se denomina StarBasic. Es un lenguaje muy intuitivo y que se parace al Visual Basic para Aplicaciones de la Excel.Figura 10.14.

Cuando la se tenga la macro finalizada, se puede asignar dicha a macro a una tecla o combinación de teclas. Para esto se accede al menú [`Herramientas`], [`Macro`]. Dentro del cuadro de Macro presionaremos sobre `Asignar...` y seleccionaremos la combinación de teclas que quereamos utilizar para activar la macro creada.

[7] En nuestro caso no podemos usar la Fuente externa/interfaz porque no tenemos registrada ninguna base de datos.

Figura 10.13: Parar grabación de la macro.

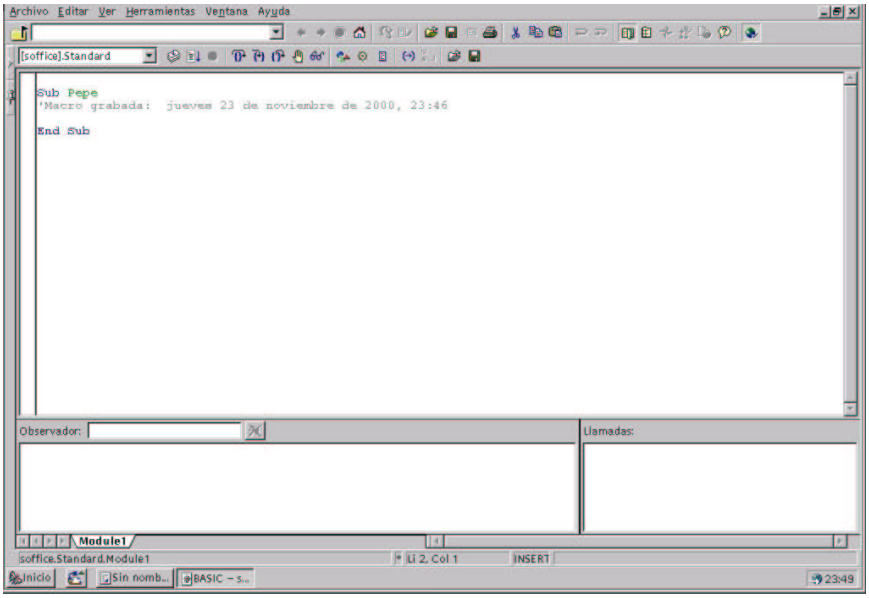


Figura 10.14: Entorno de StarBasic

10.3.8. De prácticas

En estas prácticas se pretenden enseñar rudimentos de la utilización de hojas de cálculo. Así como técnicas que no se enseñan en la parte teórica. Se presentan de menor a mayor dificultad. Exceptuando las dos últimas que son casos aparte.

10.3.8.1. Control de ingresos y gastos.

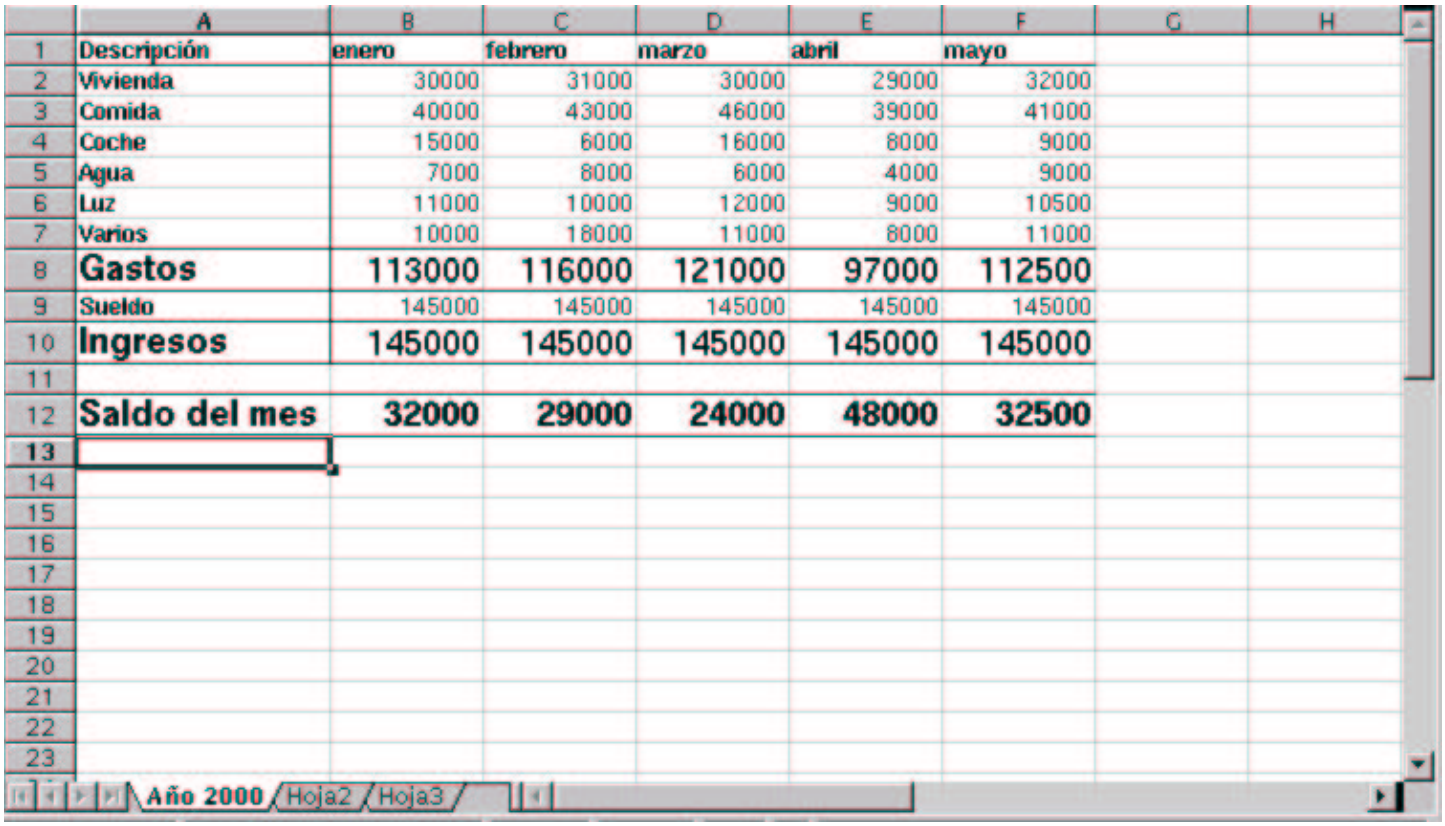
Objetivo. Llevar la contabilidad doméstica. Don Rufino Pensante no sabía por dónde se le escapa el dinero que ganaba con su trabajo en una tienda de caramelos así que decidió montar un sistema de contabilidad doméstica para llevar un orden y control de todo el dinero que pasaba por sus manos.

Lo primero de todo es diseñar la hoja para no tener que estar más tarde rediseñando todo el trabajo. Vemos de que información disponemos y lo que queremos conseguir. Esto es, tenemos series de datos correspondientes a ingresos y gastos y queremos ver el saldo al final de cada periodo. Lo más lógico es pensar en tener una periocidad mensual aunque no se descarta que sea semanal para aquellas personas, que, por ejemplo, tengan una nómina semanal.

El siguiente paso lógico es crear un libro nuevo con la StarCalc. Para eso nos iremos a [`Archivo`], [`Nuevo`], [`Hoja de Cálculo`]. Pondremos los títulos de los encabezados para cada una de las columnas, empezando en A1, insertaremos

Mes	Enero	Febrero	Marzo	Abril	Mayo
Sueldo	145.000	145.000	145.000	145.000	145.000
Vivienda	30.000	31.000	30.000	29.000	32.000
Comida	40.000	43.000	46.000	39.000	41.000
Coche	15.000	6.000	16.000	8.000	9.000
Agua	7.000	8.000	6.000	4.000	9.000
Luz	11.000	10.000	12.000	9.000	10.500
Varios	10.000	18.000	11.000	8.000	11.000

`Descripción`. Para los rótulos de las fechas introducimos la primera fecha y utilizaremos seleccionar y arrastrar para que StarCalc complete el resto de las cabeceras. Luego pondremos en la columna A todas las descripciones de los gastos e ingresos, y debajo de cada fecha las cantidades correspondientes. Para seguir con la claridad, insertaremos `Gastos` para tener todos los gastos agrupados e `Ingresos` para los ingresos. De esta manera desglosaremos los gastos e ingresos y sólo nos restaría añadir una línea para el cálculo del saldo.



	A	B	C	D	E	F	G	H
1	Descripción	enero	febrero	marzo	abril	mayo		
2	Vivienda	30000	31000	30000	29000	32000		
3	Comida	40000	43000	46000	39000	41000		
4	Coche	15000	6000	16000	8000	9000		
5	Agua	7000	8000	6000	4000	9000		
6	Luz	11000	10000	12000	9000	10500		
7	Varios	10000	18000	11000	8000	11000		
8	**Gastos**	**113000**	**116000**	**121000**	**97000**	**112500**		
9	Sueldo	145000	145000	145000	145000	145000		
10	**Ingresos**	**145000**	**145000**	**145000**	**145000**	**145000**		
11								
12	**Saldo del mes**	**32000**	**29000**	**24000**	**48000**	**32500**		
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								
23								

Año 2000 / Hoja2 / Hoja3

Figura 10.15: Tabla de datos

Luego procedemos a retocar la estética para resaltar aquellos datos que nos interesen, tal como los Ingresos, los Gastos y el Saldo. En este punto cada cual puede poner su sello personal. Como guía, resaltaremos los rótulos en negrita y las cuentas de Ingresos, Gastos y Saldo con una letra mayor, negrita y usándo bordes.

Para el cálculo de los totales usaremos la función `SUMA`, para lo que elgiremos la columna correspondiente a enero y la fila que corresponde a Gastos e introduciremos `=suma(`. En este punto tendremos dos opciones: ingresar manualmente el rango de celdas o seleccinarlas con el ratón, en ambos casos hay que terminar cerrándo el paréntesis [8].

Para calcular el resto de meses, se procederá a copiar la fórmula anterior en el resto de celdas de la fila correspondientes a los siguientes meses[9]. La StarCalc corregirá la fórmula para que coincida con los datos de cada uno de los meses.

Hasta el momento tenemos los saldos de todas las cuentas de nuestro ejemplo. Pero queremos que nuestros datos nos faciliten una mayor cantidad de información. Podemos tener muchas cosas en mente, pero para esta primera práctica vamos a ver máximos, mínimos, promedios y porcentajes.

Comenzaremos con los máximos y mínimos. Nos situaremos en la barra de rótulos ingresaremos `Máximo` en N2, `Mínimo` en O2 y`Promedio` en P2, después de los meses. En la columna de Máximos utilizaremos la fórmula `MÁX()`, con la tilde incluida porque en otro caso no será reconocida. Para el mínimo utilizaremos `MÍN()`. La secuencia a ingresar sería: en N3 `=máx(b3:m3)` y en O3 `mín(b3:m3`. En el caso de los promedios, nos situaremos en P3 y teclearemos `promedio( b3:m3)`. Luego sólo nos quedará copiar las tres celdas construidas, en cada una de las celdas correspondientes a las distintas cuentas.

La salida de los porcentajes se pueden hacer en una hoja distinta o en la misma hoja. Nosotros lo haremos en la misma hoja, aunque lo que aquí se explique se puede aplicar a otras hojas.

[8] StarCalc es capaz de cerrar el mismo el paréntesis en el caso de no escribirlo.

[9] Una manera rápida de hacerlo es seleccionado la celda a copiar y utilizando las macros. Esto es, `Ctrl+C` para copiar, seleccionar el resto de celdas y `Ctrl+V` para pegar en todas las celdas a la vez

Primero hemos de construir los encabezados y las descripciones, esto es, los meses y las cuentas. Para ello usaremos un truco que nos permitirá ahorrar tiempo en el caso de que cambie alguno de los rótulos. Nos situaremos en A15 e introducimos `=A2` y copiamos esta celda hasta M15 y luego repetimos desde A3 hasta A25.
En la celda C16[10] introducimos `=(c3-b3)/b3` y lo copiamos en el resto de celdas. No se preocupe de la salida `Err:503`, se produce porque no hay datos correspondientes a esos meses. Para que los datos sean en porcentaje tenemos que dar formato a las celdas. Seleccionaremos tadas las celdas que contengan número, incluso las que contengan errores, y accederemos a [`Formato`], [`Celda`] y presionaremos las pestaña `Números` si no se encuentra activada, seleccionaremos `Porcentaje` en Categoría. Presionaremos `Aceptar` y tendremos la salida con porcentajes.
De esta manera hemos obtenido las variaciones porcentuales. Pero, ¿no sería interesante ver el porcentaje del total que significa cada cuenta? Repetimos el paso anterior para copiar las etiquetas correspondientes a los meses y a las cuentas, empezando en A28. Y añadimos en B30 = `b3/b$9`, esta fórmula sólo se copiará hasta la cuenta Gastos, que tendrá una salida del 100 %. En B35 introducimos `=b10/b$11` y lo copiamos en todas las cuentas de Ingresos. Luego seleccionamos la columna de enero y la pegamos en el resto de meses. De esta manera obtenemos el peso de cada una de las cuentas en los distintos meses.

Por último, sólo nos queda añadir a las tablas efectos de resalte para destacar aquellos datos que nos interese tener mejor visualizados. También puede añadirse a la práctica una tabla de datos en la que se muestren los datos acumulados [11] de mes a mes y, de esta manera, llevar un control de los gastos e ingresos que llevamos para todo el año.

10.3.8.2. Productos de financiación.

Objetivo. Calcular las mensualidades del pago de un préstamo o el rendimiento de los ahorros.

10.3.8.3. Análisis de proyectos de inversión.

Objetivo. Ayuda a la toma de decisiones en inversiones.

10.3.8.4. Seguimiento de Acciones y opciones.

Objetivo. Ver la evolución de los títulos de acciones y opciones que tengamos. Modelo Black-Scholes. Hoy en día no es extraño que una persona particular posea una pequeña cantidad de acciones o de opciones. En esta práctica intentaremos hacer el seguimiento de las rentabilidades de dichos títulos tratándo de anticipar posibles cambios que nos indiquen pautas de compra o venta y así evitar riesgos. Aquí haremos referencia al parqué de la bolsa madrileña por ser la que más conicimientos poseeo. Para aquellas personas que conozcan la temática no hace falta que se expliquen los conceptos. Para aquellos que no conozcan los términos, el objetivo de la práctica son los conceptos de hoja de cálculo y no los de economía financiera.

Para hacer el trabajo más fácil de hacer y de presentar se hará uso de de cuatro hojas diferentes: una para las acciones, otra para las opciones, otra para los cálculos de acciones y opciones y una cuarta para la salida de gráficos.

El primer paso es crear un libro nuevo de StarCalc para lo que usaremos el menú [`Archivo`], [`Nuevo`] y [`Hoja de cálculo`]. Una vez creado el libro, comezaremos con la parte de acciones que es la más fácil pues sólo se trata de controlar lo cotización diaria de los títulos.

La fórmula de Black-Scholes:

$$C(K,S,T-t,r,\sigma) = S\Delta N(d_1) - K\Delta e^{-r(T-t)}\Delta N(d_2)$$

10.3.8.5. Análisis de tendencia.

Objetivo. Calcular las previsiones para años futuros dada una serie de datos con un modelo sencillo. Los datos a utilizar son los siguientes y que se presentan de esta manera por economía de espacio.

Se va a indicar la manera de hacer esta práctica paso a paso, por lo que no será necesario el conocimiento previo ni de análisis de series de datos ni del uso de la hoja de cálculo [12]

[10] No habrá salida para enero porque no tenemos datos del mes anterior
[11] la fórmula a usar para la salida sería `=b3` para el primer mes y `=b40+c3` y copiar para el resto de celdas esta última. Nótese que b40 hace referencia a la primera cuenta del mes de febrero y que podría ser otra en su ejemplo.
[12] El lector que haya seguido este curso puede tratar de personalizar cada uno de los pasos, o saltar aquellos que considere que no hacen falta.

t	1	2	3	4	5	6	7	8
X_t	99,93	103	103,41	112,79	109,52	113,57	110,5	120,42
t	9	10	11	12	13	14	15	16
X_t	118,23	119	118,75	127,32	124,93	128,13	127,55	137,06
t	17	18	19	20	21	22	23	24
X_t	131,95	133,91	135,35	146,19	138,57	143,14	144,64	153,72

- El primer paso sería introducir los datos de la tabla anterior en una columna que denominaremos X.
- Segundo paso, representar la serie X en el tiempo. Para eso seleccionamos los datos de la columna X, y vamos a [`Insertar`] [`Diagrama`]. Seguiremos los pasos de la sección *Gráficos* en la página 124.
- Supongamos:

$$X_t = T_t + S_t + I_t$$
$$T_t = 100 + 2_t$$

- Obtener el componente tendencial, T_t.
  En la celda B1, ponemos `t` introducimos `1`, en B3 escribimos `=1+b2`. Copiamos la celda B3 desde B4 hasta B25. En la celda C1 escribimos Tt. En C2, `=100+2*b2` [13]. Y copiamos C2 desde C3 hasta C25.
- Componente estacional, S_t.
  Definimos las variables cualitativas:
- Obteniendo las variables cualitativas estacionales.
  En D1 escribimos `d1`, en D2 ponemos `1`, en las celdas D3, D4, D5 se escribe `0`. En E1, `d2`, para E2, escribimos `1`, y en E2, E4 y E5, `0`. Para F1, `d3`, `1` en F4, `0` para F2, F3 y F5. En G1 ponemos `d4`, em G2, G3, G4 `0`, para G5 `1`. Se seleccionan las celdas D2 hasta G5 y se copian desde D6 hasta G25.
  En la celda H1 escribimos `St`, y en H2 `=-1*d2-1*e2-2*f2+4*g2`, copiaremos esta casilla desde H3 hasta H25.
- Componente irregular, I_t. En la celda I1 se escribe `It`. En la celda I2 `=a2-c2-h2`. Copiaremos I2 desde I3 hasta I25. Calculamos el valor medio de la serie: en I26 insertamos `=suma(i2:125)/24`.
- El último paso que faltaría es representar gráficamente las series T_t, S_t e I_t frente al tiempo.

Para finalizar, lo más aconsejable es guardar los datos con [`Archivo`] y [`Guardar`].

10.4. StarDraw: Creador de dibujos

10.4.1. Importancia de los gráficos

La mayor parte de los documentos que se manejan en informática contienen **texto y gráficos**. También se puede ver en libros, folletos, carteles publicitarios, etc. que el uso combinado de texto y gráficos resulta ser muy bueno para comunicar ideas.

Cuando se maneja un gráfico en informática, nos interesa el **resultado final** y también la **facilidad de manejo**. Con los dos tipos de gráficos que hay se puede conseguir la misma calidad, pero el trabajo que demandan y la manera de manejar cada uno hace que sea importante conocer las distintas características de los dos tipos.

10.4.2. Tipos de gráficos

Existen dos tipos: los gráficos de **mapa de bits**, también conocidos por su nombre en inglés de gráficos *bitmap*, y los gráficos **escalables** o **vectoriales**. Cualquiera de los dos tipos permite manejar imágenes en **blanco y negro**, en **escala de grises** y en **color**.

La diferencia entre los dos tipos es interna. En los gráficos bitmap, la imagen se almacena como un conjunto de puntos, dispuestos en filas y columnas. Cada punto se llama **píxel** y puede tener un color o nivel de gris distinto. En los gráficos

[13] Este es el valor para t=1

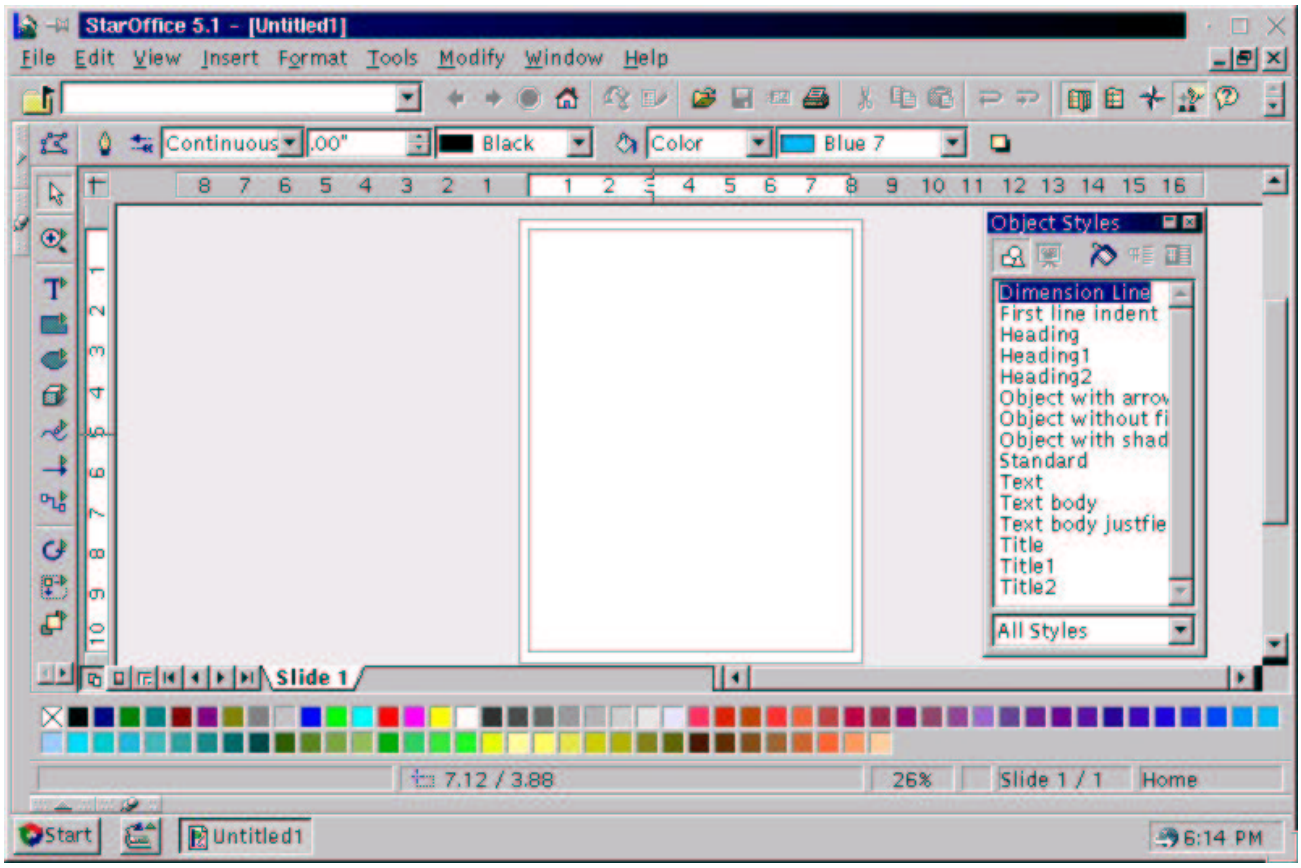


Figura 10.16: El creador de ilustraciones de StarOffice

escalables lo que se almacena es una descripción matemática de las rectas, curvas, rellenos, etc. que definen cada elemento del gráfico; los elementos pueden ser rectángulos, elipses, curvas, etc.

Veamos un ejemplo del mismo gráfico almacenado como bitmap y como escalable:

El gráfico de la izquierda representa una línea cerrada compuesta por tres segmentos rectos y uno curvo. Para poder apreciar bien los píxeles, se ha creado un gráfico muy pequeño y luego se ha ampliado. Normalmente, tendría muchos más puntos y por lo tanto mucho mejor aspecto.

El gráfico de la derecha representa la misma figura pero como gráfico escalable. Lo único que habrá que conocer son las coordenadas de los puntos A, B, C y D para saber por dónde pasa la figura, y los puntos E y F para conocer la curvatura del segmento AB. Con seis pares de coordenadas es suficiente para representar la figura, pero los programas que quieran representarla o imprimirla deberán calcular el resto de los puntos.

10.4.3. Gráficos bitmap

Normalmente se obtienen gráficos bitmap cuando se digitalizan imágenes reales con los aparatos llamados en inglés *scanners*. Por eso a los programas de alta gama que manejan este tipo de gráficos se les llama programas de **retoque fotográfico**.

Si se quieren usar para imprimirlos deben ser de gran tamaño. Cuando son pequeños se suelen usar para representar iconos en pantalla, ya que se pueden representar con mucha rapidez.

Suele ser difícil manipular gráficos bitmap y uno de los mayores inconvenientes que tienen es su pérdida de calidad cuando se reducen o amplían. Vemos un ejemplo de un gráfico bitmap ampliado:

Se puede apreciar fácilmente la pérdida de calidad. Sobre todo en las líneas diagonales y en las curvas se aprecia la aparición de «dientes de sierra». Esto se llama **efecto escalonamiento**.

10.4.3.1. Resolución

La resolución de un gráfico bitmap es la cantidad de píxeles que tiene en cada dimensión. Es muy habitual que la resolución sea la misma que la de una pantalla (640x480, 800x600, 1024x768,...) pero para imprimir el gráfico con calidad hacen falta resoluciones mayores.

10.4.3.2. Profundidad

Se llama profundidad a la cantidad de bits que hacen falta para representar el valor de cada píxel. Un dibujo en blanco y negro tiene profundidad 1, ya que con un bit es suficiente para saber si el punto es blanco o negro. Las imágenes en escala de grises y en **paleta de color** tienen profundidades 4 (16 valores) u 8 (256 valores). Las imágenes en **color real** tienen una profundidad de 24, ya que de cada punto se necesita saber su componente roja, verde y azul, a 8 bits cada una.

10.4.4. Gráficos escalables

Son los que crean los diseñadores que trabajan con ordenador. El gráfico se va creando figura a figura. Son muy fáciles de modificar, pero si son complejos requieren muchos cálculos. Su principal ventaja es la que les da nombre: se pueden cambiar de tamaño sin pérdida de calidad; esto permite imprimirlos usando al máximo la resolución de la impresora.

10.4.4.1. Curvas de Bézier

Uno de los métodos más utilizados en los gráficos escalables es la descripción de las curvas como curvas de Bézier. Se puede ver un ejemplo en el primer gráfico de esta hoja: el segmento AB es una curva de Bézier. Los puntos E y F se llaman **puntos de control**. Sirven como "imanes" que atraen a la curva y le dan su forma característica.

10.4.5. Conversión de gráficos

Es posible convertir gráficos de tipo bitmap a escalable y viceversa, pero los procesos son muy distintos:

De escalable a bitmap. El proceso se llama en inglés *raster*. Todos los programas lo saben hacer, ya que la única manera de poder ver en pantalla o imprimir un gráfico escalable es convertirlo previamente en bitmap.

De bitmap a escalable. Esto se llama **vectorización**. Es muy difícil y suele dar malos resultados, a no ser que el dibujo tenga bordes muy nítidos.

10.4.6. Compresión de datos

Como los ficheros de gráficos suelen tener gran tamaño, sobre todo los bitmaps, se han desarrollado técnicas para almacenar la información usando menos bytes. Existen muchas, aunque casi todas intentan detectar la existencia de grandes zonas de imagen del mismo color: en vez de almacenar todos sus puntos, se almacena su número y su color. Dependiendo de la imagen, se puede convertir el fichero en incluso la décima parte de su tamaño original. Existen dos tipos de compresión: sin pérdidas y con pérdidas. El primer tipo es mejor para imágenes artificiales y el segundo para las naturales.

10.4.7. Formatos de ficheros

Un mismo gráfico, sea bitmap o escalable, se puede almacenar en un fichero de muchas formas distintas, que reciben el nombre de **formatos**. Hay muchos programas que permiten convertir imágenes de un formato a otro. De todas formas, los buenos programas pueden leer y escribir ficheros en muchos formatos distintos. Algunos de los más conocidos son:

BMP. Usados ampliamente por Microsoft Windows y OS/2. Son bitmap.

TIFF. Típicamente obtenidos con escáneres. Son bitmap. Admiten muchos tipos de compresión y todo tipo de profundidad. Se pueden leer indistintamente en sistemas GNU/Linux, Windows, UNIX y Macintosh.

EPS. Significa *Encapsulated PostScript*. El estándar en el mundo de la autoedición. Son escalables. Totalmente compatibles con impresoras y filmadoras PostScript.

JPEG. El formato más difundido con compresión con pérdidas. Ideal para manejar fotografías. Es posible controlar el grado de compresión: a mayor compresión, menor calidad de imagen.

10.4.8. Las ventanas

En StarOffice cada ventana de documento tiene sus peculiaridades, para poder ofrecer las funcionalidades que demanda cada módulo. Aunque comparten muchas características, hay pequeños detalles que distinguen las ventanas de cada tipo de documento. En la ilustración de la derecha se muestra una ventana de StarOffice que contiene un documento de Draw. Normalmente se trabaja con la ventana del documento maximizada, pero aquí se muestra en posición *flotante*, para poder apreciar mejor qué componentes pertenecen a la **ventana de aplicación** de StarOffice y cuáles a la **ventana de documento** de Draw.

10.4.8.1. La ventana principal

Recorriendo desde arriba hacia abajo la ventana principal, vemos:

- La barra de título.
- El menú principal.
- La barra de funciones.
- La barra de objetos del desktop.
- La zona de trabajo, en la que está la ventana del documento draw.sda.
- La barra de tareas.

10.4.8.2. La ventana del documento

Si repasamos desde arriba hacia abajo la ventana del documento, nos encontramos:

- La barra de título.
- La barra de objetos de dibujo/imagen.
- La regla horizontal.
- La zona de trabajo (donde se prepara el dibujo).
- La barra de desplazamiento horizontal, con los botones de vistas y las pestañas de los dibujos a la izquierda.
- La barra de opciones, que por defecto no está activada.
- La barra de colores, que por defecto presenta dos líneas de colores, pero que cambia el número de líneas arrastrando el extremo superior.
- La línea de estado, con información sobre el documento.

Y si la repasamos de izquierda a derecha, tenemos esto:

- La barra de herramientas.
- La regla vertical.
- La zona de trabajo.
- La barra de desplazamiento vertical.

10.4.9. Las reglas y la línea de estado

Aunque son muy útiles y normalmente se mantienen a la vista, se pueden eliminar. En el menú [`Ver`] se encuentran la opciones [`Reglas`] y [`Barra de estado`] para regular su aparición.

10.4.10. Determinación del papel

Para determinar el tamaño y la orientación del papel que se va a utilizar hay que elegir en el menú [Formato] la opción [Página], para obtener el cuadro de diálogo **Preparar página**. En él se elige la ficha Página, como se ve aquí:

La lista desplegable de la sección Formato de papel presenta una relación con algunos tamaños muy comunes (el más usado es el *A4*). La orientación se elige con los botones de opción Vertical y Horizontal. Cuando todo está bien, se pasa a especificar otros parámetros del cuadro de diálogo o se pulsa el botón Aceptar.

Si ninguno de los tamaños de la lista coincide con el deseado, se puede especificar en los cuadros Ancho y Altura las dimensiones exactas del papel.

Si no se elige ningún tamaño, el programa usa por defecto el *A4*.

10.4.10.1. Márgenes generales

Los márgenes se pueden definir con gran precisión y en varias unidades distintas. Lo normal es escribirlos en centímetros con uno o dos decimales. Por defecto, el programa presenta la característica de que los márgenes tienen «imán», de modo que cuando se desplazan los objetos con el ratón, es muy fácil dejarlos alineados con los márgenes.

Aunque son muy útiles y normalmente se mantienen a la vista, se pueden eliminar. En el menú [Ver] se encuentran la opciones [Reglas] y [Barra de estado] para regular su aparición.

10.4.11. Objetos básicos

El principio de la creación de dibujos vectoriales es la acumulación de objetos simples para formar el resultado final. En esta hoja se presenta el modo de crear los objetos de dibujo más sencillos, dejando para otras hojas su modificación, así como la creación de objetos más complejos, como curvas de Bézier y objetos de texto. Los objetos sencillos son sin embargo las piezas fundamentales de los diseños y por tanto no hay que menospreciar su importancia.

10.4.11.1. La barra de herramientas

En los programas de dibujo en general, la barra de herramientas es un componente fundamental, porque es la que decide el modo de trabajo: es completamente distinto encontrarse seleccionando objetos que dibujando rectángulos o polígonos, por ejemplo. La barra de herramientas de StarOffice Draw presenta este aspecto cuando se elige la apariencia flotante y orientación horizontal (normalmente está anclada a la izquierda y vertical):

La primera herramienta por la izquierda es la que más se utiliza, la de selección. Sirve para elegir uno o más objetos y manipularlos. Cuando se utiliza otra cualquiera de las herramientas, Draw devuelve el control a la de selección. Si se desea usar de manera continuada alguna herramienta, hay que seleccionarla con una doble pulsación.

10.4.12. Método para crear

Todas las herramientas que se van a ver a continuación funcionan de un modo muy similar:

1. Se selecciona la herramienta.
2. Se pulsa en un punto y se arrastra hasta otro punto.
3. La figura está creada con la esquina superior izquierda en el primer punto y con la inferior derecha en el segundo.

Y durante el arrastre del ratón se admiten estas acciones:

- Si se pulsa Esc se anula la creación de la figura.
- Si se arrastra pulsando Alt, el primer punto es el centro de la figura y el segundo define el tamaño.
- Si se arrastra pulsando Shift, la figura es regular.
- Si se arrastra pulsando Ctrl, la posición y el tamaño encajan en una cuadrícula predeterminada con puntos cada 0,25 cm (aunque es configurable).

10.4.12.1. Rectángulos

A la derecha se ve la barra de herramientas que se usa para crear rectángulos. Se pueden crear con o sin relleno y con o sin las esquinas redondeadas. También se pueden crear cuadrados (que no son más que un caso particular de rectángulo) directamente, sin recurrir a pulsar Shift.

10.4.12.2. Elipses

A la derecha se ve la barra de herramientas que se usa para crear elipses. Se pueden crear con o sin relleno y completas o parciales. También se pueden crear circunferencias (que no son más que un caso particular de elipse) directamente, sin recurrir a pulsar Shift.

10.4.12.3. Líneas y flechas

A la derecha se ve la barra de herramientas de línea, que también permite crear flechas, ya que para StarOffice Draw los objetos *línea* y *flecha* pertenecen a la misma clase y siempre se puede pasar de uno a otro.

10.4.12.4. Objetos 3D

A la derecha se ve su barra. Todas la figuras se crean igual, ya que el usuario lo único que define al crearlas es el tamaño, luego el programa se encarga de representar la forma.

10.4.13. Conectores

Estos objetos son un poco diferentes a los ya explicados, y se crean de una manera ligeramente distinta. Los conectores son líneas que unen otros dos objetos entre sí, con la peculiaridad de que si se modifican estos objetos, el conector cambia automáticamente para acomodarse a la nueva situación. Existe una gran variedad de ellos, como puede verse a la derecha, en la barra de herramientas Conector.

Para crearlos, se sigue este método:

1. Se elige la herramienta.
2. Al acercar el puntero a un objeto, se marcan los posibles puntos de unión que ofrece el objeto; se pulsa el ratón y se mantiene.
3. Se arrastra el ratón hasta llegar al segundo objeto, que también mostrará sus puntos de unión. Se suelta el ratón en el punto deseado.
4. El conector queda dibujado.

10.4.14. Polígonos

Para crear polígonos abiertos o cerrados se usan los cuatro iconos centrales de la barra de herramientas Curvas. Los dos iconos de la derecha crean polígonos con ángulos rectos o de 45°; los de la izquierda, polígonos generales.

Para crearlos, se sigue este método:

1. Se elige la herramienta.
2. Se pulsa en el primer punto, sin soltar el ratón.
3. Se arrastra hasta llegar al segundo punto, en el que se suelta el ratón.
4. Se mueve el ratón al tercer punto, donde se pulsa o se pulsa y arrastra.
5. Se continúa como en el paso 4, añadiendo los puntos necesarios.
6. El último punto se define con una doble pulsación. El programa cerrará el polígono si se estaba preparando uno cerrado.

10.4.15. Modificación rápida

Aunque más adelante se explicará con detalle cómo modificar los objetos, ahora se dan unas pautas para poder experimentar sin más dilación:

- Se selecciona un objeto pulsando sobre él con la herramienta de selección.
- Se mueve arrastrándolo.
- Se cambia su tamaño arrastrando los manejadores.
- Se cambia la línea que lo rodea y el relleno usando la Barra de objetos de dibujo/Imagen:
- Los colores de línea y relleno se pueden elegir en la barra de colores: con el botón izquierdo se elige el relleno y con el izquierdo la línea.
- Se cambia su forma pulsando el botón Modificar puntos y arrastrando los cuadrados que aparecerán en varios puntos. Véase el ejemplo de la derecha.
- Se elimina pulsando `Supr`.

10.4.16. Origen de las Curvas de Bézier

El ingeniero aeronáutico francés Pierre Bézier trabajaba en la empresa Renault y estaba diseñando la forma de los parachoques de los coches. Necesitaba un sistema eficiente y flexible para representar dicha forma, e inventó estas curvas. Hoy en día se utilizan en muchas áreas de la informática, como tipografía e infografía, además de en los programas de diseño.

10.4.17. Nomenclatura de las Curvas de Bézier

Aunque los conceptos que intervienen en una curva de Bézier son claros, las palabras que los representan a veces no lo son tanto. Además, cambian de programa en programa. En estas hojas se utilizará la traducción al español utilizada en StarOffice, que no coincide con la que se puede leer en otros lugares.

10.4.18. Conceptos de las Curvas de Bézier

Una curva de Beziér está formada por varios **segmentos**, pueden ser **curvos** o **rectos**. La curva puede ser **abierta** o **cerrada**. Éste es un ejemplo de curva de Bézier, sobre el que se explicarán los distintos conceptos:

Puntos de apoyo son los puntos extremos de los segmentos. Por ellos pasa la curva y siempre hay que definirlos. En general, cuando se crean curvas de Bézier se procura que haya la menor cantidad posible de puntos de apoyo. En el ejemplo, del A al G.

Puntos de control son los puntos a los que «intenta acercarse» la curva, aquéllos que definen su curvatura. Siempre hay que definirlos. La curvatura de cada segmento viene definido por un punto de control, dos o ninguno. En el ejemplo, del 1 al 7.

Líneas de control unen los puntos de apoyo con los puntos de control. Son meras referencias para ayudar en la creación de las curvas, luego no aparecen.

10.4.18.1. Tipos de puntos de apoyo

En StarOffice Draw cada punto de apoyo puede ser de estos tipos:

Punto de esquina. Tiene dos líneas de control que forman un ángulo. La curva presenta un brusco cambio de curvatura al pasar por un punto de esquina. En el ejemplo, el F.

Punto liso. Tiene dos líneas de control que forman una línea recta; la distancia del punto de apoyo a los puntos de control no tiene por qué ser la misma. La curva tiene distinta curvatura a un lado o a otro del punto de apoyo. En el ejemplo, el E es el más claro.

Punto simétrico. Tiene dos líneas de control que forman una línea recta y la distancia del punto de apoyo a los puntos de control es la misma. La curva tiene la misma curvatura a uno y otro del punto de apoyo. En el ejemplo, el B.

10.4.18.2. Tipos de segmento

El tipo de un segmento es una característica determinada por el primero de los puntos de apoyo que lo definen.

Segmento curvo. Su curvatura estará definida por un punto de control (que corresponderá al primer punto de apoyo) o por dos (cada uno correspondiente a un punto de apoyo).

Segmento recto. No le corresponde ningún punto de control. En el ejemplo, el CD.

10.4.19. Creación

El modo de crear curvas de Bézier en StarOffice Draw presenta algunas limitaciones que a veces impiden crear exactamente la curva deseada; pero una vez creada, es muy fácil modificarla. Concretamente, tiene estas limitaciones:

El primer segmento ha de ser curvo.

Sólo se puede definir el primer punto de control del último segmento, ya que el segundo queda obligatoriamente oculto por el último punto de apoyo.

Cuando se define el segundo punto de control de un segmento, se está definiendo también el primer punto de control del siguiente segmento. El punto de apoyo intermedio entre los dos segmentos queda definido como liso, pero con las distancias como si fuera simétrico.

10.4.19.1. Procedimiento

1. En la barra de herramientas se elige Curva,rellena o Curva según se vaya a crear una curva cerrada o abierta.
2. Se pulsa en el primer punto de apoyo de la curva y no se suelta el ratón.
3. Se arrastra hasta llegar al primer punto de control del primer segmento, donde se suelta el ratón.
4. Ahora hay dos posibilidades:
   a) Se pulsa y arrastra para definir el segundo punto de control.
   b) Se pulsa para definir el segundo punto de apoyo. En ese caso, el siguiente segmento será recto.
5. Se continua con los pasos 3 y 4 definiendo más puntos de apoyo y control.
6. En el último punto de apoyo, se hace una doble pulsación. El programa cerrará la curva si se estaba preparando una cerrada.

10.4.20. Edición

Una vez creada una curva de Bézier se puede modificar la posición y carácter de sus puntos de apoyo y de control. Se selecciona la curva y se pulsa el botón Modificar puntos, de la barra de objetos, y ésta se sustituye por la barra de objetos de Bézier, que aparece aquí:

- Para seleccionar un punto de apoyo basta pulsar sobre él. Para seleccionar más de uno, se pueden «atrapar» marcando un rectángulo que los contenga. También se pueden seleccionar o deseleccionar de uno en uno pulsando con `Shift` pulsada.
- Una vez seleccionados, se pueden desplazar arrastrándolos y cambiar su carácter con los botones de la barra de opciones.
- Se pueden eliminar puntos de apoyo seleccionándolos y pulsando [Supr].
- Se pueden añadir puntos de apoyo usando el botón Insertar puntos.
- A veces hay un punto de control sobre un punto de apoyo. Se sabe cuál se va a manipular por la forma del puntero: con un cuadradito los de apoyo, con una curvita los de control.
- Para terminar la edición de la curva se vuelve a pulsar el botón Modificar puntos.

10.4.21. Trazados a mano alzada

Se pueden definir curvas, tanto cerradas como abiertas, sin más que dibujarlas arrastrando el ratón. Se elige una de las dos posibilidades de la derecha de la barra de herramientas Curvas, se arrastra el ratón, y al terminar el programa realiza unos cálculos y ofrece una curva de Bézier que sigue la forma dibujada. Esta curva se puede editar como cualquier otra.

10.4.22. Tipos de texto

En StarOffice Draw se pueden introducir cuatro tipos distintos de texto, cada uno con una finalidad diferente. Para poder hacer referencia a cada tipo, se usarán aquí unos nombres adaptados de la denominación oficiosa presente en la ayuda del programa.

Texto normal. Es el equivalente a textos distribuidos por párrafos que se puede encontrar en un procesador de textos. Se usa cuando hay que introducir una cantidad grande de texto, como en una explicación larga en un folleto de publicidad, por ejemplo.

Texto ajustado a marco. Es el más usado con fines decorativos. Se usa para textos cortos sobre los que haya que aplicar efectos. Por ejemplo, el título de una película en un cartel.

Texto en leyenda. Consiste en texto dentro de un cuadro y con una flecha que señala a algún lugar. Es muy útil para hacer anotaciones que expliquen la función de otros objetos de dibujo. Por ejemplo, para escribir los nombres de las piezas de una máquina.

Texto en objeto. Consiste en un texto contenido dentro de un objeto, que se desplaza y cambia con él. Permite hacer organigramas muy fácilmente, por ejemplo.

10.4.23. Introducción de texto

Para introducir texto normal, ajustado o en leyenda se usa la barra de herramientas Texto, que se ve a la derecha. Para introducir texto en objeto sólo es necesario tener previamente creado el objeto.

Texto normal. Se elige su icono y luego pulsando y arrastrando en la zona del dibujo, se define un rectángulo, que contendrá al texto. A continuación aparece un punto de inserción, que invita a escribir el texto. Cuando se termina de escribir, se pulsa en algún punto vacío del dibujo.

Texto ajustado a marco. Se prepara igual que el texto normal, pero la importante diferencia es que al terminar de escribir, el texto se ajusta al rectángulo definido al principio.

Texto en leyenda. Se elige su icono y se pulsa en el punto donde debe aparecer la punta de la flecha; sin soltar el ratón, se arrastra hasta donde se quiere colocar el cuadro y se suelta el ratón. Para comenzar a escribir, se hace una doble pulsación sobre el cuadro. Cuando se termina de escribir, se pulsa en algún punto vacío del dibujo.

Texto en objeto. Se hace una doble pulsación sobre el objeto y aparece un punto de inserción en el centro del objeto, donde se escribe el texto. Cuando se termina de escribir, se pulsa en algún punto vacío del dibujo.

10.4.24. Modificación de texto

Para cambiar alguna característica de un texto, basta hacer una doble pulsación sobre él. En ese momento se puede editar el texto y cambiar las características de la parte del texto que se seleccione con el ratón. El método más rápido es usar la **Barra de objetos de texto/Draw**:

Con ella se puede elegir directamente la familia tipográfica, el tamaño, algunas variedades, el color de los caracteres, la alineación y separación de los párrafos y el interlineado.

Si se desea cambiar alguna de estas características a *todos* los caracteres a la vez, es necesario tener seleccionado el cuadro de texto, pero no estar editándolo. Esto se consigue saliendo de la edición pulsando Esc en vez de pulsando fuera del cuadro.

10.4.24.1. Los cuadros de diálogo

Además de mediante la barra de objetos, se pueden cambiar las características básicas de los caracteres y los párrafos mediante los cuadros de diálogo **Caracteres** y **Párrafo**, a los que se accede desde el menú [Formato], el menú de contexto o la barra de objetos:

10.4.25. Relación del texto con el cuadro

Los textos siempre están contenidos en algún tipo de cuadro. La relación entre los dos se puede configurar eligiendo en el menú [Formato] la opción [Texto], lo que abre el cuadro de diálogo **Texto**, que se ve a la derecha. En la ficha Texto se encuentran las opciones pertinentes en Draw (con la ficha Animación de texto se accede a efectos espectaculares, pero se utiliza en el módulo StarOffice Impress).

- La casilla de verificación Ajustar altura al texto permite que el cuadro aumente su altura automáticamente cuando se introduce más texto. Está marcada por defecto en el texto normal.
- La casilla de verificación Ajustar al marco es la que permite el comportamiento de los textos ajustados a marco.
- La casilla de verificación Ajustar al contorno permite que el texto tome la forma del objeto que lo contiene, por ejemplo tomando la curvatura de una elipse o siguiendo los ángulos de un polígono.
- En la sección Distancia al marco se define el espacio en blanco que hay que reservar entre el borde del marco y el comienzo del texto. Si se especifica una cantidad negativa, el texto saldrá del marco.
- En la sección Anclaje del texto se determina la alineación horizontal y vertical del texto.

10.4.25.1. Observación de texto

Cuando se selecciona cualquier cuadro de texto y se usa la barra de objetos de dibujo/Imagen o la barra de colores, se modifican las características del cuadro, no las del texto que contiene.

10.4.26. Selección de objetos

Para modificar y manipular objetos, primero hay que seleccionarlos. Es posible seleccionar uno o más objetos. En la línea de estado quedará reflejado qué tipo de objeto o cuántos se han seleccionado. Visualmente se aprecia también por los ocho manejadores que aparecen alrededor del objeto, o los objetos. Para seleccionar objetos hay que usar, obviamente, la herramienta de selección.

10.4.26.1. Un objeto

Pulsando sobre el objeto, se selecciona. Si el objeto no tiene relleno, es necesario pulsar sobre su línea.

Pulsando la tecla Tab se van seleccionando todos los objetos por el orden en que se han creado. Con Shift-Tab se van seleccionando en orden inverso al de creación.

Si hay varios objetos apilados, pulsando sobre ellos con Alt pulsada se van seleccionando de arriba hacia abajo y con Shift-Alt pulsada, de abajo hacia arriba.

10.4.26.2. Varios objetos

Arrastrando y soltando se marca un rectángulo, y todos los objetos contenidos íntegramente en el rectángulo quedan seleccionados.

Pulsando sobre un objeto con la tecla Shift pulsada, se añade o elimina del conjunto de objetos seleccionados.

10.4.27. Modificar puntos

El botón del mismo nombre permite activar o desactivar esta posibilidad. Si se selecciona un objeto, al marcar Modificar puntos se puede modificar la forma de cada objeto, cada uno según su naturaleza:

- En un rectángulo se puede modificar el tamaño y la curvatura de las esquinas.
- En las variedades de elipse, el tamaño y la posición de los puntos.
- En los polígonos, la posición de los vértices.
- En las curvas de Bézier, todas las características explicadas en la hoja «Curvas de Bézier».

10.4.28. Posición de objetos

El método más sencillo para cambiar la posición de los objetos seleccionados es arrastrarlos con el ratón.

Y el método más preciso es elegir en el menú Formato la opción Posición y tamaño, para ver el cuadro de diálogo Posición y tamaño, en el que se elige la ficha Posición, que se ve a la derecha.

- Las nueve casillas de opción permiten elegir respecto a qué punto se va a definir la posición.
- La coordenada horizontal se marca en Posición X y la vertical en Posición Y.
- Si se marca la casilla de verificación Proteger, ya no se podrá cambiar la posición del objeto con el ratón.

10.4.29. Tamaño de objetos

Arrastrando los manejadores de un objeto se puede cambiar su tamaño, y están disponibles estas posibilidades:

- Si se pulsa `Esc` se anula el cambio de tamaño.
- Si se arrastra pulsando `Shift`, se mantiene la proporción de las dimensiones.
- Si se arrastra pulsando `Alt`, el cambio se hará respecto al centro.
- Si se arrastra pulsando `Ctrl`, el tamaño cambiará en saltos de 0,25 cm.

Sin embargo, el método más preciso es elegir en el menú [`Formato`] la opción [`Posición`] y tamaño, para ver el cuadro de diálogo Posición y tamaño, en el que se elige la ficha Tamaño, que se ve a la derecha.

- Las nueve casillas de opción permiten elegir qué punto quedará fijo durante el cambio de tamaño.
- Si se marca la casilla de verificación Igualar, las dimensiones siempre cambiarán respetando la proporción. Si el usuario cambia una, el programa calcula la otra.

10.4.30. Rotación e inclinación de objetos

Para rotar o inclinar objetos, una vez seleccionados, se elige en la barra de herramientas Efectos el botón Rodar; aparecen ocho nuevos manejadores alrededor del objeto y un punto de mira en el centro; véanse las ilustraciones que aparecen un poco más abajo, a la izquierda.

Para rotar el objeto con el ratón, se arrastra el punto de mira para indicar el centro de giro que se desea usar y después se arrastra uno de los manejadores de las esquinas.

Para inclinar el objeto, basta arrastrar alguno de los manejadores de los lados

Si se desea más precisión, se elige en el menú Formato la opción Posición y tamaño, para ver el cuadro de diálogo Posición y tamaño, en el que se elige la ficha Rotación, que se ve un poco más arriba a la derecha, o la ficha Inclinación/Radio de ángulo.

10.4.31. Reflejo de objetos

Es posible convertir un objeto en su simétrico respecto a una recta. Lo más sencillo es el reflejo horizontal o vertical, al que se accede directamente desde el menú [Modificar], submenú [Reflejar], con sus dos opciones, [Horizontal] y [Vertical].

Si se desea que la simetría sea respecto a una recta cualquiera, el proceso es un poco más largo:

1. Se selecciona el objeto.
2. Se elige en la barra de herramientas **Efectos** el botón [Reflejar].
3. Aparece una línea (roja en la pantalla) acabada en dos puntos de mira.
4. Arrastrando los puntos de mira se cambia la dirección de la línea; arrastrando ésta se cambia la posición.
5. Se arrastra alguno de los manejadores del objeto al otro lado de la línea (como se ve a la derecha) y se suelta.

10.4.32. Barra o cuadros

Casi todas las características que se van a explicar en esta hoja están disponibles tanto en la barra de opciones como en los menús, pero algunas características adicionales estarán disponibles en menús y no en la barra.

10.4.33. Líneas

Para definir la línea que tienen los objetos se elige en el menú [Formato] la opción [Línea], lo que lleva al cuadro de diálogo **Línea**, del que se muestran sus tres fichas:

La ficha Línea está dividida en tres secciones. En la de la izquierda se definen las características de la línea; en la de la derecha se definen los dos extremos de la línea, en los que se puede añadir puntas de flecha; en la sección de abajo se ve el aspecto que tendría la línea.

Las fichas Estilos de línea y Fin de línea permiten crear nuevos estilos de línea y de puntas de flecha y guardarlos en archivos

10.4.34. Rellenos

Los objetos cerrados pueden tener varios tipos de relleno. Se elige en el menú [Formato] la opción [Relleno] y aparece el cuadro de diálogo **Relleno**, en el que se puede definir no sólo el relleno, sino alguna característica más. Éstas son las tres primeras fichas:

- La ficha Relleno es la básica, ya que en ella se elige entre los cinco tipos de relleno. Según el tipo que se elija, el resto de la ficha cambiará. El tipo Invisible permite que el objeto sea transparente y se pueda ver a través de él; el resto de los tipos se verá a continuación.
- En la ficha Sombra se puede conseguir que el programa añada automáticamente una sombra con la misma forma que el objeto.
- Con la ficha Transparencia se activa un bonito efecto, que consiste en que el objeto sea semitransparente, es decir, tiene un relleno, pero a pesar de todo se puede ver a través de él. Cuanto mayor sea el porcentaje de transparencia, más se verá a través del objeto.

10.4.34.1. Gestión de rellenos

Éstas son las cuatro últimas fichas del cuadro de diálogo:

En ellas se puede definir de un modo mucho más preciso cada tipo de relleno. La idea es tener una lista de estilos, que se puede modificar, guardar en archivos, leer de archivos, etc. y posteriormente bastará elegir el estilo de la lista en la primera ficha o en la barra de opciones.

10.4.35. Fondo de la página

Todo lo explicado sobre rellenos puede ser aplicado directamente a la página completa. Se puede definir un relleno que ocupe todo el fondo de la página: en el menú [`Formato`] se elige [`Página`] y en el cuadro de diálogo **Página** se elige la ficha **Fondo**, que se ve aquí:

Como se ve, están disponibles los cinco tipos de relleno (el establecido por defecto es Invisible). Al ir eligiendo cada tipo, la ficha va cambiando.

10.4.36. Posición

Los objetos están colocados sobre el dibujo como en una pila, unos por encima de otros. Esto no se aprecia cuando los objetos están separados, pero cuando se sobreponen, uno tapa parte del otro. La posición relativa se puede cambiar en cualquier momento mediante siete órdenes disponibles en el menú [`Modificar`], submenú [`Posición`], o con la barra de herramientas Posición, que se ve a la derecha.

- Las cuatro primeras hacen avanzar o retroceder una posición o colocar al principio o al final de la pila.
- Las dos siguientes colocan el objeto seleccionado por delante o por detrás del objeto que se marque a continuación; el programa lo pide con un puntero en forma de mano.
- La última se usa cuando hay seleccionados dos objetos y se desea invertir sus posiciones.

10.4.37. Alineación

En muchas ocasiones hay que ajustar las posiciones de los objetos en el dibujo de modo que queden alineados entre sí. La alineación se puede realizar de seis modos distintos, puesto que hay tres posibilidades en horizontal y otras tres en vertical. Las seis son accesibles desde el menú [`Modificar`], submenú [`Alineación`], o con la barra de herramientas Alineación, que se ve a la derecha.

- Si sólo se selecciona un objeto, la alineación se realiza respecto a los márgenes de la página.
- Las alineaciones de varios objetos por el centro se realizan colocando los centros de todos los objetos en la línea media que definían los extremos de los objetos; por tanto, es posible que se muevan todos los objetos.
- Las alineaciones de varios objetos por los lados se realizan colocando todos los lados requeridos alineados con uno de los objetos, que no se moverá, y es el que tenga ese lado más al extremo. Por ejemplo, al alinear por arriba, el objeto que esté más arriba no se moverá y los demás igualarán con él los lados superiores.

10.4.38. Agrupación

El modo habitual de trabajo consiste en crear un componente de un dibujo a partir de varios objetos elementales; por ejemplo, la cara de la derecha está compuesta de cinco elipses, un polígono y un rectángulo. Una vez creados y colocados los objetos elementales, lo que se hace es **agruparlos**, para formar el componente y así poder trabajar con él de modo unificado. Se seleccionan los objetos y en el menú [`Modificar`] se elige [`Agrupar`]. A partir de entonces, el programa se refiere al grupo y no a sus componentes, y se puede modificar como un objeto cualquiera.

En cualquier momento se pueden recuperar los objetos individuales, con sólo seleccionar el grupo y en el menú [`Modificar`] elegir [`Desagrupar`].

10.4.38.1. Edición

Si hay que hacer algún cambio en algún componente de un grupo, se elige en el menú [`Modificar`] la opción [`Editar grupo`]. En ese momento, se pueden volver a seleccionar individualmente los elementos del grupo, y ninguno más. Para terminar, se elige en el menú [`Modificar`] la opción [`Abandonar grupo`].

10.4.39. Combinar

La combinación de dos o más objetos crea uno nuevo, con características distintas a los originales. La propiedad más relevante de la combinación es que el nuevo objeto puede presentar "agujeros"; también se puede formar un solo objeto que esté compuesto de varias partes inconexas.

Para combinar dos o más objetos se comienza por seleccionarlos y en el menú [`Modificar`] se elige la opción [`Combinar`]. En el proceso los objetos pierden la condición que tuvieran y se convierten en curvas de Bézier.

Y si un objeto está formado por varias curvas cerradas, es posible usar la opción Descombinar para obtener dos o más objetos, uno por curva.

10.4.39.1. Ejemplo

En la figura que aparece a la derecha se han combinado un rectángulo y una elipse, ambos con relleno blanco. Obsérvese cómo después de la combinación se puede ver a través de la elipse, ya que ahora no es tal elipse, sino un agujero del anterior rectángulo.

10.4.40. Convertir en curva de Bézier

Los objetos básicos (con la excepción de los objetos 3D) y el texto ajustado se pueden convertir en curvas de Bézier, lo que permite una posterior modificación. Para hacerlo, basta seleccionar el objeto y en el menú [`Modificar`], submenú [`Convertir`], elegir la opción [`En curva`]. En la ilustración se presentan un rectángulo con bordes redondeados, un segmento de elipse y una letra que han sido convertidos a curvas, y por tanto tienen nuevos puntos de apoyo.

10.4.41. Formas

Una de las maneras más utilizadas para obtener objetos nuevos consiste en realizar ciertas operaciones con objetos existentes. En StarOffice Draw hay tres operaciones disponibles, agrupadas en el menú [`Modificar`], submenú [`Formas`]: Unir, Substraer y Cortar.

Probablemente la mejor manera de entender las operaciones sea mediante un ejemplo: se dibujan primero un rectángulo y posteriormente una elipse, que se sitúan como se ve a la izquierda. A continuación se puede ver el resultado que se obtendría con cada una de las tres operaciones.

Hay que resaltar que la operación de substraer tendría un resultado distinto si se hubieran creado las dos figuras en el orden inverso. Se pide al lector que averigüe por sí mismo ese resultado.

10.4.42. Duplicar

Cuando hay que repetir un objeto, lo más sencillo es usar copiar y pegar. Pero si hay que obtener varias copias del mismo objeto, es mucho mejor usar la orden "duplicar". Además, esta orden permite crear fácilmente algunas figuras.

Para usarla, se selecciona el objeto, se elige en el menú [`Editar`] la opción [`Duplicar`] y se ajustan los valores deseados en el cuadro de diálogo **Duplicar**, que se ve más abajo, a la izquierda. Por ejemplo, eligiendo un cuadrado transparente y aplicando los valores que se ven, se obtiene la figura de abajo a la derecha, que a su vez permite crear una estrella.

10.4.43. Deformaciones

En la barra de herramientas Efectos se encuentran tres opciones que permiten distintas deformaciones de los objetos (se muestran sus iconos a la derecha). Se llaman Posicionar en círculo (en perspectiva), Posicionar en círculo (inclinar) y Distorsionar. Su uso es muy sencillo: se selecciona el objeto, la opción y se arrastran los manejadores del objeto.

10.4.44. Disolvencia

Este efecto también se conoce como *morphing*. Consiste en que un objeto se va transformando en otro en una serie de pasos. Para aplicarlo, se seleccionan los dos objetos y en el menú [`Editar`] se elige la opción [`Disolvencia`]; en el cuadro de diálogo **Disolvencia** se decide cuántas etapas se desean y el comportamiento que debe tener el programa con los atributos del objeto. Más abajo aparece el cuadro de diálogo y el resultado de la disolvencia entre un polígono y una elipse.

10.4.45. Conversión a 3D

Cualquier objeto plano, o grupo de objetos planos, se puede convertir en tridimensional. Esto se conoce como «extrusión». Para aplicar el efecto, se selecciona el objeto y en el menú [Modificar], submenú [Convertir], se elige la opción [En 3D].

10.4.46. Cuerpos de rotación

A partir de una figura plana se puede crear una figura tridimensional mediante la rotación respecto a un eje. Los cuerpos así generados se llaman «cuerpos de revolución». En StarOffice Draw se pueden crear eligiendo en el menú [Modificar], submenú [Convertir], la opción [En cuerpo de rotación 3D]. Si se hace con el botón del mismo nombre de la barra de herramientas Efectos, será posible definir el eje de rotación.

10.4.47. Efectos 3D

Las figuras tridimensionales tienen su propio rango de efectos en el programa. La rotación es diferente a la que se realiza con figuras planas, ya que es una rotación en el espacio. Pero el modo más específico de manejar un objeto 3D es eligiendo en el menú [Formato] la opción [Efectos 3D], que abre el cuadro de diálogo **Efectos 3D**, que se muestra a la derecha.

10.4.48. FontWork

Uno de los efectos más habituales es deformar un texto para que siga el recorrido de una curva. Casi todos los programas de diseño tienen este efecto, pero cada uno lo denomina de una forma distinta. En StarOffice se llama FontWork. Se selecciona el texto, en el menú [Formato] se elige la opción [FontWork], y en el cuadro de diálogo **FontWork**, de muy fácil manejo, se van definiendo todos los parámetros, mientras se va viendo el resultado. Éste es un ejemplo:

10.4.49. Entrega de un trabajo

Las ilustraciones creadas en un programa de diseño gráfico se deben mostrar al exterior. Hacerlo en formato nativo, es decir, el propio del programa, es lo mejor para preservar todos los efectos y poder modificarlos, pero no es lo habitual. Normalmente los trabajos se entregan en un formato que no admita apenas retoques. Y los más comunes son el papel o ficheros gráficos bitmap. Para dar el trabajo en papel hay que imprimir el documento y para darlo en formato bitmap hay que exportarlo.

10.4.50. Imprimir

En el menú [Archivo] se elige [Imprimir] y aparece el cuadro de diálogo **Imprimir**, que se muestra abajo, a la izquierda. Las opciones que aparecen en él son las habituales, pero es importante saber que pulsando el botón [Opciones] se accede al cuadro de diálogo **Opciones** de impresión, se ve abajo, a la derecha.

En la sección **Opciones de página** se encuentran dos posibilidades muy útiles:

- La casilla de verificación Ajustar al tamaño de la página permite imprimir el trabajo en un tamaño de papel diferente al que esté establecido para el dibujo, ya que el programa se encargará de escalar todos los objetos adecuadamente.
- La casilla de verificación Páginas como azulejos permite imprimir un trabajo en el que el tamaño de página definido sea mayor que el realmente accesible; lo que hace el programa es imprimir el trabajo a escala real, pero en varias hojas, que luego se cortan y montan.

10.4.51. Exportar

En el menú [Archivo] se elige [Exportar], lo que abre el cuadro de diálogo **Exportar**. Tiene el mismo aspecto y uso que el cuadro de diálogo **Guardar como**. Lo más importante es elegir el formato con que se desea generar el archivo. Los formatos disponibles están en la lista desplegable Tipo de archivo; aparecen formatos escalables y bitmap, todos los tipos para los que se hayan cargado los filtros correspondientes cuando se instaló el programa.

10.4.51.1. Opciones por tipo de archivo

Algunos de los tipos de archivo piden opciones de exportación adicionales antes de generarse el archivo. Los datos que más comúnmente se piden son el nivel y método de compresión, la profundidad de color y la resolución. A continuación aparecen varios de los cuadros de diálogo que piden esos datos:

10.4.52. El problema de la resolución

Cuando se exporta un gráfico vectorial a un formato bitmap es crucial poder decidir el tamaño en píxeles del resultado. Según se acaba de ver, esto es algo que StarOffice Draw sólo admite con algunos formatos, la minoría. Realizando algunas pruebas, se descubre que el programa utiliza la resolución de pantalla para calcular el tamaño. Esta resolución se suele medir en puntos por pulgada (abreviado a «ppp»), en inglés *dots per inch* (abreviado «dpi»). De modo que, conocida la resolución y el número de puntos que se desea obtener, una sencilla operación matemática da las dimensiones que hay que dar a la página. Además, con más pruebas, se ha visto que al exportar sólo se genera la parte de la imagen que esté dentro de los márgenes de la página, pero no la página completa.

10.5. StarSchedule: Agenda

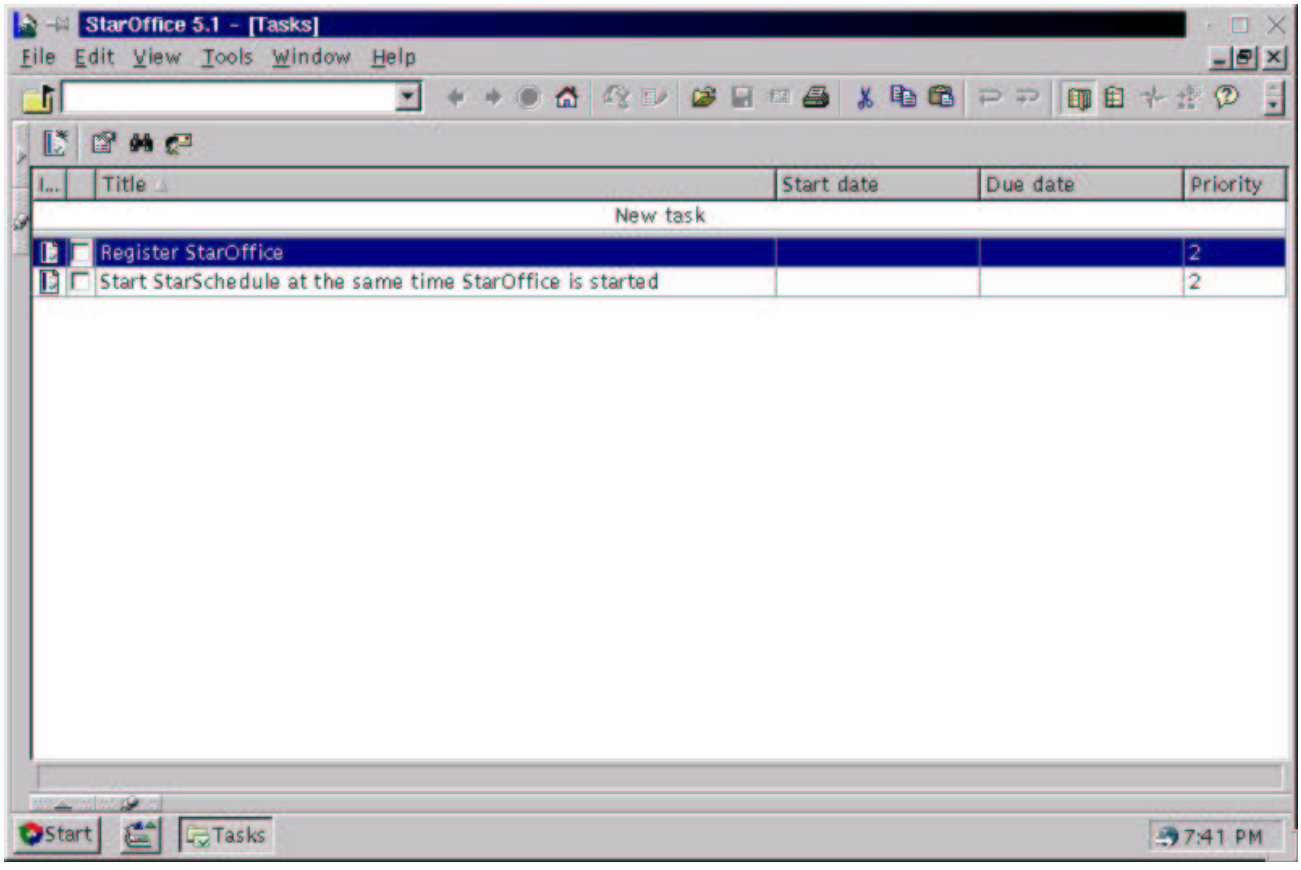


Figura 10.17: Organizar las tareas con StarSchedule

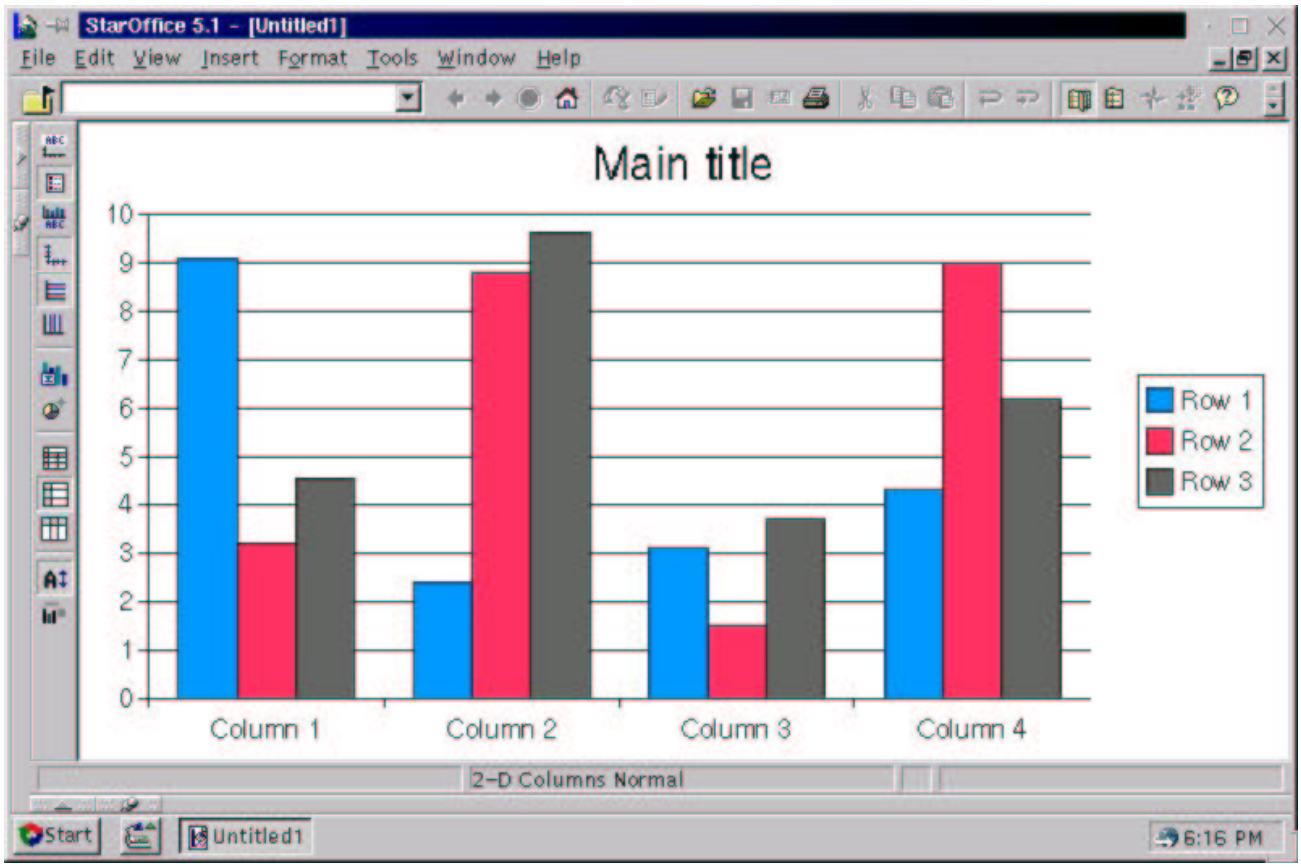


Figura 10.18: Armado de gráficas en StarOffice

10.6. StarChart: Generador de gráficas

10.7. StarImage: Editor de Imágenes

10.7.1. Crear

10.7.1.1. Nueva imagen

Cuando se comienza una nueva imagen en StarOffice y se arranca el módulo Image, lo primero que aparece es el cuadro de diálogo [`Nueva imagen`], en el que hay que definir los dos parámetros básicos de cualquier gráfico bitmap: las dimensiones y la profundidad de color.

10.7.1.2. La ventana de Image

En la ilustración de la derecha se muestra una ventana de StarOffice que contiene un documento de Image. Normalmente se trabaja con la ventana del documento maximizada, pero aquí se muestra en posición *flotante*, para poder apreciar mejor qué componentes pertenecen a la **ventana de aplicación** de StarOffice y cuáles a la **ventana de documento** de Image.

10.7.1.3. La ventana principal

Recorriendo desde arriba hacia abajo la ventana principal, vemos:

- La barra de título.
- El menú principal.
- La barra de funciones.

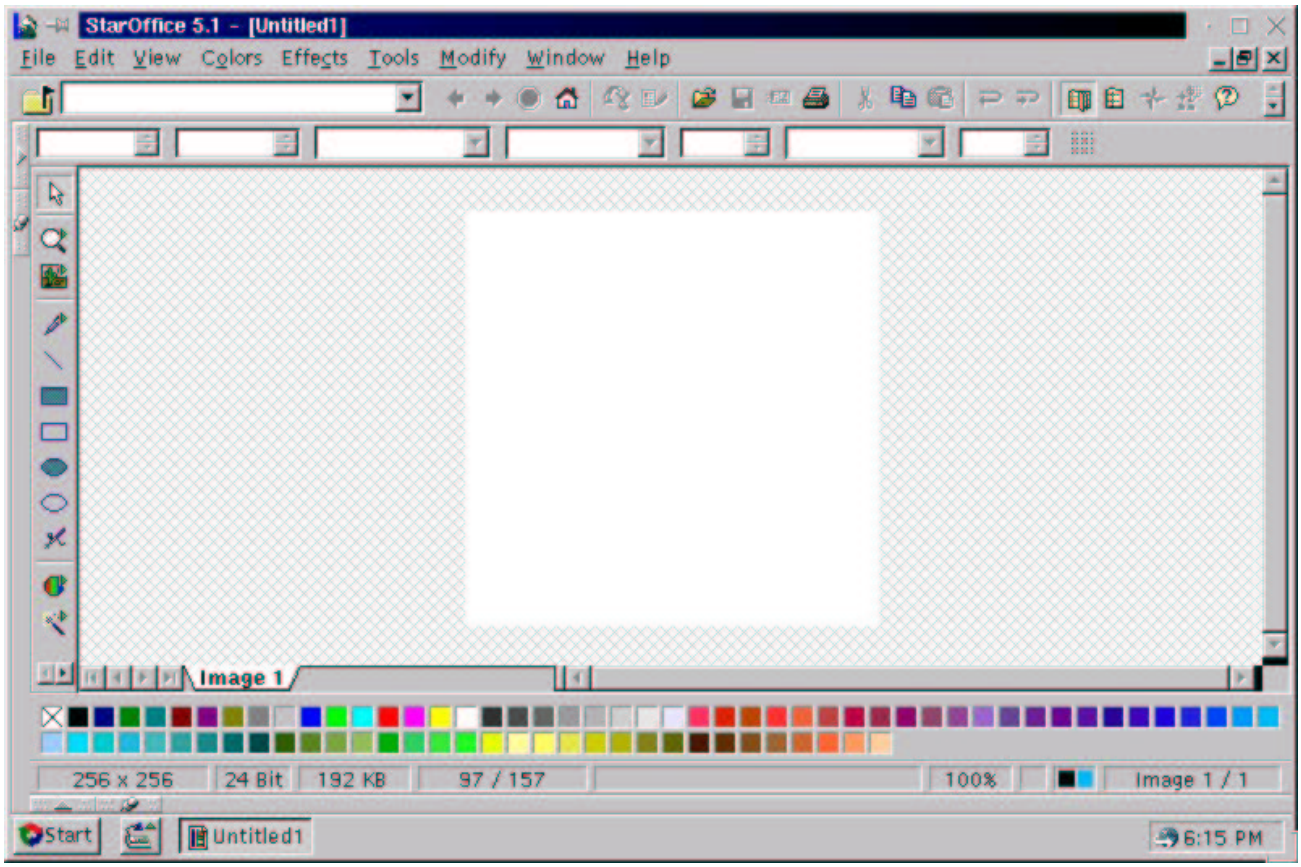


Figura 10.19: Retocador de imágenes del starOffice

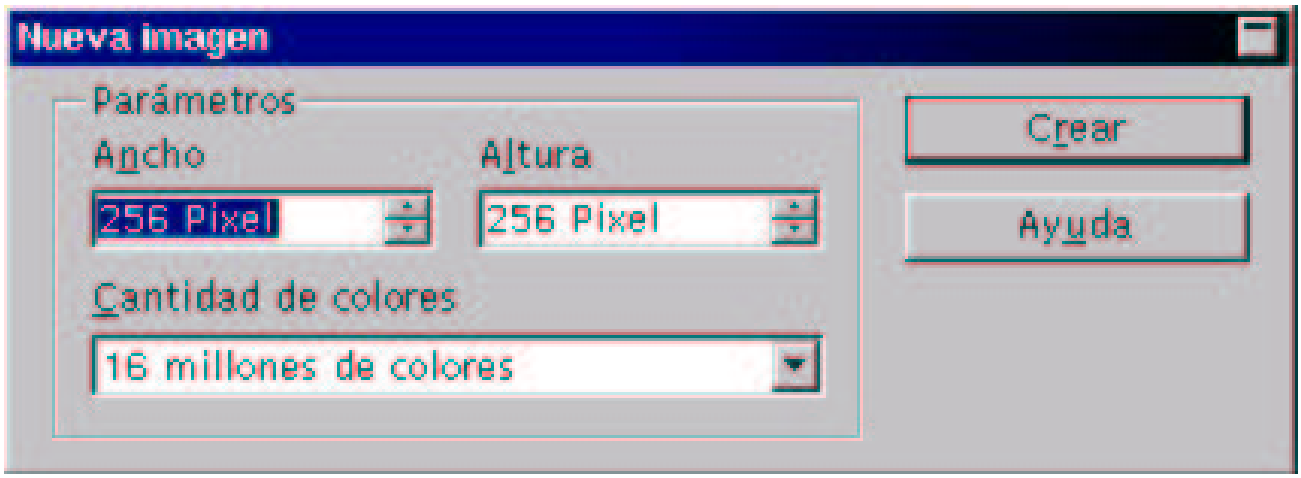


Figura 10.20: Creación de una imagen

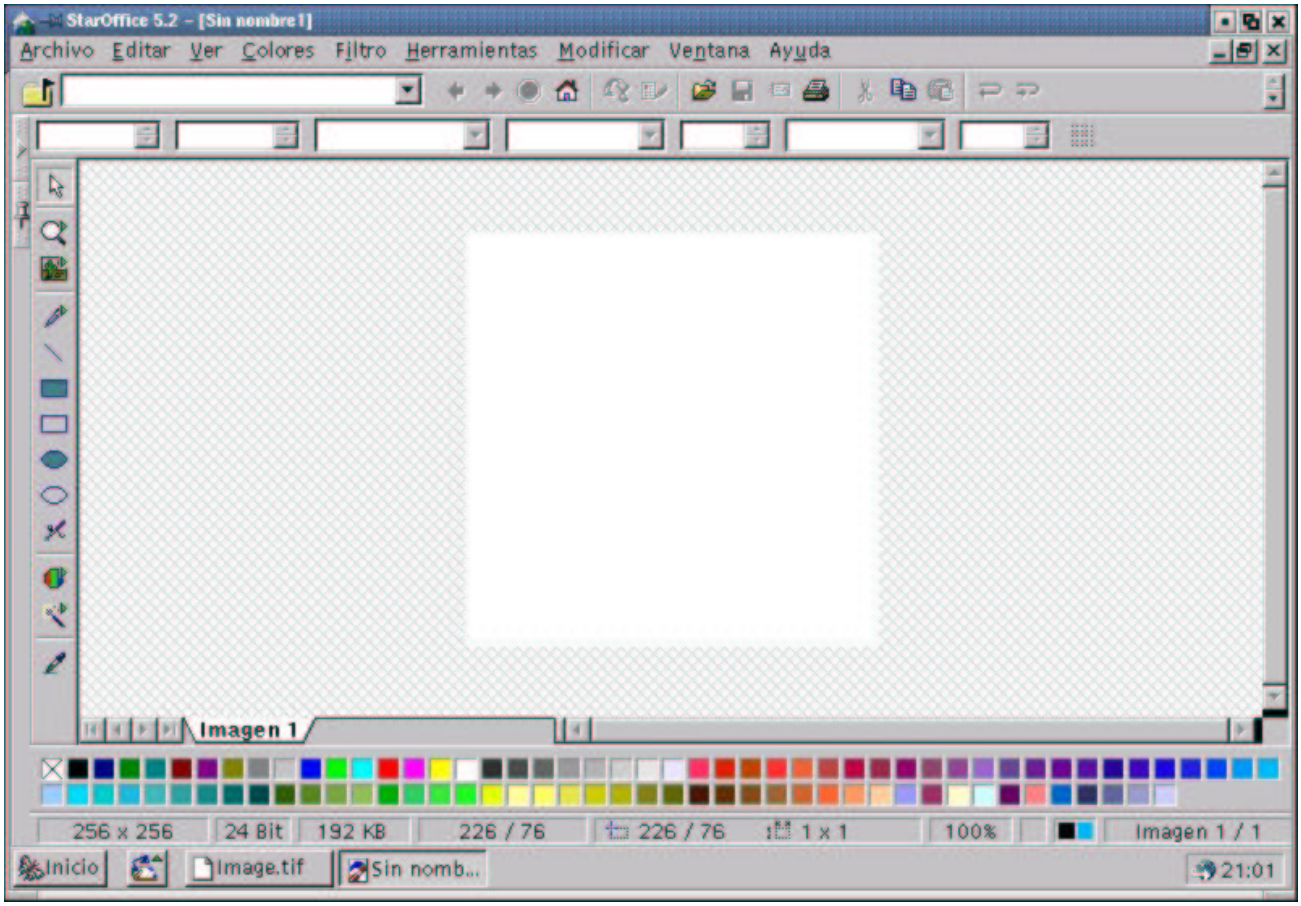


Figura 10.21: Ventana nueva imagen

- La barra de objetos del desktop.
- La zona de trabajo, en la que está la ventana del documento *Sin nombre1*.
- La barra de tareas.
- La ventana del documento.

Si repasamos desde arriba hacia abajo la ventana del documento, nos encontramos:

- La barra de título.
- La barra de objetos de Image.
- La zona de trabajo (donde se prepara la imagen).
- La barra de desplazamiento horizontal, con las pestañas de las imágenes a la izquierda.
- La barra de colores.
- La línea de estado, con información sobre la imagen.

Y si la repasamos de izquierda a derecha, tenemos esto:

- La barra de herramientas.
- La zona de trabajo.
- La barra de desplazamiento vertical.
- Cambio de la profundidad de color.

Para cambiar el número de colores de la imagen se elige en el menú [`Colores`] el submenú [`Modificar profundidad de color`] y se toma una de sus posibilidades, ver figura 10.22.

Si el cambio consiste en cambiar una imagen a color en una imagen en escala de grises, se pueden elegir más opciones tomando en el menú [`Colores`] la opción [`Conversión`] de escalas de grises para abrir el cuadro de diálogo Escala de grises, que también se muestra a la derecha.

10.7.1.4. Cambio de dimensiones

Para cambiar las dimensiones de la imagen se elige en el menú [`Modificar`] la opción [`Modificar`] tamaño, que abre el cuadro de diálogo Modificar tamaño. Esta operación hay que evitarla, si se puede, ya que siempre implica una pérdida de calidad. El programa escala la imagen, y para ello debe añadir o eliminar puntos, cosa que es imposible de realizar conservando todas las características.

10.7.1.5. Escala

Además del cuadro de diálogo Escala, en Image se dispone de la barra de herramientas Escala para modificar el tamaño con que se ve la imagen en pantalla. De las opciones disponibles, la única que muestra la imagen como es exactamente es la opción **1:1**, con la que cada punto de la imagen se representa con un píxel de la pantalla.

10.7.1.6. Manejo de archivos

Image no dispone de un formato propio para almacenar imágenes, de modo que no se pueden proteger con contraseña, como ocurre con los formatos de texto y hoja de cálculo. Image puede grabar y leer la mayoría de los archivos de imagen estándar. Muchos de estos formatos requieren información adicional, que habrá que dar en el momento de grabar.

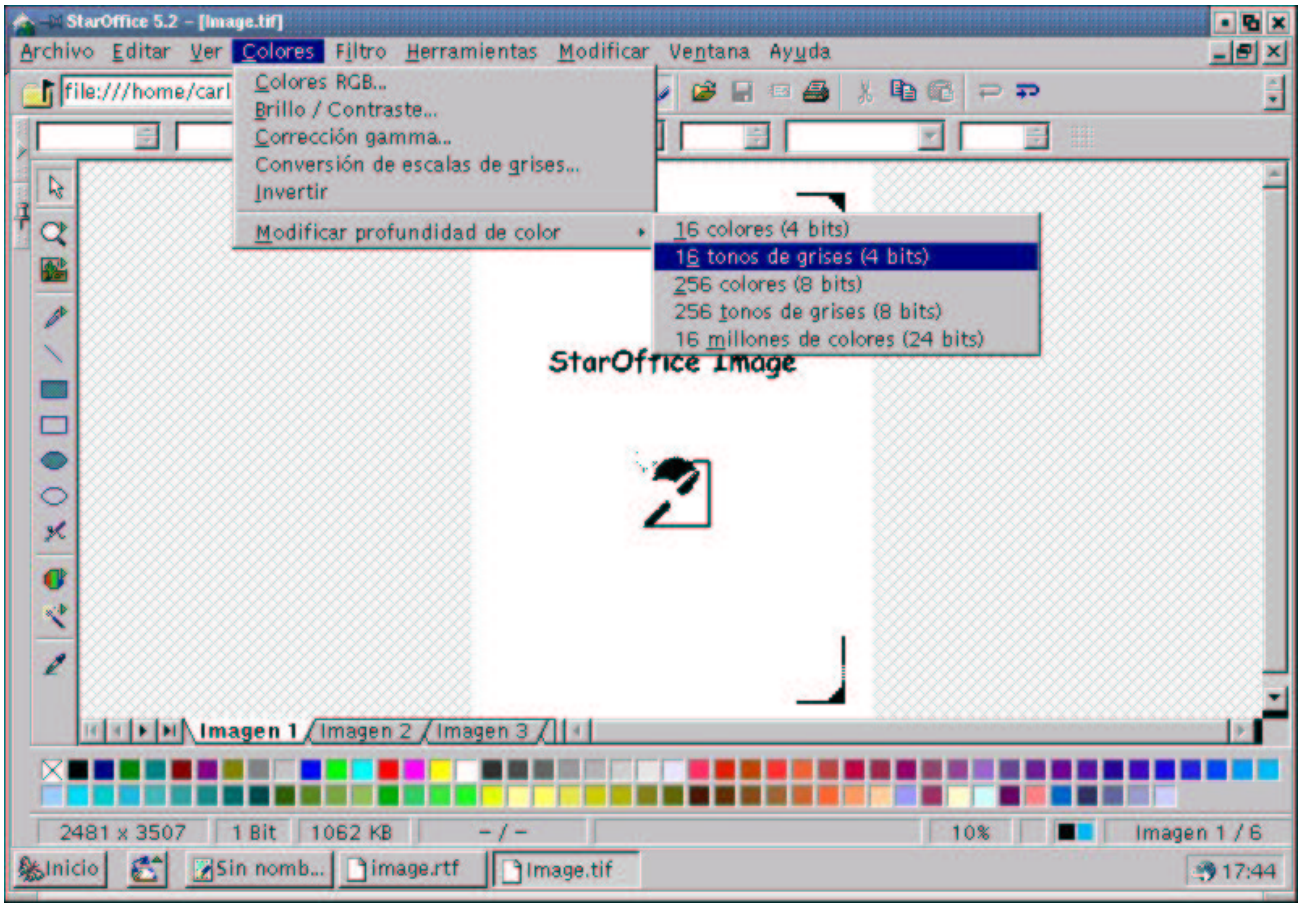


Figura 10.22: Selector de Colores

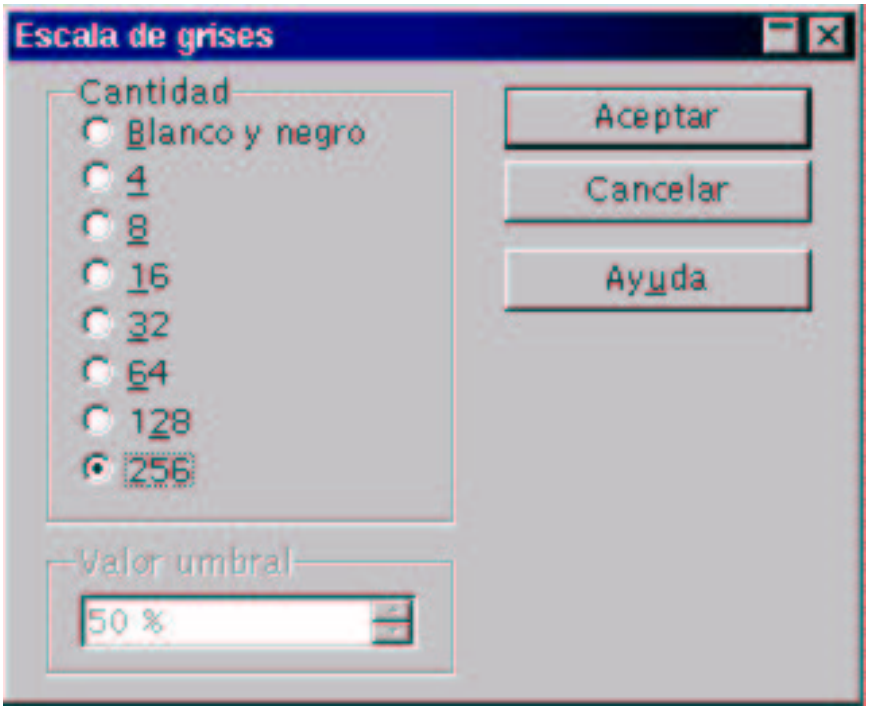


Figura 10.23: Conversión a grises

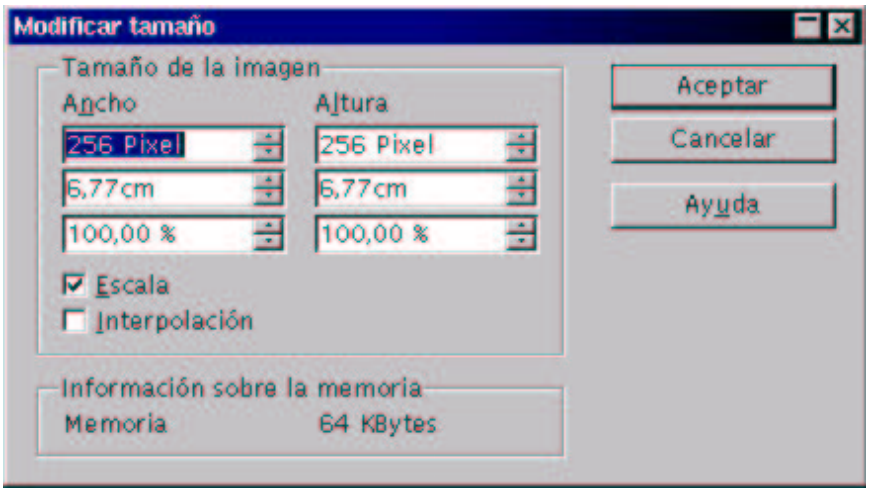


Figura 10.24: Modificar tamaño

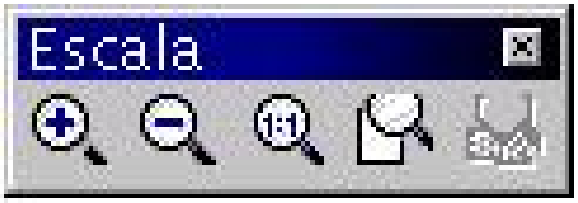


Figura 10.25: Barra de escala

10.7.2. Dibujar

10.7.2.1. Herramientas de dibujo

Image dispone de unas pocas herramientas que permiten dibujar de un modo intuitivo y sencillo. Se explicará brevemente cómo elegir los colores con los que dibujar y el manejo de las herramientas.

10.7.2.2. Los colores

Image maneja dos colores simultáneamente, lo que da mayor flexibilidad al uso de las herramientas. Un color se llama **color de primer plano** y el otro **color de fondo**. Ambos aparecen en la barra de opciones y en la de estado. Se pueden elegir en la barra de opciones (primer plano a la izquierda, fondo a la derecha) y en la de colores (cada uno con un botón del ratón).

10.7.2.3. Pluma

Sirve para dibujar a mano alzada arrastrando el ratón con cualquiera de los botones. La forma de la pluma se puede elegir entre siete posibilidades, en la barra de herramientas Plumas, que se ve a la derecha. La anchura del trazo se elige en la barra de opciones.

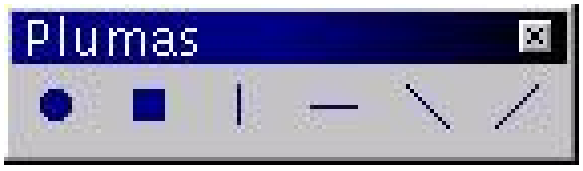


Figura 10.26: Plumas

10.7.2.4. Líneas

Sirve para dibujar un segmento recto. Hay que pulsar en el punto inicial del segmento y arrastrar hasta el punto final, con cualquiera de los botones.

10.7.2.5. Rectángulos y elipses

Sirven para dibujar la figura correspondiente, rellena o no. Se dibujan pulsando y arrastrando el ratón con cualquiera de los botones. La anchura del borde se elige en la barra de opciones.

10.7.2.6. Aerógrafo

Sirve para dibujar con pequeñas manchitas de color; la densidad de las manchitas se regula en la barra de opciones y también con la velocidad de arrastre del ratón; el grosor se elige en la barra de opciones.

10.7.2.7. Reflejos

Para reflejar la imagen horizontal o verticalmente se usa el menú [`Modificar`], submenú [`Reflejar`]. Las opciones también se encuentran en la barra de herramientas Imagen. Si se desea reflejar sólo una parte de la imagen, se puede seleccionar primero con la herramienta de selección.

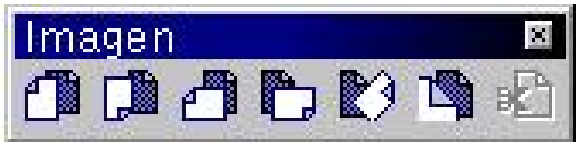


Figura 10.27: Reflejos

10.7.2.8. Rotaciones

Es posible rotar la imagen completa mediante las opciones del menú [`Modificar`], submenú [`Rodar`], que también se encuentran en la barra de herramientas Imagen. Para elegir el ángulo de giro hay que usar la opción `Ángulo libre de rotación` para poder indicar el ángulo en el cuadro de diálogo Ángulo de rotación libre. Hay que saber que el uso de esta herramienta origina que el tamaño de la imagen aumente.

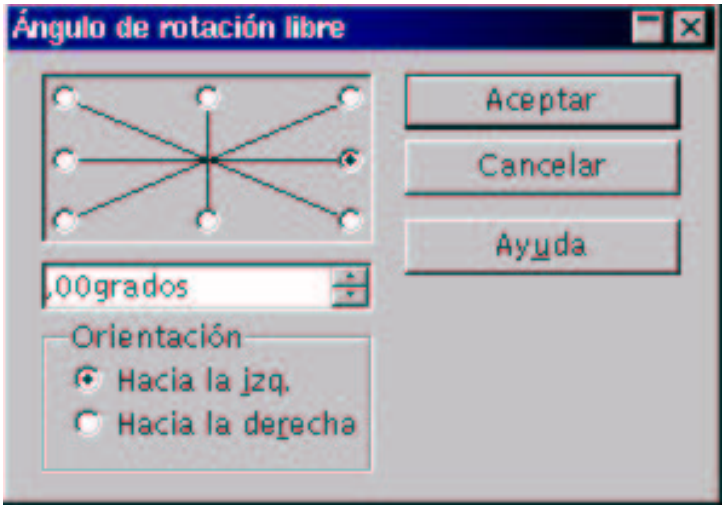


Figura 10.28: Rotación

10.7.2.9. Recortar

Consiste en dejar en la imagen sólo una parte seleccionada, con lo que la imagen disminuye de tamaño. Se selecciona un rectángulo de la imagen y en el menú [`Modificar`] se elige [`Recortar`]. Resulta sorprendente que en este programa el típico modo de trabajo de copiar y pegar no sirva para reproducir una parte de la imagen (cosa que no se puede realizar de ninguna forma) sino que equivale a recortar.

10.7.3. Modificar

10.7.3.1. Selección

Todos los modos de modificación de imagen que se van a ver a continuación se pueden aplicar a toda la imagen o sólo a una parte de ella, según se seleccione o no la parte con la herramienta de selección.

10.7.3.2. Invertir

Para invertir los colores se elige en el menú [`Colores`] la opción [`Invertir`].

10.7.3.3. Brillo y contraste

Para modificar estas características se elige en el menú [`Colores`] la opción [`Brillo/Contraste`], y en el cuadro de diálogo Brillo y contraste se regulan a voluntad.

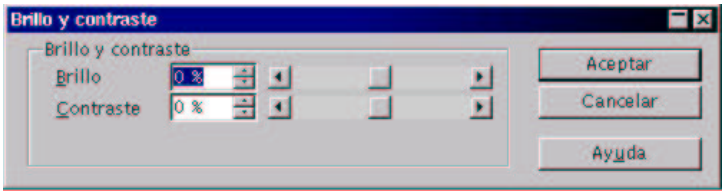


Figura 10.29: Brillo y contraste

10.7.3.4. Valores RGB

Las imágenes que se forman por emisión de luz, como las que se ven en la pantalla de ordenador o la televisión, están formadas por tres componentes: rojo (red, R), verde (green, G) y azul (blue, B). Se puede regular cada componente eligiendo en el menú Colores la opción Valores RGB y modificando ahí sus cantidades.

Figura 10.30: Valores RGB

10.7.3.5. Color

La barra de herramientas Color, permite modificar el brillo, el contraste y los valores RGB, así como cambiar la profundidad de color.

10.7.3.6. Filtros

Los filtros son modos de modificación de la imagen que pueden basarse en cualquier característica: los hay artísticos, que simulan tipos de pintura; técnicos que ayudan a enfocar o desenfocar imágenes; decorativos, para dar más variedad a las imágenes, etc. En Image se pueden aplicar desde el menú Filtro o desde la barra de herramientas Filtros, si bien desde la barra no se pueden ajustar algunos parámetros y desde el menú sí. Ambos se muestran a la derecha. Para conocer sus efectos, lo mejor es utilizar una fotografía e invertir algo de tiempo en ir probando cada filtro.

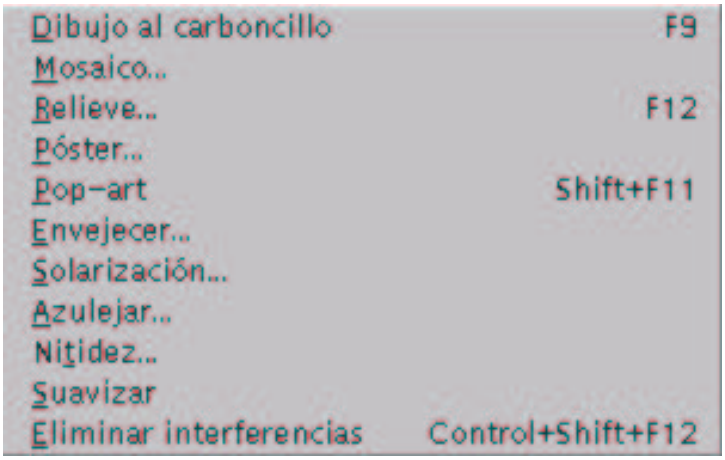


Figura 10.31: Menú de filtros

Figura 10.32: Barra de filtros

TEMA 11

El lenguaje HTML

Los orígenes de HTML se sitúan en la necesidad de proporcionar una estructura a documentos que fueran accesibles a través de la red. Más que definir la apariencia de un documento, HTML describe cómo debería construirse. Con una metáfora visual, HTML le explica cómo se deben encajar seis planchas de madera para crear un cubo, pero no dice nada sobre si la madera debe ser de roble, pino o caoba, o si debe barnizarse o pintarse.

A lo largo del tiempo, la demanda de los usuarios ha motivado la inclusión de elementos relativos a la presentación en las especificaciones de HTML, a pesar de que cada vez se veía más claro que la idea original de HTML era definir estructuras y no presentaciones. Tratemos de explicarlo mejor: la idea original es que cuando hacemos una página web con HTML, deberíamos disponer de dos ficheros: la propia página web con el contenido que nos interesa mostrar y un fichero de estilos. Combinando estos dos ficheros tendremos nuestra página web. Ahora bien, con el tiempo, se han fundido estos dos ficheros en uno sólo creando un gran problema que es de no seguir los estándares que inicialmente se crearon. Y ése es el principal problema de ver páginas webs desde distintos navegadores [1]

11.1. Definición de la estructura de un documento

Todos los documentos HTML están formados por cuatro partes:

1. Una línea declarando qué versión de HTML se ha usado para crear el documento. Esta parte es simplemente esta línea:

   ```
   <!DOCTYPE HTML PUBLIC = "-//W3C//DTD HTML 4.0 Transitional//EN"
   "http://www.w3c.org/TR/REC-html40/loose.DTD">
   ```

2. Un elemento HTML que describe el documento como un documento HTML. Todos los contenidos de un documento HTML, con la excepción de la declaración del tipo de documento, **deben encerrarse entre las etiquetas** <HTML> **y** </HTML>. El resto del documento se encontrará entre las etiquetas de apertura y cierre de HTML.

3. Una sección que declara el encabezado, con las etiquetas <HEAD> y </HEAD>. Esta sección contendrá *metadatos*, que son datos que describen otros datos e incluyen información tal como palabras claves o descripciones cortas que usarán las herramientas de búsqueda de la Red, autor del documento, información del control de versiones y cualquier otro dato que no pueda ser considerado como contenido del documento.

4. El cuerpo principal, donde se encuentra el contenido del documento. Para ello se pueden utilizar las etiquetas <BODY> o <FRAMESET>.

Por tanto, un documento muy básico HTML que contuviera un encabezado con un título tendría el siguiente aspecto:

```
<!DOCTYPE HTML PUBLIC = "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3c.org/TR/REC-html40/loose.DTD">
<HTML>
   <TITLE>
        Esto es un ejemplo en HTML.
   </TITLE>
</HTML>
```

[1] No hay más que recordar cuando leemos "Página optimizada por Internet Explorer" o lo mismo para el Nestcape

11.2. ¿Qué hacemos con esto?

Con este somero ejemplo lo que se debe hacer es guardarlo en un fichero de texto con la extensión `.htm` o `.html` y desde un navegador (Konqueror, Netscape o Galeon, según se prefiera) abrir el fichero con el nombre con que lo hemos guardado. Desde ese momento ya podemos ver resultados.

Cada vez que cambiemos algo, lo guardamos e indicamos al navegador que refresque la información.

11.3. El cuerpo del documento

Como su propio nombre indica, en el cuerpo, del documento se sitúan los contenidos del documento. Es lo que normalmente se considera como el "contenido"del propio documento o el texto de un libro sin pensar en las cubiertas o tapas del libro.

Estos contenidos deben encerrarse entre las etiquetas <BODY> y </BODY>, que deben estar situados **inmediatamente** después de </HEAD>.

```
<!DOCTYPE HTML PUBLIC = "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3c.org/TR/REC-html40/loose.DTD">
<HTML>
   <TITLE>
        Esto es un ejemplo en HTML.
   </TITLE>

   <BODY>
        Aquí ya estamos dentro del propio cuerpo.
   </BODY>
</HTML>
```

Cuando abrimos el cuerpo (<BODY>), podemos establecer unos atributos o propiedades, que pueden ser los siguientes:

ALINK Especifica el color de un enlace cuando se activa.

BACKGROUND="imagen" Especifica un gráfico que se usará como fondo a modo de mosaico para el documento.

BGCOLOR="nombrecolor" Especifica el color de fondo del documento.

LINK="nombrecolor" Especifica el color de los enlaces del documento.

TEXT="nombrecolor" Especifica el color del texto del documento.

VLINK="nombrecolor" Especifica el color de los enlaces del documento que han sido visitados.

Los colores que se pueden usar son:

- Acqua (Agua)
- Black (Negro)
- Blue (Azul)
- Fuchsia (Fucsia)
- Grey (Gris)
- Green (Verde)
- Lime (Verde Lima)
- Maroon (Marrón
- Navy (Azul Marino)

- Olive (Verde oliva)
- Purple (Púrpura)
- Red (Rojo)
- Silver (Plata)
- Teal (Azul Verdoso)
- White (Blanco)
- Yellow (Amarillo)

Por ejemplo,

```
<BODY BGCOLOR="Blue" Text="Yellow" >
        Esto es una página web con letras amarillas y fondo azul.
</BODY>
```

11.4. Estructura del contenido de un documento con encabezados

Casi todos los documentos pueden dividirse en distintos bloques de texto o apoyarse con información adicional como ilustraciones o fotografías. Incluso los documentos HTML más básicos presentarán este comportamiento.

Por ejemplo, puede que su documento tenga dos párrafos, una imagen, una cita y un párrafo final, por lo que podría decirse que su documento consta de cinco bloques diferentes.

Para ello, echamos mano de los encabezados. Éstos constan de seis niveles, desde <H1> hasta <H6>, es decir, de mayor a menor importancia. Lo que entendemos por nivel de importancia no es más que resaltar un texto variando su tamaño. Evidentemente, un texto resaltará con un gran tamaño frente a otro de menor tamaño. Por tanto, con <H1> obtendremos el mayor tamaño y con <H6> el menor.

```
<H1>
        Resaltado de nivel 1
</H1>

<H2>
        Resaltado de nivel 2
</H2>

...

<H6>
        Resaltado de nivel 6
</H6>
```

11.4.1. Formato de texto

Párrafo

El tipo más básico de formatos de texto es la agrupación de frases en párrafos. Para ello se usa el elemento <P>:

```
<P>El tipo más básico de formatos de texto es la agrupación de
frases en párrafos.</P>
```

Énfasis

En casi todos los bloques existe una parte que se le quiere dar una mayor importancia que el resto del texto. HTML marca esta parte con <EM>:

```
Si te digo que no, es que quiero decir <EM>NO</EM>.
```

Énfasis fuerte

Como todo en la vida, hay también grados de acentuar el énfasis en las cosas. Esto lo hacemos con <STRONG>:

```
Si te digo que no, es que quiero decir <STRONG>NO</STRONG>.
```

11.4.2. Listas

Las listas en HTML se dividen en tres categorías básicas:

- *Listas numeradas*
- *Listas no numeradas*
- *Listas de definición*

Las dos primeras son similares. Una *lista numerada* usará los elementos <OL> y </OL> mientras que una *lista no numerada* hará uso de los elementos <UL> y </UL>. Dentro de estos elementos, para marcar cada miembro de la lista, se utiliza la etiqueta común <LI>.

Listas no numeradas

Son aquéllas cuyos elementos no siguen un orden determinado, aunque se muestran en el orden en el que se escriban en el archivo fuente.

Para crear una lista no numerada, se usa la etiqueta <UL> seguido de una colección de elementos con <LI>:

```
<P>Coches que he tenido</P>
  <UL>
     <LI> 1993 - Seat Ronda </LI>
     <LI> 1996 - Volkswagen </LI>
     <LI> 1999 - Audi A4 </LI>
  </UL>
```

Listas numeradas

Son aquéllas cuyos elementos deben seguir un orden determinado. Por ejemplo, los pasos que deben seguir en una receta de cocina o las instrucciones para montar en bicicleta.

Para crear una lista numerada, se usa la etiqueta <OL> seguido de una colección de elementos con <LI>:

```
<P>Cosas que tengo que hacer hoy</P>
  <UL>
     <LI> Mirar el correo </LI>
     <LI> Escribir un poco del libro </LI>
     <LI> Dar una clase </LI>
  </UL>
```

Listas de definiciones

Estas listas se diferencian de las anteriores en que cada entrada de la lista consta de dos partes: el término definido, <DT>, y la propia descripción de la definición, <DD>. La lista comienza con la etiqueta de las listas de definiciones, <DL>. Cada definición debe tener obligatoriamente un término y una descripción, tal como se ve en este ejemplo:

```
<DL>
   <DT>RAM
     <DD>Memoria de acceso aleatorio</DD>
   </DT>

   <DT>BIOS
     <DD>Sistema de entrada y salida básico</DD>
   </DT>
</DL>
```

11.4.3. Tablas

Se utiliza la etiqueta <TABLE>, que acepta siete atributos principales:

SUMMARY: Contiene un texto que describe el contenido y la estructura de la tabla.

WIDTH: Valor que indica la anchura que se le quiere dar a la tabla en proporción con la anchura total de la tabla del navegador (es decir, una valor de 100 indica que la tabla debe ocupar todo el ancho disponible de la ventana, lo cual no equivale al término de "pantalla completa").

BORDER: Determina la anchura del borde de la tabla, medido en píxels. Un valor de 0 (cero) hace que no se ponga borde.

FRAME: También se posible colocar un marco alrededor de la tabla. El valor predeterminado es " *void*"(vacío), lo que significa que no existe ningún marco. Se admiten valores alternativos para indicar cada uno de los lados o una combinación de los lados del marco.

RULES: Son separadores visuales entre filas, columnas o grupos de celdas en el interior de la tabla. De este modo se puede incluir un separador vertical entre las columnas pero ninguno que separe las filas entre sí. El valor predeterminado es "*none*"(ninguno), que significa que no se mostrará ningún separador. Los separadores son elementos visuales distintos a los bordes.

CELLSPACING: Espacio entre las celdas, de la tabla medido en píxels, cuyo valor predeterminado es 0 (cero).

CELLPADDING: Espacio entre de la celda y su contenido, medido también en píxels, cuyo valor predeterminado es 0 (cero).

Lo primero que hacemos es poner una tabla entre las etiquetas <TABLE> Y </TABLE>. Después se va de línea en línea con las etiquetas <TR> Y </TR>. Dentro de ellas, ya usamos las etiquetas para cada celdas de estas líneas que son <TD> y </TD>, que serán tantas como columnas queramos:

```
<TABLE BORDER=1>
        <TH>  ;Si queremos poner una fila a modo de cabecera ...
                <TD>Estamos en la primera celda de la cabecera</TD>
                <TD>Estamos en la segunda celda de la cabecera</TD>
                <TD>Estamos en la tercera celda de la cabecera</TD>
        </TH>
        <TR>  ;Primera fila
                <TD>Estamos en la primera celda de la primera fila</TD>
                <TD>Estamos en la segunda celda de la primera fila</TD>
                <TD>Estamos en la tercera celda de la primera fila</TD>
        </TR>
        <TR>  ;Segunda fila
                <TD>Estamos en la primera celda de la segunda fila</TD>
```

```
                <TD>Estamos en la segunda celda de la segunda fila</TD>
                <TD>Estamos en la tercera celda de la segunda fila</TD>
        </TR>
;Y así sucesivamente ...
</TABLE>
```

En caso de que no queramos rellenar nada en una celda, simplemente ponemos <TD></TD>

11.4.4. Enlaces

Antes que nada, pasemos a definir lo que es un enlace. Todos hemos visto que cuando navegamos por una página web, hay ciertas palabras o gráficos que en el momento de pasar el puntero del ratón se convierte en una mano indicándonos que si pinchamos ahí nos enviará a otra página web. Pues bien, eso es un enlace y en el momento de crearlo dentro de un documento HTML tendrá el siguiente aspecto:

```
<A HREF="http://www.google.com">Este enlace nos lleva a la página del
Google</A>
```

Donde el texto que hemos escrito, "Este enlace..."es el área donde debemos pinchar con el ratón para acceder a esa página. No sólo se puede meter texto, sino también imágenes u otros objetos.

Con esto también podemos hacer un acceso a un correo electrónico:

```
<A HREF="mailto:cila@fmat.ull.es">Podemos probar a enviar un mensaje
a los autores de este libro</A>
```

Con todo lo explicado en esta introducción podemos empezar a hacer nuestros pinitos en el apasionante mundo del lenguaje HTML.

TEMA 12

LyX

LyX, proporciona estupendo tutorial en castellano que son ayuda suficiente para comenzar a escribir apuntes, trabajos, cartas y otros documentos sin necesidad de estudiar nada complicado. Para leer la introducción o el tutorial de LyX basta con elegir Tutorial respectivamente en el menú Ayuda. En el mismo menú se encuentran los restantes capítulos de la documentación de LyX en inglés: Guía del usuario, Características extendidas, Personalización, Manual de referencia y Preguntas de uso frecuente.

Lo que sigue es el resultado de modificar la traducción al castellano del tutorial de LyX, escrito por Sergio García Reus, para adecuarla al estilo de este libro. Aunque esto es suficiente para defenderte con LyX te recomendamos que te leas la traducción original.

12.1. ¡Bienvenido a LyX!

Este fichero ha sido diseñado para todos aquellos que nunca han oído hablar de LaTeX, o no lo conocen muy bien. No tengas miedo, no tendrás que aprender LaTeX para poder usar LyX. Ese es, al fin y al cabo, el punto fuerte de LyX: proporcionar un interfaz casi WYSIWIG (*What You See Is What You Get*) para LaTeX. Sin embargo, hay algunas cosas que necesitarás aprender para usar LyX de forma eficiente.

Probablemente has acabado consultando este documento porque has intentado poner dos espacios después de un punto, o tres líneas en blanco entre dos párrafos. Tras mucha frustración, has comprobado que no se puede. De hecho, descubrirás que la mayoría de los pequeños trucos que estabas acostumbrado a usar con otros procesadores de texto no funcionan en LyX. La razón es que la mayoría de los procesadores de texto que has usado hasta ahora necesitaban que el usuario hiciera a mano todo el espaciado, los cambios de tipo de letra, etc. Así, no sólo se escribía el documento, sino que además se acababa realizando todo el trabajo de formato y composición. LyX realiza este trabajo por tí de forma consistente, dejando que te centres en lo más importante: el contenido del documento.

Así pues, ten paciencia con nosotros y sigue leyendo. Merece la pena que leas este tema.

12.1.1. Qué *es* este tema y qué *no es*

Antes de que empecemos con esta sección, queremos hacer un pequeño apunte. El *Tutorial* usa las convenciones tipográficas señaladas en la página x Si has llegado a este manual primero, ve y lee las convenciones tipográficas. Sí, ahora.

Una vez que ya sabes qué significa cada tipo de letra, vamos a hablar un poco sobre la finalidad de este *Tutorial*.

12.1.1.1. Para aprovechar al máximo el tema

Este *Tutorial* se compone de ejemplos y ejercicios. Para obtener el máximo provecho de este documento, deberías leerlo todo, tecleando todos los pequeños detalles que te vayamos explicando (por simples que sean) e intentando hacer todos los ejercicios para comprobar que lo entiendes.

Si estás familiarizado con LaTeX, probablemente podrás leer el *Tutorial* más deprisa, ya que muchas de las ideas de LyX son realmente ideas de LaTeX disfrazadas. No obstante, LyX no tiene peculiaridades[1] que tengas que aprender. Incluso aunque

[1] o dicho de manera más optimista, "características"

no te apetezca leer el resto del *Tutorial*, harías bien en mirar la sección 12.7, que ha sido escrita específicamente para usuarios experimentados de LaTeX.

La sección 12.1.2 ha quedado sin actualizar de una versión anterior del *Tutorial*, y es un poco general. Aún así, es una buena introducción "panorámica" a LyX, así que deberías echarle un vistazo para ir haciéndote una idea sobre él.

12.1.1.2. Qué *no* vas a encontrar

- Explicación detallada de todas las características de LyX.

  Nuestro objetivo con este tutorial es prepararte para que sólo necesites la *Guía del Usuario*. Tratar de duplicar toda la información sobre las características de LyX aquí sería redundante, demasiado largo y estaría siempre obsoleto. Todo lo que pretendemos es introducir las cosas; como puedes imaginar, hay un "ver *Guía del Usuario*" al final de cada sección.

- Explicación detallada de LaTeX. Para eso tienes el tema 13

Así pues, espíritu intrépido, es hora de seguir adelante. Puedes hacer una breve excursión por la siguiente sección, o puedes continuar con la sección 12.2.

12.1.2. ¿Qué es LyX?

12.1.2.1. Visión general

Parte del reto inicial de usar LyX surge del cambio en la manera de pensar que tú, el usuario, debes hacer. En su momento, todo lo que teníamos para crear documentos eran máquinas de escribir, así que aprendimos verdaderas artimañas para evitar sus limitaciones. Subrayar, que es poco más que sobreescribir con el carácter "_", se convirtió en un forma de resaltar texto. Para crear una tabla, establecías a mano el ancho de cada columna y ponías las tabulaciones necesarias. Lo mismo se aplicaba para cartas y otros textos sangrados a la derecha. Además, la ruptura de palabras al final de línea requería ser muy cuidadoso y previsor.

En otras palabras, todos hemos sido entrenados para preocuparnos por los pequeños detalles de "qué caracter va en qué lugar".

Como consecuencia, casi todos los procesadores de texto se basan en esta mentalidad. Todavía usan tabuladores para añadir espacios en blanco. Todavía te tienes que preocupar de en qué parte exacta de la página saldrá cada cosa. Resaltar texto significa cambiar el tipo de letra, similar a cambiar la rueda de una máquina de escribir.

Aquí es donde LyX difiere de un procesador de texto corriente. No te tienes que preocupar de que una letra vaya en un sitio determinado. Le dices a LyX *lo que estás haciendo* y él se preocupa de todo lo demás, siguiendo un conjunto de reglas llamado estilo. Veamos un pequeño ejemplo.

Supón que estás realizando un informe. Quieres que comience con una sección llamada "Introducción". Así pues, te diriges a cualquiera que sea el menú de tu procesador de texto que cambia el tamaño de fuente y eliges un nuevo tamaño. Después cambias también a negrita. Seguidamente escribes: "1. Introducción". Por supuesto, si más tarde decides que esta sección pertenece a alguna otra parte del documento, o bien insertas una nueva sección anterior a ésta, tienes que cambiarle la numeración a ella y a todas las posteriores, además de las correspondientes entradas en el índice.

En LyX, te diriges a la lista situada a la derecha de todos los botones y eliges Sección, y escribes "Introducción".

Eso es todo. Si cortas y pegas la sección en otra parte, todo se renumera automáticamente. Se puede hacer incluso que LyX actualice cualquier referencia a la sección que esté dentro del fichero.

Con el procesador de texto tradicional hay problemas de consistencia. Cinco días más tarde, abres tu informe y comienzas la sección 4. Sin embargo, has olvidado que estabas usando la letra en negrita de 18 puntos, y usas la de 16, así que acabas escribiendo el encabezado de la sección 4 con un tipo de letra distinto al que usaste para la sección 1. Este problema ni siquiera existe en LyX. El ordenador se encarga de todo el tedioso trabajo de llevar la cuenta de tamaños y fuentes, no tú. Al fin y al cabo, para eso está hecho.

Otro ejemplo. Supón que estás haciendo una lista. En otros procesadores de texto una lista es sólo una mera secuencia de tabuladores y saltos de línea. Necesitas pensar dónde poner la etiqueta de cada elemento de la lista, qué debe ser esa etiqueta, cuántas líneas en blanco hay que poner entre cada elemento, etc. Con LyX, sólo tienes dos preocupaciones: qué clase de lista es, y qué vas a poner en ella. Eso es todo.

Así pues, la idea esencial detrás de LyX es especificar lo que se está haciendo, no cómo hacerlo. En lugar de un procesador "lo que ves es lo que obtienes" (WYSIWIG, What You See Is What You Get), el modelo de LyX es "lo que ves es lo que quieres decir" (WYSIWIM, What You See Is What You Mean).

12.1.2.2. Diferencias entre LyX

He aquí una lista de cosas que no encontrarás en LyX:

- La regla (para medir márgenes)
- Tabuladores
- Espacios en blanco adicionales (i.e. pulsar Intro o Espacio dos o más veces)

Los tabuladores, así como la regla (que te muestra la posición de cada elemento en la página), son inútiles en LyX. El programa se preocupa de dónde tiene que ir cada cosa, no tú. Con los espacios en blanco adicionales ocurre lo mismo; LyX los añade conforme son necesarios, según el contexto. Al principio puede resultar molesto no poder escribir dos líneas en blanco seguidas, pero cobra mucho más sentido una vez que empiezas a pensar en términos WYSIWYM.

Y aquí tienes algunas cosas que presenta LyX pero que no se usan como podrías pensar:

- Controles de sangrado
- Saltos de página
- Espaciado entre líneas (i.e. espaciado simple, doble, etc.)
- Espacio en blanco, horizontal y vertical
- Tipos de letra y tamaño
- Estilo de letra (negrita, cursiva, subrayado, etc.)

Aunque aparecen en LyX, no se necesitan normalmente. El programa se preocupa de estas cosas por tí, actuando en consecuencia según lo que estés haciendo. Diferentes partes del documento son automáticamente puestas en diferente tamaño y estilo. El sangrado de cada párrafo es dependiente del contexto; cada tipo de párrafo se sangra de manera diferente. Los saltos de página se manejan también de forma automática. En general, el espacio entre líneas, entre palabras y entre párrafos es variable, elegido por LyX.[2]

Por último, éstas son las áreas en las que LyX (y LaTeX) sobrepasan a muchos procesadores de texto:

- Separación de palabras a final de línea
- Listas de cualquier tipo
- Matemáticas
- Tablas
- Referencias cruzadas

Por supuesto, muchos procesadores de texto modernos manejan símbolos matemáticos, tablas, separación de palabras a final de línea, e incluso comienzan a aproximarse a las definiciones de estilo y el concepto WYSIWYM. Sin embargo, acaban de empezar a incluir estas características, mientras que LyX está construido sobre el sistema de proceso de documentos LaTeX. Éste lleva más de 10 años con ellas, y *funciona*. Todos los errores han sido subsanados hace tiempo.[3]

[2] Se pueden ajustar todas estas características (sólo el ajuste de unas pocas requiere conocimientos de LaTeX), tanto para todo el documento como para una parte concreta. Ver Guía del Usuario para más detalles.

[3] De acuerdo, nada es perfecto, pero LaTeX es lo más cercano a un programa libre de errores que se puede conseguir.

12.1.3. ¿Qué es LaTeX?

LaTeX es un sistema de preparación de documentos diseñado por Leslie Lamport en 1985.[4] Fue construido gradualmente sobre un lenguaje de composición de documentos llamado TeX, creado por Donald Knuth en 1984. "TeX" se pronuncia como "blech" en inglés[5]. Sin embargo, muchos no comprenden qué es exactamente. TeX toma una secuencia de órdenes de composición escritos en un fichero ASCII, y los ejecuta. Es más complicado que una máquina de escribir, pero sin llegar a la especialización y complejidad de una auténtica imprenta. En cualquier caso, produce como salida un fichero de formato llamado "independiente de dispositivo", o `dvi` para abreviar. El fichero `dvi` puede ser leído depués por otro programa que acepte este formato, o convertido a otros formatos, como PostScript®.

Si no tuviera más característica que ésta, sería un mero motor de composición. Sin embargo, TeX también permite definir macros. Aquí es donde comienza la acción.

La mayoría de la gente que usa TeX está usando realmente un conjunto de macros que Knuth creó para ocultar muchos de los detalles de composición. Esto es en lo que piensa la gente cuando habla de TeX. Los usuarios normales no trabajan con TeX puro, un esqueleto desnudo formado únicamente por comandos de composición. Sólo aquéllos que crean conjuntos de macros lo hacen. Y aquí es donde Leslie Lamport entra en nuestra historia. Él quería macros que fueran más orientadas al usuario y menos a la composición, un conjunto de comandos que sirvieran para componer secciones, tablas o fórmulas matemáticas de una forma consistente y uniforme, sin demasiadas complicaciones. Así nació LaTeX.

Ahora, de manera simultánea al desarrollo y crecimiento de LaTeX, otras personas están creando sus propios paquetes personalizados de macros para TeX, algunos para realizar trabajos en publicaciones matemáticas y cosas así. Unos usaron TeX directamente, otros comenzaron a modificar LaTeX. Para tratar de unificar este lío, un equipo de expertos en LaTeX (incluyendo a Lamport, por supuesto), empezaron a trabajar en LaTeX 2_ε, la versión actual del programa, a finales de los ochenta. Esta nueva versión posee comandos que proporcionan una interfaz más fácil para la creación de macros, ayuda para usar las nuevas fuentes, y más mejoras. De hecho, LaTeX es en sí mismo un vasto lenguaje por derecho propio. Usuarios de todo el mundo han estado creando sus propios añadidos para LaTeX, además de los estándar.

Existen dos formas de extender LaTeX: las clases y los estilos. Una *clase* es un conjunto de macros de LaTeX (y de TeX) que describen un nuevo tipo de documento, como un libro o un artículo. Hay clases para transparencias, para publicaciones de física y matemáticas... ¡algunas universidades tienen incluso una clase para su propio formato de tesis! Un *estilo,* a diferencia de una clase, no define un nuevo tipo de documento, sino un nuevo tipo de *comportamiento*, que puede ser utilizado por cualquier documento. Por ejemplo, LyX controla los márgenes de página y el espaciado entre líneas usando dos ficheros de estilo diferentes de LaTeX, diseñados para este fin. Hay ficheros de estilo para gran cantidad de cosas: imprimir etiquetas o sobres, cambiar el sangrado normal del texto, añadir nuevos tipos de letra, manipular gráficos, diseñar elaborados encabezados de página, personalizar bibliografías, alterar la posición y apariencia de notas a pie de página, tablas y figuras, personalizar listas, etc, etc.

Aquí tienes un resumen:

TeX:	Lenguaje de composición con capacidad para uso y creación de macros.
LaTeX:	Paquete de macros construido sobre TeX.
clases:	Descripciones de un tipo de documento, usando LaTeX.
estilos:	Alteran algún aspecto del comportamiento normal de LaTeX.
LyX:	Procesador de texto visual, WYSIWYM, que usa toda la potencia de LaTeX para realizar el trabajo de composición.

La idea de esta sección ha sido tratar de explicarte *por qué* LyX funciona de manera diferente a otros procesadores de texto. La razón es simple: usa LaTeX como motor de composición. Como este último, se centra en el contexto de tu escritura *(lo que* estás escribiendo). El ordenador se encarga entonces de cómo debe aparecer.

Ah, una cosa más. LaTeX se pronuncia como TeX. Rima con "hey blech".[6] Lamport, sin embargo, dice en su libro que "*lay*-tecks también es posible". Por otro lado, "LyX" se pronuncia como "licks", "lucks" o "looks", dependiendo de la pronunciación de cada país...

[4] La información de esta sección ha sido extraída de "A Guide to LaTeX 2_ε", de Helmut Kopka y Patrick Daly, documento incluido en la bibliografía de la *Guía del Usuario.*

[5] El ruido que se hace cuando se come algo con mal sabor, especialmente los niños pequeños cuando no les gusta la comida. (N. del T.)

[6] Estos detalles de pronunciación no son relevantes para hispanohablantes. TeX, LaTeX y LyX tienen pronunciación directa en español (N. del T.)

12.2. Tu primer documento LyX

Muy bien. Ya estás listo para empezar a escribir. Sin embargo, antes de que lo hagas hay un par de cosas que debemos decir, y que esperamos harán el *Tutorial* más instructivo, útil y divertido.

Como hay mucha información que no te vamos a dar aquí, lo primero que tienes que hacer es encontrar los otros ficheros de ayuda. Afortunadamente, esto es muy simple. Arranca LyX. Elige la *Guía del Usuario* en el menú Ayuda. También puedes cargar el *Tutorial* (si es que no lo estás leyendo ya desde ahí). De esta forma, puedes leerlos mientras escribes tu propio fichero[7]. Ten en cuenta que, una vez que tienes más de un documento abierto, puedes usar el menú Documentos para alternar entre ellos. El *Tutorial* no va a cubrir en detalle aquellos temas que sean tratados en otros manuales de LyX. Esto puede complicarte la vida al principio, pero evita que el *Tutorial* se haga muy extenso. Te acostumbrará también a usar los demás manuales, lo cual —a largo plazo— te ahorará mucho tiempo.

En este *Tutorial*, vamos a asumir que tienes instalada una versión de LyX funcionando perfectamente, así como LaTeX, `xdvi` o cualquier otro visor de ficheros dvi, `dvips` o alguna otra forma de convertir documentos `dvi` a PostScript®, y una impresora. Esto es asumir mucho. Si alguna de estas condiciones no se cumple, tú mismo deberás configurar aquello que falte (o bien tu administrador del sistema). Encontrarás información sobre configuración en otros manuales.

Finalmente, hemos preparado un fichero para que practiques tus habilidades con LyX en él. Se llama `es_ejemplo_sin_lyx.lyx`. Imagina que fue escrito por alguien que no conoce ninguna de las magníficas características de LyX. Conforme vayas aprendiendo nuevas funciones te iremos sugiriendo que corrijas las partes correspondientes del fichero `es_ejemplo_sin_lyx.lyx`. Además, contiene "sutiles" trucos sobre cómo arreglar las cosas[8]. Si quieres hacer trampa (o comprobar lo que has hecho), hay también un fichero llamado `es_ejemplo_con_lyx.lyx` que contiene el mismo texto, pero escrito por un experto en LyX.

Los ficheros de ejemplo se pueden encontrar en el directorio `examples/`, que puedes conseguir seleccionando Archivo ▷ Abrir y pulsando el botón Ejemplos. Abre el documento sin procesar (`es_ejemplo_sin_lyx.lyx`), y usa el menú Archivo ▷ Guardar Como para guardar una copia en tu propio directorio para que puedas trabajar con él. Conforme vayas arreglando el documento, comprueba cómo los cambios afectan a la salida dvi (Archivo ▷ Ver dvi).

Por cierto, el directorio `examples/` contiene muchos otros ficheros de ejemplo que te enseñarán cómo hacer con LyX algunas cosas bastante elaboradas. Son especialmente útiles para mostrar aquello que no cabría en la documentación (por su extensión u otras razones). Después de leer el *Tutorial*, o cuando estés confundido a la hora de hacer algo complicado con LyX, echa un ojeada a estos ficheros.

12.2.1. Escribir, ver e imprimir

- Abre un fichero nuevo con Archivo ▷ Nuevo
- Escribe una frase: `<Este es mi primer documento LyX!`[9]
- Guarda el documento con Archivo ▷ Guardar Como.
- Ejecuta LaTeX para crear un fichero `dvi`, con Archivo ▷ Ver dvi. Puedes ver que se imprimen algunas líneas en la ventana desde la que has ejecutado el comando lyx. Se trata de mensajes de LaTeX, que por ahora puedes ignorar. LyX lanzará el programa `xdvi` (o algún otro visualizador de ficheros `dvi`), que abrirá una nueva ventana mostrándote el aspecto de tu documento cuando esté impreso.[10]
- Imprime con Archivo ▷ Imprimir y pulsa OK.

¡Enhorabuena! Has escrito e impreso tu primer documento LyX. Todo lo demás son detalles, a cubrir por el resto del *Tutorial*, la *Guía del Usuario* y el *Manual de Referencia*.

[7] También pueden servir como buenos ejemplos de uso de las características de LyX.

[8] Los trucos se encuentran en "notas" amarillas. Puedes leerlas pulsado con el ratón sobre ellas.

[9] De acuerdo. Realmente puedes escribir lo que quieras. No importa. Nos disculpamos por la estupidez de esta frase, y de todas las que te vamos a pedir que escribas de aquí en adelante.

[10] Puedes ahorrar tiempo dejando que `xdvi` se ejecute en segundo plano. De esta forma, puedes usar Archivo ▷ Actualizar dvi y simplemente pulsar sobre la ventana de `xdvi` (o restaurarla) cuando LaTeX termine de ejecutarse.

12.2.2. Operaciones sencillas

Por supuesto, LyX puede realizar la mayoría de las cosas a las que estás acostumbrado con tu procesador de texto. Separará las palabras y justificará los párrafos automáticamente. Basta que accedas a un par de menús[11] para que veas cómo la mayor parte de los comandos simples (i.e., Archivo ▷ Salir, Edición ▷ Pegar, Archivo ▷ Imprimir) tienen los nombres que esperas que tengan, están en el menú donde esperas que estén, y funcionan tal y como esperas que funcionen. A continuación tienes una descripción rápida de cómo realizar algunas acciones sencillas.

Deshacer LyX tiene capacidad para "deshacer infinitas veces", lo que significa que puedes deshacer todo lo que lo que hayas hecho desde que empezaste la sesión actual, aplicando una y otra vez Edición ▷ Deshacer. Si deshaces demasiado, elige simplemente Edición ▷ Rehacer para recuperar los cambios. Actualmente, el comando deshacer está *limitado a 100 pasos*. Tampoco funciona *para todo* (por ejemplo, en los cambios de formato de documento).

Cortar/Pegar/Copiar Utiliza Edición ▷ Cortar, Edición ▷ Pegar, y Edición ▷ Copiar para cortar, pegar y copiar. O pega automáticamente el texto seleccionado con el botón central del ratón.

Buscar/Reemplazar Utiliza Edición ▷ Buscar y Reemplazar para realizar una búsqueda sensible a las mayúsculas. En el menú que se despliega a tal efecto, puedes desplazarte hacia delante y hacia atrás en la búsqueda mediante las flechas, y reemplazar aquellas palabras que hayas encontrado con el botón Reemplazar.[12]

Formato de caracteres Puedes *resaltar* texto (lo que normalmente significa poner los caracteres en cursiva), ponerlo en **negrita**, o en ESTILO NOMBRE (habitualmente en minúsculas, para nombres propios de personas) desde los botones interruptor en el menú Formato.

Barra de herramientas Sus botones (justo debajo de los menús) te permiten realizar las funciones más usuales, como Pegar e Imprimir. Si mantienes el cursor del ratón sobre alguno de los botones de la barra, una pequeña nota amarilla te informará sobre la función concreta del botón.

Minibuffer La franja gris en la parte de abajo de la ventana de LyX recibe el nombre de *minibuffer*. Se encarga de mostrarte toda clase de información útil. Por ejemplo, cuando guardas, te dice el nombre del fichero que acabas de guardar. También muestra algunos mensajes de error. Observa que puedes escribir en él. Esto te ofrece una gran funcionalidad, incluyendo la posibilidad de estropear el documento. En otras palabras, no escribas en el *minibuffer* a menos que sepas lo que estás haciendo.

Por supuesto, todavía no has escrito suficiente para encontrar útiles todas estas funciones. Conforme vayas escribiendo más, prueba a deshacer, copiar, pegar, etc.

12.2.3. WYSIWYM: el espacio en blanco en LyX

Una de las cosas más difíciles para los nuevos usuarios es acostumbrarse a la forma en que LyX maneja el espacio en blanco. Por mucho que pulses Retorno de carro, sólo conseguirás una única línea en blanco. Por mucho que pulses la Barra espaciadora, sólo conseguirás un único espacio en blanco. En una línea vacía LyX no te dejará poner ni siquiera un espacio. El Tabulador no te adelantará ningún espacio, ¡de hecho no hay tabulación! Tampoco hay ninguna regla en la parte superior de la página que te permita definir tabulaciones o márgenes.

Muchos procesadores de texto comerciales están basados en el principio WYSIWYG: "lo que ves es lo que obtienes". LyX, por el contrario, está basado en el principio "lo que ves es lo que quieres decir". Escribes lo que quieres decir, y LyX se preocupará de la composición por tí para que el resultado final quede bien. Un Retorno de carro gramaticalmente separa párrafos, y de la misma forma un espacio separa palabras, así que no hay ninguna razón para poner varios seguidos; un Tabulador no tiene función gramatical alguna, así que LyX no los usa. Con LyX emplearás más tiempo en el contenido del documento, y menos en la forma. Ver Sección 12.1.2 para más información sobre el concepto WYSIWYM.

LyX tiene (muchas) formas de ajustar al detalle el formato del documento. Después de todo, podría no imprimirse exactamente lo que querías decir. La *Guía del Usuario* contiene información sobre eso. Incluye espaciado vertical y horizontal —mucho más potentes y versátiles que múltiples espacios o líneas en blanco— y formas de cambiar tamaño y estilo de

[11] Si eres como tantos usuarios de UNIX, lo habrás hecho ya mucho antes de empezar a leer el *Tutorial*.

[12] Cierra la ventana cuando hayas acabado, o déjala abierta si lo encuentras más conveniente. La mayoría de los menús contextuales de LyX (incluyendo los de Buscar y Reemplazar, Índice General y Formato, así como los de matemáticas) son ventanas que pueden ser apartadas, en vez de cerradas. Unos pocos menús como Archivo ▷ Abrir, no te dejarán escribir nada en la ventana principal hasta que los cierres. Asegúrate de que el foco está en la ventana correcta cuando estés tratando escribir en la ventana principal de LyX o introduciendo un comando en alguna ventana de diálogo.

letra y alineación de párrafos a mano. La idea es que puedas escribir todo el documento concentrándote en el contenido, y solamente te preocupes de ajustar los detalles al final. Con los procesadores de texto convencionales, el formato te distrae continuamente.

12.3. Entornos

Las diferentes partes de un documento tienen propósitos diferentes; llamaremos a estas partes *entornos*. La mayor parte del documento está formada por texto normal. Los títulos de sección (capítulos, subsecciones, etc.) permiten al lector saber que se va a tratar un nuevo concepto o idea. Ciertos tipos de documentos tienen entornos especiales. Un artítculo de periódico tendrá un resumen y un título. Una carta no tendrá nada de eso, pero probablemente contendrá un entorno para la dirección del remitente.

Los entornos son una parte importante en la filosofía "lo que ves es lo que quieres decir" de LyX. Un entorno dado puede requerir un cierto estilo o tamaño de letra, sangrado, espaciado y otras características. Este problema se agrava cuando el formato exacto de un entorno puede cambiar: un periódico puede usar letra en negrita, de 18 puntos con párrafos centrados para los títulos, mientras que otros pueden usar párrafos justificados con letra cursiva de 15 puntos; idiomas distintos pueden tener diferentes convenios para el sangrado; y los formatos de bibliografía pueden variar ampliamente. LyX te evita tener que aprender todos los diferentes estilos de formato.

La caja de Entorno se sitúa al final a la izquierda en la barra de herramientas (justo debajo del menú Archivo). Indica qué entorno estás usando en cada momento. Mientras escribías tu primer documento, decía "Standard" (normal), que es el entorno por defecto para texto. Ahora vas a usar varios entornos en el nuevo documento para que puedas ver cómo funcionan. Lo harás con el menú Entorno, que puedes abrir pulsando sobre el icono con "la flecha hacia abajo" justo a la derecha de la caja de Entorno.

12.3.1. Secciones y subsecciones

Escribe la palabra `Introducción` en la primera línea de tu fichero LyX, y elige Section (sección) en el menú de entorno[13]. LyX numera la sección como "1" y escribe el encabezado (título) en un tipo de letra mayor (por supuesto, el encabezado aparecerá de esta forma en el fichero `dvi` y en el documento impreso). Ahora pulsa Retorno de carro. Observa que la caja de entorno cambia de "Section" a "Standard". Se asume que los títulos de sección, como muchos entornos, terminan cuando introduces un Retorno de carro[14]. Escribe la introducción del documento:

```
Esto es una introducción a mi primer documento LyX.
```

Pulsa Retorno de carro otra vez, y elige de nuevo Section en el menú entorno. LyX escribe un "2" y espera que intoduzcas un título. Escribe `Más cosas`, y verás que de nuevo lo establece como título de sección.

La cosa mejora. Ve al final de la sección 1 otra vez (tras "mi primer documento LyX"), pulsa Retorno de carro, y selecciona Section en el menú Entorno. Una vez más, LyX escribe "2" y espera a que introduzcas un título. Escribe `Acerca de este documento`. La sección "Más cosas", que antes era la sección 2, ¡ha sido automáticamente renumerada a sección 3! De una forma verdaderamente WYSIWYM, sólo necesitas identificar los títulos de las distintas partes de que se compone el texto, y LyX se encarga de la numeración de secciones y su formato.

Pulsa Retorno de carro para volver al entorno Standard y escribe las siguientes cinco líneas:

```
Secciones y subsecciones se describen más adelante.
Descripción de sección
Las secciones son mayores que las subsecciones.
Descripción de subsección
Las subsecciones son menores que las secciones.
```

[13] No tienes que seleccionar la línea. Si no hay nada seleccionado, LyX cambia el párrafo en el que estás escribiendo ahora al entorno elegido. Alternativamente, puedes cambiar varios párrafos seleccionándolos antes de elegir el nuevo entorno.

[14] Ver *Guía del Usuario* para poder escribir títulos de más de una línea. Desde luego, el entorno Standard puede continuar a lo largo de varios párrafos. Los entornos de listas (ver más adelante) tampoco terminan con Retorno de carro. Siempre puedes saber el entorno en el que estás mirando la caja de Entorno.

Colócate en la segunda línea y elige Subsection (subsección) en el menú Entorno. LyX numera la sección como "2.1", y la escribe con un tamaño de letra mayor que el de texto regular pero menor que el de un título de sección. Cambia también el entorno de la cuarta línea a Subsection. Como probablemente esperabas, LyX la numera automáticamente como sección "2.2". Si pones una sección más antes de la sección 2, ésta será renumerada como sección 3, y sus subsecciones corespondientes como "3.1" y "3.2".

Niveles más profundos de sección son la subsubsección (Subsubsection), el párrafo (Paragraph), y el subpárrafo (Subparagraph). Dejaremos que juegues tú mismo con ellos. Notarás que los títulos de párrafo y de subpárrafo no están numerados por defecto, y que los subpárrafos están sangrados; ver *Guía del Usuario* para cambiar esto. Los encabezados de capítulo (Chapter) son realmente el nivel más alto de la jerarquía, por encima de las secciones, pero solamente se pueden utilizar en ciertos tipos (clases) de documentos (ver Sección 12.4.1).

Finalmente, puede que quieras usar secciones y subsecciones sin numerar. Existen entornos para esto también. Si cambias uno de los encabezados de sección al entorno Section* (puede que tengas que bajar en el menú Entorno para encontralo), LyX usará el mismo tamaño de letra que en las secciones normales, pero no la numerará. También están los entornos "no numerados" correspondientes a Subsection y Subsubsection. Intenta cambiar alguna de tus secciones o subsecciones a entornos no numerados, y comprueba cómo los demás números de sección se actualizan.

Ejercicio: Arregla los encabezados de sección y subsección del fichero `es_ejemplo_sin_lyx.lyx`.

12.3.2. Listas y sublistas

LyX tiene diferentes entornos para componer listas. Los variados entornos de listas te evitan tener que pulsar el Tabulador un millón de veces cuando estás escribiendo un esquema, o de renumerar toda la lista cuando quieres añadir un nuevo punto en mitad de ella. Así puedes concentrarte en el contenido de la lista[15]. Distintos tipos de documentos requieren, lógicamente, entornos de lista diferentes:

- Una exposición de diapositivas podría usar las listas simples (etiquetadas con bolos) del entorno Itemize para describir los diferentes puntos.
- Un esquema usaría las listas numeradas (y sublistas etiquetadas con letras) del entorno Enumerate (enumeración).
- Un documento que describa varios paquetes de software usaría el entorno Description (descripción), en el que cada elemento de la lista comienza con una palabra en negrita.
- El entorno List (que no existe en LaTeX) es ligeramente diferente al entorno Description.

Vamos a escribir una lista de razones por las que LyX es mejor que otros procesadores de texto. En cualquier parte de tu documento escribe:

```
LyX es mejor que otros procesadores de texto porque:
```

y pulsa Retorno de carro. Ahora elige Itemize en el menú de Entorno. LyX pondrá un bolo (realmente un asterisco, que se convertirá en un círculo en la salida final) en la línea. Introduce tus razones:

```
LyX realiza la composición por tí.
Las matemáticas son WYSIWYG
Las listas son muy fáciles de crear
```

Los entornos de listas, al contrario que los encabezados, no terminan cuando introduces un Retorno de carro. En vez de eso, LyX asume que vas a introducir el siguiente elemento de la lista. La anterior resultará ser una lista de tres elementos. Si deseas más de un párrafo en un solo *elemento* de la lista, una forma de conseguirlo es mediante un Retorno de carro protegido, pulsando C-Retorno de carro. Para salir de la lista tienes que volver a seleccionar el entorno Standard (o usar la combinación de teclas M-p s).

Has conseguido una bonita lista simple. Puedes ejecutar LaTeX para ver cómo aparece en la salida impresa. Pero, ¿y si querías numerar las razones? Bien, simplemente selecciona toda la lista[16] y elige el entorno Enumerate en el menú. ¡Increíble! Como ya dijimos, si añades o borras un elemento de la lista, LyX arregla la numeración.

[15] Sí, estamos recalcando una y otra vez este punto a lo largo de todo el *Tutorial*. Pero *es* la principal filosofía de LyX, así que por favor, discúlpanos.

[16] LyX no te dejará seleccionar el primer bolo a menos que selecciones el párrafo anterior a la lista, que probablemente no es lo que quieres hacer. De forma similar, no puedes seleccionar el número en el título de una sección. No te preocupes de eso.

Mientras la lista esté selecionada, puedes cambiar a los otros dos entornos, Description y List, para ver cómo son. Para ambos, cada elemento de la lista está compuesto por un término, que es la primera palabra del elemento, seguido de una definición, que es el resto del párrafo (hasta que pulses Retorno de carro). El término se escribe en negrita (Description) o separado por un "Tabulador"[17](List) del resto del párrafo. Si quieres más de una palabra en el término, separa las palabras con Espacios protegidos, que se obtienen al pulsar C-Espacio y aparecen como pequeñas "ues" rosas.

Ejercicio: Escribe correctamente la lista en el fichero `es_ejemplo_sin_lyx.lyx`

Puedes anidar listas unas dentro de otras en toda clase de formas interesantes. Un ejemplo sería escribir esquemas. Las listas simples y numeradas tendrán diferentes tipos de bolos y diferente numeración en las sublistas. Ver *Guía del Usuario* para los detalles de los distintos tipos de listas, así como ejemplos de cómo usar *mucho* anidamiento.

12.3.3. Más entornos: estrofas, citas y otros

Hay dos entornos para separar las citas del texto que las rodea: Quote para citas cortas y Quotation para las más largas. El código de ordenador (el entorno LyX-Code, usado también en este *Tutorial* para ejemplos largos) se escribe en letra de `máquna de escribir`; este entorno es el único sitio en LyX donde se permite usar varios espacios seguidos para permitir el sangrado del código. Puedes incluso escribir poesía mediante el entorno Verse (estrofa), usando Retornos de carro para separar los versos, y C-Retorno de carro para separar líneas dentro de un verso. Ver Guía del Usuario para una descripción más completa de todos los entornos disponibles en LyX.

Ejercicio: Usa los entornos Quote, LyX-Code, y Verse donde corresponda en el fichero `es_ejemplo_sin_lyx.lyx`

12.4. Escribiendo documentos

Esperamos que el capítulo anterior te haya servido para acostumbrarte a esribir con LyX. Te hemos presentado las operaciones básicas de edición, así como el potente método de escribir con entornos. Sin embargo, la mayoría de la gente que usa LyX querrá escribir documentos: periódicos, artículos, libros, manuales o cartas. Este capítulo pretende que pases de escribir simple texto a escribir un documento completo. Te presentará las clases de texto, que te permiten crear distintos tipos de documentos, y te describirá muchas características nuevas que convierten texto en un documento, como títulos, notas a pie de página, referencias cruzadas, bibliografías e índices.

12.4.1. Clases de texto y modelos

Diferentes tipos de documentos deben componerse de forma diferente. Por ejemplo, normalmente los libros se imprimen a doble cara, mientras que los artículos se imprimen a simple. Además, muchos documentos contienen entornos especiales: las cartas tienen entornos (como la dirección del remitente o la firma) que no tienen sentido en un libro o un artículo. Las *clases de texto* de LyX[18] se encargan de estas grandes diferencias entre cada tipo de documento. Este Tutorial, por ejemplo, se ha escrito con la clase de texto Book (libro). Estas clases son otro de los grandes pilares de la filosofía WYSIWYM; le dicen a LyX cómo tiene que componer el documento, así que tú no necesitas saberlo.

Tu documento se está escribiendo seguramente con la clase Article (artículo)[19]. Prueba a cambiar a otras clases (usa el menú Clase dentro de Formato▷Documento) para ver cómo se compone cada una de ellas. Si la cambias a Book y miras en el menú Entorno, verás que la mayoría de entornos permitidos son los mismos. Sin embargo, ahora puedes utilizar el entorno Chapter (capítulo). Si alguna vez no estás seguro de cuáles tienes disponibles en una clase de texto dada, sólo tienes que consultar el menú Entorno.

El tamaño de letra, la impresión a una o dos columnas, o los encabezados de página son sólo algunas de las cosas en las que difiere el formato de composición de los distintos periódicos. Conforme la Era Digital ha ido madurando, éstos han empezado a aceptar presentaciones electrónicas, creando "ficheros de estilo" LaTeX para que los autores puedan enviar sus artículos correctamente maquetados. LyX también está preparado para esto. Así por ejemplo, ofrece soporte para composición (y entornos adicionales) para los periódicos de la Sociedad Americana de Matemáticas mediante la clase de texto Article (AMS).

A continuación te damos una breve referencia rápida de algunas clases de texto. Como siempre, dirígete a la *Guía del Usuario* para más detalles.

[17] Pero un tabulador de composición, que cambiará para ajustarse al tamaño del mayor término de la lista, no un rígido, inmutable y patético Tabulador de máquina de escribir.

[18] Para usuarios de LaTeX: equivalente a las clases de documento de LaTeX (*documentclass*).

[19] Normalmente el artículo es la clase de texto por defecto, aunque puedes establecerla en tu fichero de configuración `lyxrc`.

Nombre	Comentarios
article	artículo — simple cara, sin capítulos
article (AMS)	formato y entornos de la Sociedad Americana de Matemáticas
report	informe — más extenso que el artículo, doble cara
book	libro — informe + portada y contraportada
slides	transparencias (incluyendo FoilTEX)
letter	carta — entornos adicionales para la dirección, la firma...

12.4.2. Modelos: escribir una carta

Una de las clases de texto más populares es la carta. Una forma de escribir una carta sería abrir un Nuevo fichero, y elegir Letter en el menú Clase dentro de Formato ▷ Documento. Aunque esta es la manera más obvia de hacerlo, supone trabajo de más. Cada vez que escribes una carta de negocios pones tu dirección, la del destinatario, el cuerpo, la firma, etc. Por tanto, LyX ofrece un *modelo* para cartas, que contiene un ejemplo de carta; una vez que tienes el modelo, sólo tienes que sustituir un par de cosas cada vez que quieras escribir una nueva carta.

Abre un archivo nuevo con Archivo ▷ Nuevo basado en Modelo. Tras decidir un nombre para el nuevo fichero, elige `latex_letter.lyx` en el menú Seleccionar Modelo. Guarda e imprime el fichero para ver cómo se componen los distintos entornos.

Si te fijas en el menú Entorno, verás algunos entornos, como My Address (dirección del remitente), que no están disponibles en otras clases. Otros, como Quote y Description, son familiares. Puedes jugar con ellos para ver cómo funcionan. Comprobarás, por ejemplo, que en el entorno Signature (firma) la palabra "Signature:" en rojo antecede al texto de la firma. Esta palabra no se muestra en la verdadera carta, como podrás ver si imprimes el fichero. Sólo está ahí para que sepas dónde va la firma. Ten en cuenta también que no importa dónde esté situada la línea Signature. Recuerda que LyX es WYSIWYM, así que puedes poner el entorno Signature en el lugar que quieras, él sabe que en la salida impresa la firma debe ir al final.

Un modelo es simplemente un fichero de LyX. Esto quiere decir que puedes completarlo con tu dirección y tu firma y guardarlo como un nuevo modelo. A partir de ahora, siempre que quieras escribir una carta ahorrarás tiempo usando tu nueva plantilla. Probablemente no necesitamos sugerirte que hagas un verdadero "ejercicio", ¡escribe una carta a alguien![20]

Los modelos pueden ahorrar muchísimo tiempo, así que te instamos a que los uses siempre que puedas. Además, te pueden ayudar a usar algunas de las clases de texto más elaboradas y complejas. Finalmente, pueden ser útiles para alguien que quiera configurar LyX para un grupo de usuarios poco experimentados con los ordenadores. A la hora de empezar a trabajar con LyX, resulta mucho menos intimidatorio tener, por ejemplo, un modelo de carta personalizado para tu empresa.

12.4.3. Título del documento

LyX (al igual que LaTeX) considera el título —que puede incluir el título propiamente dicho, el autor, la fecha e incluso el resumen del documento— como una parte independiente.

Vuelve a tu documento `nuevo-archivo.lyx` y asegúrate de que está usando la clase de texto Article.[21] Escribe un título en la primera línea, y cámbiala al entorno Title (título). En la siguiente línea escribe tu nombre y cámbiala a entorno Author. En la siguiente escribe la fecha con el entorno Date (fecha). Escribe un párrafo o dos resumiendo el contenido de tu documento y usa el entorno Abstract (resumen). Ahora mira cómo queda una vez impreso.

Ejercicio: Arregla el título, la fecha y el autor en `es_ejemplo_sin_lyx.lyx`

12.4.4. Etiquetas y referencias cruzadas

Puedes etiquetar una sección de tu documento (o una subsección, o incluso, con menos frecuencia, un fragmento de texto cualquiera). Una vez que lo hagas puedes hacer referencia a esta sección desde otras partes del documento mediante referencias cruzadas. Puedes referirte al número de sección o bien a la página donde aparece. Como sucedía con las secciones y las notas a pie de página, el propio LyX se encarga también de las referencias. La gestión automática de etiquetas y referencias cruzadas es una de las mayores ventajas de LyX (y LaTeX) sobre los procesadores de texto convencionales.

[20] Una advertencia si estás escribiendo a partir de un modelo. Si borras todo el texto de un entorno —por ejemplo, si borras todo el campo My Address para poder poner tu propia dirección— y entonces mueves el cursor sin escribir nada, el entorno podría desaparecer. Esto se debe a que muchos entornos no pueden existir sin contener texto alguno. Para restituirlo, solamente tienes que volver a seleccionarlo en el menú Entorno.

[21] No debes usar la clase carta, ya que ésta no permite títulos.

Tu primera etiqueta

Vamos a marcar nuestra segunda sección, cuyo título es "Acerca de este documento". Pincha al final de la línea del título, y selecciona en el menú Insertar ▷ Etiqueta. Se deplegará una ventana de diálogo preguntándote el nombre de la etiqueta. Escribe `sec:acercadeldocumento`, que parece una etiqueta suficientemente descriptiva para evitar confusiones con otras que podamos añadir más adelante[22]. Cuando pulses el botón OK, el nombre de la etiqueta se situará en un recuadro cerca del título de la sección.

Por cierto, también puedes colocar la etiqueta en cualquier lugar de la sección; las referencias a sección se referirán a la última cuyo encabezado vaya antes que la etiqueta. Sin embargo, ponerla en la misma línea que el título (o bien en la primera línea de texto) asegura que las referencias a página se refieran al comienzo de la sección.

Hasta ahora no hemos hecho nada (el fichero `dvi` tiene el mismo aspecto, ya que las etiquetas no se muestran en el documento impreso). Pero has añadido una, y ahora puedes hacer referencia a ella. Vamos a hacerlo a continuación.

Tu primera referencia

Sitúa el cursor en algún lugar de la sección 2 de tu documento. Escribe

```
Si quieres saber más acerca de este documento, ve a mirar \\
la sección , que se puede encontrar en la página .
```

Ahora —con el cursor tras la palabra "sección"— elige Insertar ▷ Referencia Cruzada. Aparecerá la ventana Insertar referencia. Ésta muestra una lista de etiquetas posibles a las que puedes hacer referencia. Por el momento sólo hay una, "sec:acercadeldocumento". Selecciónala (estará seleccionada por defecto) y pulsa el botón Insertar referencia. Ahora sitúa el cursor tras la palabra "página", y pulsa el botón Insertar número de página en la misma ventana.

LyX coloca las referencias dentro de un recuadro en el lugar donde estaba el cursor. En el documento impreso, cada marcador de referencia será reemplazado por el número de página o de sección (según lo que hayas elegido en el menú Insertar referencia). De manera muy práctica, las referencias actúan como enlaces hipertexto cuando estás editando el documento con LyX; pinchar sobre una de ellas moverá el cursor hasta la etiqueta referenciada. Utiliza Archivo ▷ Actualizar dvi, y verás como en la última página hacemos referencia a la "sección 2" y la "página 1" (o cualquiera que sea la página en donde aparezca el título de la sección 2).

Más diversión con etiquetas

Te hemos dicho que LyX se encarga él solo de la numeración de las referencias cruzadas; ahora podrás comprobarlo. Añade una nueva sección antes de la sección 2. Ahora vuelve a ejecutar LaTeX, y —voilà!— la referencia ha cambiado a la sección 3. Convierte "Acerca de este documento" en una subsección, y la referencia indicará ahora la subsección 2.1 en lugar de la sección 3. Por supuesto, la referencia de página no cambiará a menos que añadas una página entera antes de la etiqueta.

Si quieres practicar algo más con las etiquetas, prueba a poner otra, "sec:miprimeraetiqueta", donde estaba la primera referencia cruzada, y haz referencia a aquélla desde cualquier parte del documento. Si vas a usar las referencias a menudo (si, por ejemplo, estás escribiendo un artículo de prensa), puede ser conveniente que dejes la ventana Insertar referencia abierta.

Si quieres asegurarte de que las referencias a página se mantienen correctas incluso en documentos extensos, usa Copiar para llevar un par de páginas de la Guía del Usuario al portapapeles, y usa Pegar para inserrtar el texto robado en tu documento[23].

Ejercicio: Arregla las referencias en el fichero `es_ejemplo_sin_lyx.lyx`

12.4.5. Notas a pie de página y notas al margen

Las notas a pie de página se pueden añadir usando el botón Insertar Nota a pie en la barra de herramientas[24] o bien accediendo en el menú a Insertar ▷ Nota al pie. Pincha al final de la palabra "LyX" en cualquier parte de tu documento y pulsa el botón Insertar Nota a pie. Una línea de pie de página se abrirá debajo de la línea en la que estabas escribiendo. En el extremo izquierdo verás la palabra "*foot*" (pie) escrita en rojo sobre fondo gris. El resto de la línea está enmarcada en rojo; aquí es donde escribirás la nota. LyX sitúa el cursor al principio de la línea. Escribe:

[22] Escribimos "sec:" porque también podemos etiquetar ecuaciones, tablas y figuras.

[23] Por cierto, copiar un capítulo causará un error de LyX, ya que los capítulos no se permiten en la clase artículo. Si te sucede, borra simplemente el título de capítulo. Si quieres saber por qué ocurre esto, ve a la sección 12.4.1.

[24] El botón muestra una flecha señalando texto en rojo, justo debajo de texto en negro.

```
LyX es un procesador de texto de composición.
```

Ahora pulsa en la palabra *"foot"*. La línea de la nota desaparece, dejando *"foot"*, subrayada en rojo, mostrando el sitio donde aparecerá el marcador de la nota en el texto impreso. A esto se le denomina "plegar" la nota. Puedes desplegar la nota en cualquier momento —y volver a editar el texto si quieres— pulsando de nuevo en el marcador rojo.

Te preguntarás por qué el marcador de nota es una palabra en vez de un número. La respuesta es que LyX se encarga también de la numeración de las notas en el texto impreso. Puedes comprobarlo tú mismo mirando la salida `dvi` o impresa. Si añades más notas, LyX las renumera. Como LyX (bueno, realmente LaTeX) se preocupa de esto, no hay necesidad de poner los números en el fichero LyX.

Una nota al pie puede ser cortada y pegada como texto normal. Adelante, ¡inténtalo! Todo lo que necesitas es seleccionar el marcador[25], Cortarlo y Pegarlo. Además, puedes convertir texto normal en una nota, basta que lo selecciones y pulses el botón Insertar Nota a pie; convierte una nota en texto normal pulsando el mismo botón cuando el cursor esté dentro de la nota.

Las notas al margen se pueden añadir mediante el botón Insertar Nota al margen[26] o bien el menú Insertar ▷ Nota al Margen. Son como las notas a pie de página, salvo que:

- los marcadores en pantalla dicen *"margin"* (margen) en vez de *"foot"*
- las notas se sitúan en el margen de la página, en vez de bajo el texto.
- no se numeran
- cuando una nota es plegada, se sitúa un signo de admiración en el margen, que no se verá en el texto impreso.

Convierte ahora tu nota al pie en texto, selecciónala y conviértela en una nota al margen. Ejecuta LaTeX de nuevo para ver su aspecto.

Ejercicio: Arregla la nota a pie de página en `es_ejemplo_sin_lyx.lyx`

12.4.6. Bibliografía

Las bibliografías funcionan de manera similar a las referencias cruzadas. La bibliografía contiene una lista de referencias al final del documento que pueden ser referenciadas desde cualquier parte del texto. Al igual que los títulos de sección, LyX y LaTeX hacen tu trabajo más fácil numerando automáticamente los elementos de la bibliografía y modificando las referencias cuando la numeración cambia.

Ve al final del documento y activa el entorno Bibliography. Ahora, cada párrafo que escribas será una referencia. Escribe `El Tutorial de Lyx, por el equipo documentación de LyX` como primera referencia. Observa que LyX pone automáticamente un número encerrado en un recuadro antes de cada referencia. Pincha con el ratón en el recuadro, y se abrirá una ventana de diálogo Elemento de Bibliografía. El primer campo, la clave, te sirve para referirte a esta entrada desde el documento LyX. Por defecto es un número. Cambia la clave a "tutorialdelyx" para que sea fácil de recordar.

Ahora escoge un lugar cualquiera del documento donde querrías insertar una referencia. Hazlo con Insertar ▷ Referencia a Cita. El programa dibuja un recuadro gris con tres signos de interrogación entre corchetes, y aparece una ventana Cita. El primer campo también se llama Clave, y te permite elegir la entrada bibliográfica que quieres citar[27]. Con ayuda de la flecha hacia abajo a la derecha del campo Clave, selecciona "tutorialdelyx" (en este momento es el único elemento de la bibliografía). Ahora ejecuta LaTeX, y verás que la cita aparece entre corchetes en el texto, referenciando a la bibliografía al final del documento.

¿Cómo se usan los demás campos? El campo Comentario en la ventana de Cita pondrá un comentario (como una referencia a una página o capítulo del libro o artículo) entre corchetes tras la referencia. Si quieres que las referencias tengan etiquetas en vez de números en la salida impresa (por ejemplo, algunos periódicos usarían "[Smi95]" para hacer referencia a un artículo escrito por Smith en 1995), utiliza el campo Label (etiqueta) en la ventana Elemento de Bibliografía. Como siempre, puedes obtener más información en la *Guía del Usuario*.

Ejercicio: Arregla la bibliografía y las citas en `es_ejemplo_sin_lyx.lyx`

[25] Puede serte más fácil seleccionarla con el teclado, ya que puedes abrir sin querer la nota si estás intentando seleccionar el marcador con el ratón.

[26] El botón muestra una flecha apuntando a texto en rojo, al lado de (al margen de) texto en negro, y se encuentra cerca del botón Insertar Nota a pie; en la barra de herramientas.

[27] Esta es la razón por la que es una buena idea dar a las claves nombres únicos y lógicos, en lugar de dejar el número por defecto.

12.4.7. Índice general

Puede que quieras poner un índice al principio de tu documento. LyX hace que sea algo muy fácil. Simplemente pulsa Intro después del título del documento y antes del título de la primera sección[28], y elige en el menú Insertar ▷ Listas e Índice gral. ▷ Índice general. Aparecerá una caja (también conocida como recuadro) con las palabras "Índice general" en la primera línea del documento.

Esto puede no parecer muy útil. Sin embargo, si observas el fichero `dvi`, verás que se ha generado un índice con todas las secciones y subsecciones de tu documento. Una vez más, si reordenas las secciones o añades alguna, estos cambios se verán reflejados en el fichero `dvi` cuando lo actualices.

El índice no se imprime en la versión en pantalla del documento porque no puedes editarlo de ninguna manera. Sin embargo, puedes mostrarlo en una ventana separada pinchado con el ratón en el recuadro del índice o bien mediante Editar ▷ Índice general[29]. Esta ventana es una herramienta muy práctica. Puedes usarla para moverte a través de tu documento. Pulsando en una (sub)sección del índice se resaltará esa línea y el cursor se moverá a ese lugar del documento en la ventana de edición de LyX. También puedes usar los cursores para moverte arriba y abajo en el índice. Puede que te resulte conveniente dejar esta ventana abierta a lo largo de las sesiones de edición.

Para deshacerte del índice, puedes borrar su marcador como cualquier otro carácter.

Ejercicio: Arregla el índice en `es_ejemplo_sin_lyx.lyx`

12.5. Usar las matemáticas

LaTeX es utilizado por muchos científicos porque ofrece una gran calidad en el aspecto de las ecuaciones, evitando los caracteres de control usados por otros procesadores de texto y sus editores de ecuaciones. Sin embargo, muchos de estos científicos se sienten frustrados porque escribir ecuaciones con LaTeX se parece más a programar que a escribir. Afortunadamente, LyX tiene soporte WYSIWYM para las ecuaciones. Si estás acostumbrado a LaTeX, verás que sus comandos matemáticos usuales se pueden introducir normalmente, aunque se muestran de forma WYSIWYM. Por el contrario, si nunca has usado LaTeX, el Panel de Fórmulas te permitirá escribir matemáticas de apariencia profesional de una forma rápida y fácil[30].

12.5.1. El modo matemático

Escribe en cualquier parte de tu documento:

```
Me gusta lo que dijo Einstein, E=mc\textasciicircum{}2, porque es tan simple.
```

Ahora la ecuación no tiene muy buen aspecto, incluso en el fichero `dvi`; no hay ningún espacio entre las letras y el signo igual, y te gustaría escribir el "2" como un verdadero exponente. Esta composición tan mala se debe a que no le hemos dicho a LyX que estamos escribiendo una expresión matemática, así que la compone como texto normal.

Las expresiones matemáticas se escriben en modo matemático o de fórmulas. Para entrar en dicho modo, sólo tienes que pinchar en el botón de la barra de herramientas con un $\frac{a+b}{c}$ escrito en azul. LyX abrirá un pequeño cuadro azul, con un rectángulo magenta a su alrededor. El cuadrado azul es el punto de inserción, que te indica que está esperando a que insertes algo, y el rectangulo indica que estás en el modo matemático. LyX ha situado el cursor en el cuadro azul, así que introduce de nuevo `E=mc^2`. La expresión se escribe en azul, y el cuadro azul desaparece tan pronto como el punto de inserción deja de estar vacío. Ahora pulsa Esc para dejar el modo matemático (nota: pinchar en el botón modo de fórmulas otra vez no te servirá). El rectángulo magenta desaparece, dejando el cursor a la derecha de la expresión y ahora si escribes algo será texto normal.

Ejecuta LaTeX y mira el fichero `dvi`. Observa que la expresión se ha impreso bien, con espacios entre las letras y el signo igual, y el "2" como superíndice. Se asume que en modo matemático las letras son variables, y por tanto vienen en cursiva. Los números son sólo números y no van en cursiva.

Este nuevo modo es otro ejemplo de la filosofía WYSIWYM. En LaTeX, escribes una expresión matemática mediante texto y comandos como `\sqrt`; esto puede ser desesperante, ya que no puedes ver cómo queda la expresión hasta que ejecutas

[28] No te desesperes tratando de pinchar o borrar un número de sección. No funcionará. No se permite editar el número de sección de ninguna forma, ya que LyX controla el numerado de secciones.
[29] El comando del menú funcionará incluso si no tienes recuadro de índice en tu documento.
[30] Lyx no puede comprobar si los resultados matemáticos que escribes son realmente *correctos*. Lo sentimos.

LaTeX, y puedes perder mucho tiempo buscando algún paréntesis que falte u otros fallos. Por el contrario, LyX no pretende que la expresión aparezca perfecta en la pantalla (WYSIWYG), pero te da una buena idea de cuál será su aspecto impreso. LaTeX se encarga de la composición profesional. El 99 % del tiempo no tendrás que preocuparte de cambiar tamaños de letra o espaciado alguno. De esta forma (sentimos ser tan repetitivos) puedes concentrarte en el *contenido* de la expresión matemática, no en su formato.

12.5.2. Navegar por una ecuación

Ahora vamos a cambiar $E = mc^2$ a $E = 1 + mc^2$. Usa los cursores para introducirte en la expresión. Observa que cuando entras en la expresión el rectángulo magenta aparece para informarte de que estás de nuevo en modo de fórmulas. Ahora puedes utilizar las teclas Izquierda y Derecha para mover el cursor tras el signo de igualdad, y escribir "1+". De nuevo, utiliza los cursores o Esc para salir de la expresión, desapareciendo así el rectángulo magenta. Mucha gente encuentra apropiadas las teclas de dirección, pero también puedes pichar con el ratón en cualquier parte de la expresión para situar el cursor ahí y empezar el modo matemático.

Salvo para las teclas especiales descritas más adelante, escribir en el modo matemático es como editar texto normal. Usa Suprimir (o Retroceso) para borrar. Selecciona texto tanto con los cursores como con el ratón. Edición ▷ Deshacer funciona de la misma forma, al igual que cortar y pegar. Hay una cosa con la que debes tener cuidado: si estás fuera de la expresión y pulsas Suprimir (o Retroceso) la borrarás entera. Afortunadamente puedes Deshacer para recuperarla.

¿Y qué ocurre si quieres cambiar $E = mc^2$ a $E = mc^{2,5} + 1$? Una vez más, puedes utilizar el ratón para pinchar en el sitio que vas a modificar. También puedes utilizar las teclas de dirección. Si el cursor se encuentra detrás de la "c" y delante del "2", pulsando Arriba moverás el cursor al nivel del superíndice, justo antes del "2". Añade el ".5". Ahora, pulsando Abajo devolverás el curso al nivel normal. De hecho, si pulsas Abajo desde cualquier parte del superíndice, el cursor se sitúa justo *tras* éste (de manera que puedes introducir el "+1").

También puedes usar la Barra espaciadora para recorrer una expresión. Si ya estás en una estructura matemática (un subíndice, superíndice, fracción, raíz cuadrada, delimitadores o matriz, todo lo cual se describe en las siguientes secciones), pulsando Espacio moverás el cursor detrás de la estructura, permaneciendo en modo matemático. Así, si el cursor está en cualquier parte del superíndice, pulsando Espacio el cursor volverá al nivel normal y justo detrás del superíndice. Esto quiere decir que puedes escribir $E = mc^{1+x} - 2$ sin usar el ratón o las teclas de dirección, un método que seguramente preferirás una vez que vayas cogiendo práctica. Ten cuidado y no pulses Espacio estando entre el "1" y el signo más, o saldrás del superíndice. En aquellos sitios donde estas acciones no tienen sentido (por ejemplo, entre la "m" y la "c"), la Barra espaciadora no tiene efecto alguno[31].

Observa que si introduces una expresión y sales con Esc, no habrá ningún espacio tras ésta. Esto viene bien si vas a escribir un punto o una coma, pero si lo que quieres es escribir una palabra tras la fórmula, tienes que introducir el Espacio explícitamente después de salir de ella. Un atajo consiste en pulsar Espacio justo al final de la fórmula, lo cual te saca de ella *y* además añade un espacio. Así, puedes escribir "$f = ma$ es mi ecuación favorita" en vez de "$f = ma$es mi ecuación favorita".

12.5.3. Exponentes e índices

Un exponente se puede introducir desde el menú Fórmulas, pero es más sencillo pulsar la tecla del acento circunflejo "^". LyX coloca un punto de inserción (el recuadro azul, ¿recuerdas?) en el superíndice para que lo siguiente que introduzcas sea superíndice, con un tamaño de letra menor. Todo lo que escribas hasta que pulses Espacio (o Esc para salir completamente de la expresión) será puesto en superíndice.

Escribir un subíndice es igual de fácil: para comenzar uno pulsa la tecla de subrayado, "_". Puedes introducir subíndices y superíndices tanto en subíndices como en superíndices, como en esta fórmula: $A_{a_0+b^2} + C^{a_0+b^2}$.

Ejercicio: Pon la ecuación 1 del fichero `es_ejemplo_sin_lyx.lyx` en modo matemático.

12.5.4. El Panel de Fórmulas

El Panel de Fórmulas es una forma práctica de introducir símbolos o realizar complicadas funciones matemáticas. Muchas de estas funciones pueden llevarse a cabo con el teclado o con el menú Fórmulas. Sin embargo, nos vamos a centrar en el uso del panel para que conozcas todo lo que hay; podrás aprender atajos con el teclado más tarde en otros manuales (esto es una indirecta). Así que abre el Panel de Fórmulas y déjalo abierto durante toda la sección.

[31] El Espacio y el Tabulador *no* se utilizan para introducir espacios entre los elementos de una ecuación. Este espacio es parte de la composición, lo que significa de debes dejar que LyX se preocupe de él (ver Sec. 12.2.3). Si este espaciado no te satisface del todo, hay maneras para ajustarlo, para lo cual puedes mirar en la *Guía del Usuario* (pero no te molestes en hacer ajustes hasta que hayas escrito todo el documento).

12.5.4.1. Símbolos y letras griegas

Si pinchas en el botón "$\Gamma\rho\epsilon\epsilon\kappa$" del panel, obtendrás un menú desde el que podrás elegir una letra griega. Ésta aparecerá allí donde esté situado el cursor. Observa que hay un par de variantes de épsilon, pi, theta, y sigma. Un atajo: si estás escibiendo texto normal e insertas algo desde el panel, entras automáticamente en modo matemático.

Otros cuatro botones en la parte de abajo del panel te permiten seleccionar en una matriz gran cantidad de símbolos usados en matemáticas: flechas, relaciones, operadores, sumas e integrales. Ten en cuenta que los superíndices y los subíndices te permiten establecer los límites superior e inferior en estas últimas. El último botón es el cajón de sastre, llamado Varios. "No hay nada que no puedas hacer... Todo lo que necesitas es ♡."

12.5.4.2. Raíces cuadradas, tildes y delimitadores

Para escribir una raíz cuadrada sólo tienes que pulsar el botón con su símbolo. La raíz aparece, y el cursor va a un nuevo punto de inserción dentro de ella. Aquí puedes introducir variables, números, otras raíces cuadradas, fracciones y todo lo que quieras. LyX irá cambiando automáticamente el tamaño de la raíz para que se ajuste a su contenido.

La colocación de tildes en una letra ($\vec{v}$) o grupo de letras ($\overrightarrow{a+b}$) se realiza de la misma forma. Pulsa el botón que tiene un cuadro azul con una tilde negra ($\tilde{}$) encima. Aparecerá la ventana de Decoración. Pincha en una de las decoraciones, y LyX la imprimirá con un punto de inserción debajo (o encima). Escribe lo que quieras y pulsa Espacio para "salir" de ella.

Los delimitadores (como paréntesis, corchetes y llaves) funcionan de forma similar, pero son un poco más complicados. Pulsa el botón con un cuadro azul entre corchetes para abrir la ventana Delimitador. Pincha en un delimitador izquierdo con el *botón izquierdo del ratón* y en un delimitador derecho con el *botón derecho*. (De forma alternativa, puedes usar sólo el botón izquierdo utilizando los selectores etiquetados como "Izq." y "Dcha" al elegir cada delimitador). En cada instante, tu selección de delimitadores aparece en un recuadro en la parte de arriba de la ventana. Por defecto es un par de paréntesis, pero con este método de selección puedes elegir un par de llaves, una llave y un paréntesis, o incluso elegir el delimitador vacío para obtener algo así: "$a = \langle 7$" (el delimitador vacío se muestra como una línea negra discontinua en LyX, pero no aparece en la salida).

Una vez que has elegido tus delimitadores, pulsa OK para que aparezcan en la expresión (o pulsa Aplicar si quieres dejar la ventana abierta). Si eres un poco vago, puedes escribir los paréntesis normales desde el teclado en modo matemático, en vez de usar la ventana Delimitador. Sin embargo, estos paréntesis tendrán el mismo tamaño que el texto normal, lo cual quedará bastante mal si encierran una fracción o una matriz grande. Usar la ventana Delimitador te garantiza que los delimitadores se ajusten de forma precisa a aquello que encierren.

También puedes poner delimitadores, o una raíz cuadrada o una tilde sobre texto ya existente. Selecciona la porción de fórmula que quieres ajustar, y pulsa el botón que quieras del Panel de Fórmulas. Prueba a hacer esto para cambiar la segunda ley de Newton de forma escalar a forma vectorial ($f = ma$ a $\vec{f} = m\vec{a}$). Una vez que hayas aprendido a hacer matrices, esta será la manera de encerrarlas entre paréntesis o corchetes.

12.5.4.3. Fracciones

Las fracciones son muy sencillas en modo matemático. Pincha en el botón Fracción del panel (el que tiene un par de cuadros azules a modo de numerador y denominador). LyX coloca dos puntos de inserción. Como cabría esperar, puedes usar los cursores o el ratón para moverte a través de la fracción. Pincha en el cuadro de arriba y escribe "1". Después pulsa Abajo y escribe "2". ¡Acabas de hacer una fracción! Por supuesto, puedes escribir cualquier cosa dentro de cada uno de los dos cuadros: variables con exponentes, raíces cuadradas, otras fracciones, cualquier cosa.

Ejercicio: Pon la ecuación 2 en el fichero `es_ejemplo_sin_lyx.lyx` en modo matemático.

12.5.4.4. Modo TeX: Límites, logaritmos, senos y otros

Como las letras en modo de fórmulas se consideran variables, si introduces "sin" LyX cree que estás escribiendo el producto de tres variables s, i, y n. Las tres se imprimirán en cursiva, cuando lo que realmente querías era la palabra "sin" escrita en letra normal. Además, no se pondrá espacio alguno entre "sin" y la "x" (pulsar Espacio saldrá del modo matemático). Así que, ¿cómo conseguir "$\sin x$" en vez de "$sinx$"?

Pincha en "sin" en la lista Funciones del Panel de Fórmulas. La palabra "sin" se escribe en rojo, en tipo normal, también llamado modo TeX. La palabra completa se trata como un único símbolo, de manera que si pulsas Suprimir, la borrarás

entera. Ahora escribe "x", que aparecerá azul y en cursiva, como es de esperar en modo matemático. En el fichero `dvi`, la expresión se verá correctamente. Inténtalo.

Más comandos que necesitas escribir en modo TEX usando la lista de Funciones incluyen otras funciones trigonométricas y sus inversas, funciones hiperbólicas, logaritmos, límites, y unas pocas más. Todas aceptan subíndices y superíndices, lo cual es importante para poder escribir "$\cos^2\theta$" o "$\lim_{n\to\infty}$".

Ejercicio: Pon la ecuación 3 del fichero `es_ejemplo_sin_lyx.lyx` en modo matemático.

12.5.4.5. Matrices

Pulsa en el botón Matriz del panel. Se abrirá una ventana del mismo nombre , con dos barras deslizantes para que elijas cuántas filas y columnas quieres que tenga tu matriz. Selecciona 2 filas y 3 columnas y pulsa Aplicar u OK. LyX imprime seis puntos de inserción formando una matriz 2×3. Como siempre, puedes introducir cualquier clase de expresión matemática (un raíz cuadrada, otra matriz, etc) en cada uno de ellos. También puedes dejar alguno vacío si quieres.

Puedes usar el Tabulador para moverte horizontalmente entre las columnas de la matriz. De manera alternativa, puedes usar las teclas de dirección para desplazarte (pulsando Derecha al final de un cuadro moverá el cursor al siguiente, Abajo lo moverá a la siguiente fila, etc).

Mira la *Guía del Usuario* para obtener información sobre cómo cambiar la alineación de cada columna y la posición vertical de la matriz completa. Si lo que quieres es escribir una tabla que contenga texto, deberías usar el magnífico soporte de tablas de LyX, en lugar de tratar de escribir texto en una matriz.

12.5.4.6. El modo demostración

Todas las expresiones que hemos escrito hasta ahora estaban en la misma línea que el texto que las rodeaba. Reciben pues el nombre de expresiones en línea. Están bien para expresiones cortas y sencillas, pero si quieres escribir cosas más extensas, o si quieres que queden separadas del texto, tienes que escribirlas en modo demostración. Además, sólo las expresiones en este modo pueden ser etiquetadas o numeradas (ver *Guía del Usuario*). Por último, las ecuaciones de varias líneas (ver Sec. 12.5.5) también deben estar en modo demostración.

Pincha en el botón Mostrar del Panel de Fórmulas, que representa un par de líneas de texto rodeando un cuadro azul. LyX abre un punto de inserción, pero lo coloca en una nueva línea y centrado. Ahora escribe una expresión y ejecuta LaTeX para ver el resultado. El botón Mostrar actúa como un conmutador entre modo demostración y modo matemático normal; úsalo ahora para cambiar un par de expresiones a modo demostración y viceversa.

Este modo tiene un par de diferencias con respecto al modo normal:

- El tipo de letra por defecto es de menor tamaño para unos pocos símbolos, como $\sum$ y $\int$
- Los subíndices y superíndices en las sumas y límites (no en las integrales) se escriben debajo, en lugar de seguir a los símbolos
- El texto se centra

Aparte de estas diferencias, las expresiones en línea y en modo demostración son muy similares.

Un último apunte acerca del modo en que se componen las ecuaciones en este modo: presta atención a si lo que quieres es poner una ecuación en un nuevo párrafo o no. Si ésta se encuentra en mitad de de una frase o de un párrafo, no pulses Intro. De lo contrario harás que el texto tras la ecuación comience en un nuevo párrafo. Este texto será por tanto sangrado, que probablemente no es lo que querías.

Ejercicio: Pon las ecuaciones del fichero `es_ejemplo_sin_lyx.lyx` en modo demostración, y comprueba que se componen de manera diferente.

Ejercicio: Utilizando las herramientas que has aprendido en esta sección, deberías de ser capaz de escribir una ecuación como ésta:

$$f(x) = \begin{cases} \log_8 x & x > 0 \\ 0 & x = 0 \\ \sum_{i=1}^{5} \alpha_i + \sqrt{-\frac{1}{x}} & x < 0 \end{cases}$$

12.5.5. Ecuaciones de varias líneas

Intenta escribir las siguientes ecuaciones y observa el resultado en el fichero `dvi`. Tendrás que introducir dos ecuaciones distintas en modo demostración.

$$x = y + y + y + y + y$$

$$= 5y$$

¡No queda bien! Al escribir dos o más ecuaciones seguidas, quedan mucho mejor si sus signos de igualdad están alineados; esto se hace especialmente patente si la segunda ecuación no tiene parte izquierda. LyX te permite escribir ecuaciones de varias líneas con cierto control sobre su alineación.

$$\begin{aligned} x &= y + y + y + y + y \\ &= 5y \end{aligned}$$

¡Esto es otra cosa! Los signos de igualdad están alineados, y hay menos espacio entre las ecuaciones.

Para empezar a escribir una ecuación de varias líneas, abre una expresión en modo demostración y pulsa C-Retorno de carro. LyX imprimirá dos líneas, cada una con tres puntos de inserción. Al igual que en las matrices, puedes usar el el ratón, los cursores o el Tabulador para desplazarte a través de ellos. Intenta reproducir la ecuación anterior. Ten en cuenta que es legal dejar uno o más puntos de inserción vacíos. Esto es útil para ejemplos como el de arriba, o para separar ecuaciones muy largas, como:

$$\begin{aligned} x &= a + b + c + d \\ &\quad + e + f + g \end{aligned}$$

LyX alineará el segundo campo de cada línea, donde usualmente pondrás signos de igualdad u otros operadores; de hecho, puedes poner cualquier cosa. Pero no uses una ecuación de varias líneas para escribir una matriz. Para eso ya está la herramienta apropiada (see Sec. 12.5.4.5).

Si quieres un conjunto mayor de ecuaciones, usa C-Retorno de carro para añadir nuevas líneas con tres puntos de inserción vacíos. Si no estás al final de la línea cuando lo hagas, lo que quede de línea será llevado a la siguiente. Si pulsas C-Retorno de carro cuando ya has escrito una ecuación (de una línea), la ecuación entera quedará en el primer campo. Sitúa el cursor antes del signo igual y pulsa C-Tabulador para moverlo al segundo campo. Colócate pasado el signo y pulsa de nuevo C-Tabulador para mover el resto de la ecuación al tercer campo. Prueba a cambiar tu ecuación $E = mc^2$ a:

$$\begin{aligned} E &= mc^2 \\ &= mc \times c \end{aligned}$$

Si has escrito demasiadas líneas, sitúa el cursor al final de una de ellas y usa M-e k para borrar la siguiente. Se eliminará la alimentación de línea, concatenando la línea siguiente (los tres puntos de inserción) con el final de línea en la que estás. Si aquélla está vacía, simplemente la borrará. Aviso: usar M-e k cuando no estás al final de la línea puede llevar a comportamientos extraños.

12.5.6. Más sobre matemáticas

El modo matemático puede hacer muchas cosas más. Por ahora te has familiarizado con lo básico, así que dirígete a la *Guía del Usuario* si quieres buscar trucos para:

- Etiquetar y numerar expresiones.

- Cambiar el tipo de letra (i.e., escribir texto en negrita dentro de una expresión). Aprovechamos para decirte que pulsando en el botón Modo de Fórmulas de la barra de herramientas dentro de una expresión te permitirá escribir con tipo de letra normal hasta que introduzcas un espacio (no protegido).
- Ajustar los tamaños de letra y el espaciado en una expresión. (No te preocupes de estas cosas hasta el borrador final).
- Escribir macros. Son muy potentes, ya que te permiten definirlas al principio del documento, y usarlas en cualquier parte de él. Si cambias la definición de una macro, las referencias a ella cambiarán en todo el documento. Las macros pueden incluso tomar parámetros.
- Hacer un montón de cosas más que no tenemos tiempo de contarte en este *Tutorial*.

12.6. Otras características importantes de LyX

No hemos tratado todos los posibles comandos de LyX, y no tenemos intención de hacerlo. Como siempre, ve a la *Guía del Usuario* para más información. La función exacta de cada comando del menú se describe en el *Manual de Referencia*. Sólo mencionaremos un par más de cosas importantes que LyX puede hacer...

- LyX posee soporte WYSIWYG para tablas. Usa el comando Insertar ▷ Tabla para crear una. Pincha en la tabla con el *botón derecho* para que aparezca la ventana de Formato de Tabla, que te ofrece extensas posibilidades de edición de la tabla.
- LyX también permite incluir gráficos PostScript® (o LaTeX puro). Lo adivinaste: Insertar ▷ Figura. Después pincha en la figura para elegir el fichero que quieres incluir, rotarla, escalarla, etc. Tanto las tablas como las figuras tienen etiqueta, y LyX genera automáticamente listas de figuras y/o de tablas.
- Soporte de control de versiones, usando RCS (`man rcsintro` para más información).
- LyX es altamente configurable. Todo, desde el aspecto de la ventana hasta la forma de la salida, puede ser configurado de múltiples maneras. Gran parte de la configuración se lleva a cabo editando el fichero `lyxrc`[32]. `Para más información sobre esto, echa un vistazo a` Ayuda ▷ Personalización.
- LyX está siendo desarrollado por un equipo de programadores en los cinco continentes. De esta forma, tiene mejor soporte para otros idiomas además del inglés (como holandés, alemán, griego, checo, turco, ...) que muchos procesadores de texto. Puedes escribir documentos en otros idiomas, pero también puedes configurar LyX para que muestre los menús y los mensajes de error en otras lenguas.
- Los menús de LyX tienen asociadas combinaciones de teclas. Esto significa que puedes hacer Archivo ▷ Abrir tecleando M-F seguido de O. Las asociaciones de teclas también son configurables (y puede haber asociaciones incluso para algunos de los menús traducidos del inglés). Para más información, busca en Ayuda ▷ Personalización.
- LyX puede leer documentos LaTeX. Ver Sec. 12.7.2.
- Corrige los errores ortográficos de tu documento con Edición ▷ Ortografía.[33]

12.7. LyX para usuarios de LaTeX

Si no sabes nada de LaTeX, no tienes que leer esta sección. Realmente, puede que quieras *aprender* algo sobre LaTeX, y entonces leas este capítulo. Sin embargo, a mucha gente que empieza a usar LyX, LaTeX le será familiar. Si ese es tu caso, te estarás preguntando si verdaderamente LyX puede hacer todo lo que hace LaTeX. La respuesta corta es que, de una forma u otra, puede hacer prácticamente todo lo que hace éste, simplificando en gran medida el proceso de escritura de un documento mediante LaTeX. Actualmente hay algunos problemas convirtiendo antiguos documentos LaTeX y en un par de cosas más, pero las versiones posteriores subsanarán esto.

Como sólo se trata de un tutorial, vamos a mencionar únicamente aquellas cosas que los nuevos usuarios de LyX vayan a encontrar interesantes. Para mantener corto el *Tutorial*, vamos a dar información mínima. La *Guía del Usuario* ya posee una gran cantidad de información sobre las diferencias entre LyX y LaTeX, y de cómo hacer varios trucos de LaTeX en LyX.

[32] Actualmente, tienes que editar el fichero `lyxrc` con un editor de texto. Los programadores esperan crear una interfaz para la configuración dentro de LyX.

[33] Ten en cuenta que el corrector ortográfico sólo comprueba desde el cursor hasta el final del documento.

12.7.1. Modo TeX

Todo lo que introduzcas en modo TeX se imprimirá en rojo en la pantalla, y será pasado directamente a LaTeX. Entra en este modo con Formato ▷ Estilo TeX o pinchando en el botón con TeX escrito en rojo de la barra de herramientas.

En el modo matemático, el modo TeX se maneja de forma ligeramente diferente. En este caso para entrar tienes que introducir una contrabarra "\". Esta no saldrá en pantalla, pero todo lo que escribas de ahí en adelante aparecerá en rojo. Para salir pulsa Espacio o cualquier otro carácter no alfabético, como un número, el subrayado, el circunflejo o el paréntesis. Una vez que salgas del modo TeX, si LyX conoce el comando que acabas de introducir lo convertirá a formato WYSIWYM. Así, si en modo matemático escribes `\gamma`, cuando pulses Espacio, LyX convertirá el texto en rojo en una "γ" de color azul. Esto funcionará para casi todas las macros matemáticas que no sean muy complejas (aunque debes tener en cuenta que funciones como `\sin` permanecen en rojo, ya que esa es su forma WYSIWYM). Este sistema puede ser más rápido que usar el Panel de Fórmulas, especialmente para usuarios experimentados de LaTeX.

Como caso particular cabe comentar que si escribes una llave en modo TeX dentro de una expresión matemática, aparecerán en rojo tanto la llave de apertura como una de cierre, se te sacará *fuera* del modo TeX y se situará el cursor entre las llaves. Esto hace que sea más práctico para escribir comandos LaTeX que el modo matemático no conoce y que toman un parámetro.

LyX no puede hacer absolutamente todo lo que hace LaTeX (¿todavía no?). Algunas funciones muy elaboradas no están soportadas, mientras que algunas funcionan pero no son WYSIWYG. El modo TeX permite a los usuarios disponer de la completa flexibilidad de LaTeX, a la vez que gozan de las útiles características de LyX, como las matemáticas WYSIWYG, las tablas y la edición de texto. LyX no podría abarcar nunca todos los paquetes de LaTeX. Sin embargo, escribiendo `\usepackage{nombre_paquete}` en el preámbulo (ver Sección 12.7.4.2), puedes utilizar el paquete que quieras (aunque no tendrás soporte WYSIWYG para las características de ese paquete).

12.7.2. Importar documentos LaTeX: `reLyX`

Puedes importar un fichero LaTeX usando el comando Archivo ▷ Importar ▷ LaTeX. Éste llamará a un programa Perl llamado reLyX, que creará un fichero `fich.lyx` a partir del fichero `fich.tex` y lo abrirá. Si la traducción no funciona, puedes probar a ejecutar `reLyX` desde la línea de comandos[34], posiblemente usando opciones más complejas.

`reLyX` traducirá la mayoría de los comandos legales de LaTeX, pero no todo. Dejará lo que no entienda en modo TeX, así que después de la traducción puedes buscar el texto en rojo y editarlo a mano para tratar de arreglarlo.

`reLyX` tiene su propia página del manual. Léela si quieres conocer los comandos y entornos de LaTeX que no admite, fallos (y cómo evitarlos), y cómo usar las distintas opciones.

12.7.3. Convertir documentos LyX a LaTeX

Puede que desees convertir un documento LyX en un fichero LaTeX. Por ejemplo, un compañero de trabajo o colaborador que no tiene LyX puede querer leerlo. Esto se soluciona muy fácilmente con LyX. Selecciona Archivo ▷ Exportar ▷ como LaTeX. Se creará un fichero `cualquiera.tex` a partir del fichero `cualquiera.lyx` que estés editando. Al fin y al cabo, el programa siempre genera ficheros temporales LaTeX cuando visualiza o imprime los documentos, así que no tiene ningún problema para hacer esto.

12.7.4. Preámbulo LaTeX

12.7.4.1. Clase de documento

El menú Formato ▷ Documento se encarga de muchas de las opciones que introducirías en un comando `\documentclass`. Cambia aquí la clase del documento, el tamaño de letra por defecto y el tamaño del papel. Pon cualquier opción adicional en la zona de Opciones extra.

12.7.4.2. Otros aspectos del preámbulo

Si tienes que poner comandos especiales en el preámbulo de un fichero LaTeX, también puedes hacerlo en un documento de LyX. Elige Formato ▷ Preámbulo Latex y escribe en la ventana que aparecerá. Todo lo que introduzcas se le pasará directamente a LaTeX (como el modo TeX).

[34] Cuando LyX se instala, se crea un fichero ejecutable separado llamado `reLyX` en el mismo directorio que el propio `lyx` (i.e., `/usr/local/bin/reLyX`). `reLyX` requiere Perl (versión 5.002 en el momento de escribir esto).

12.7.5. BibTeX

LyX tiene un soporte de BibTeX bueno pero incompleto, el cual te permite construir bases de datos de referencias bibliográficas que pueden usarse en múltiples documentos. Selecciona en el menú Insertar ▷ Listas e Índice gral. ▷ Referencia BibTeX para incluir un fichero `bib`. Pincha en el cuadro resultante de "Referencias BibTeX generadas" y obtendrás un menú BibTeX. En el campo Base de Datos escribe aquello que pondrías dentro de las llaves del comando `\bibliography{}`[35]. En el campo de Estilo, escribe aquello que corresponda al comando `\bibliographystyle{}`.

Después de hacer esto, puedes hacer referencia a cualquier entrada de las bibliografías que hayas incluido mediante Insertar ▷ Referencia a Cita (ver Sección 12.4.6). El programa se preocupará de ejecutar BibTeX. La razón por la que decimos "soporte bueno pero incompleto" es que LyX no es capaz de crear ficheros `bib`, y además no te ofrece la lista con las referencias de tu fichero `bib` en la ventana Cita.

12.7.6. Misceláneo

Inserta un espacio protegido con C-espacio. Aparecerá en pantalla como una pequeña "u" rosa. Hay montones a lo largo de este *Tutorial*. En el menú Insertar ▷ Carácter Especial verás otros caracteres especiales, incluyendo puntos suspensivos, salto de línea forzoso y punto de ruptura de palabra.

12.8. ¡Errores!

A veces, al ejecutar LaTeX habrá errores, cosas que LyX o el propio LaTeX no entienden. Cuando esto sucede, LyX crea un recuadro de error (con la palabra "error" dentro). Pulsando sobre el recuadro se abrirá una ventana que muestra el mensaje de error concreto. Si se trata de algo que has hecho mal con LyX, será un error de LyX. Estos errores son muy raros. Si el problema es de LaTeX (la mayoría de las veces ocurre con cosas que has escrito en modo TeX) LyX se limitará a citar el mensaje de error que éste le devuelve.

[35] Como en LaTeX normal, dos o más bibiografías deben separarse por comas, sin espacios en blanco.

TEMA 13

LaTeX

13.1. Introducción preliminar

LaTeX [?] es un sistema de composición de textos especialmente orientado a la creación de documentos científicos que contengan fórmulas matemáticas. Además, también se pueden crear otros tipos de documentos, que pueden ser desde cartas sencillas hasta libros completos. LaTeX está organizado sobre TeX [?].

Este tema está dedicado a LaTeX y pretende versar sobre lo básico que se debe poder emplear en la mayoría de las aplicaciones de LaTeX. Existen diversos manuales [?, ?] donde se encuentra una descripción completa de LaTeX.

LaTeX está disponible para la mayoría de los miniordenadores y microordenadores, desde IBM PCs en adelante. En muchas redes universitarias de ordenadores se encuentra instalado para utilizarse al instante. Normalmente existe una *Guía Local* [?] donde podrás averiguar cómo acceder a la instalación de LaTeX, cómo se opera con ella y de qué complementos se dispone.

El propósito de este tema *no* es indicar cómo se instala y se mantiene un sistema de LaTeX, sino mostrar cómo escribir documentos para poderlos procesar con LaTeX.

Esta descripción se divide en cuatro apartados:

El primer apartado muestra la estructura básica de los documentos de LaTeX 2_ε. También se enseña un poco de la historia de LaTeX. Tras leer este tema deberías tener una visión muy escueta de LaTeX. Esta visión consistirá sólo de un pequeño "marco de trabajo" en el que podrá integrar la información que se proporcionamos en los demás apartados y la que podrás hallar en otras fuentes —como los manuales [?, ?]—.

El segundo apartado incide en los detalles sobre la composición de los documentos. En este aparatdos se explican la mayoría de las instrucciones y los entornos básicos de LaTeX. Una vez leído este apartado serás capaz de escribir tus primeros documentos.

En el tercer apartado se explican algunas nociones sobre cómo componer fórmulas matemáticas con LaTeX. Aquí te presentamos varios ejemplos para ayudarte a entender uno de los principales potenciales de LaTeX.

El cuarto apartado indica otras posibilidades que se pueden obtener de LaTeX que, si bien no son esenciales, a veces pueden resultar muy útiles. Por ejemplo, se muestra cómo incluir gráficos de PostScript Encapsulado (EPS) en sus documentos o cómo añadir un índice de materias en su documento.

El último apartado resume las herramientas básicas que debes saber manejar para trabajar con documentos LaTeX.

Se recomienda encarecidamente leer estos apartados por orden. También sería altamente recomendable estudiarse los ejemplos, ya que en ellos es donde se encuentra gran parte de la información útil.

Si por alguna razón hay necesidad de conseguir cualquier material relacionado con LaTeX, siempre se puede recurrir a cualquiera de los servidores de archivos de `CTAN`. En la República Federal de Alemania es `ftp.dante.de` y en el Reino Unido es `ftp.tex.ac.uk`. También existen otros servidores replicados. En caso de no encontrarse en uno de estos países, siempre es mejor elegir el servidor más cercano.

13.2. Lo que es fundamental saber

En la primera parte de este apartado presentamos una visión general de la filosofía e historia de LaTeX 2_ε. En la segunda parte se incide en las estructuras básicas de un documento de LaTeX. Tras leer este tema, podrás tener un conocimiento básico del modo de funcionamiento de LaTeX. A medida que se avanza, la información de este apartado resultará de mucha ayuda para integrar toda la información adicional que puedas obtener sobre LaTeX, tanto en el resto de los apartados como de otros sitios.

13.2.1. El nombre del juego

TeX TeX es un programa de ordenador de Donald E. Knuth [?]. Está orientado a la composición e impresión textos y fórmulas matemáticas.

TeX se pronuncia "Tech", con una "ch" como en la palabra alemana "Buch" o en la escocesa "Loch". Éste es el sonido de una 'h' aspirada, como en la onomatopeya española "argh". En un entorno `ASCII` TeX se escribe `TeX`.

LaTeX LaTeX es un paquete de macros que nos permite componer e imprimir más fácilmente un documento con la calidad tipográfica de TeX, empleando para ello patrones definidos. Originalmente, LaTeX lo escribió Leslie Lamport [?]. A grosso modo, podemos decir que se utiliza el cajista TeX como elemento de composición.

Desde diciembre de 1994, el paquete LaTeX está siendo actualizado por un equipo llamado LaTeX3, que dirige Frank Mittelbach, para incluir algunas de las mejoras que se habían solicitado desde hace tiempo y para reunificar todas las versiones retocadas que han surgido desde que apareciera la primera versión LaTeX 2.09 hace ya algunos años. Para distinguirla de la vieja, a la nueva versión se le llama LaTeX 2_ε. Lo que aquí presentamos es LaTeX 2_ε.

LaTeX se pronuncia "Lei-tegh", aunque entre los hispanohablantes se ha aceptado "La-tegh". Para referirnos a LaTeX en un entorno `ASCII` escribiremos `LaTeX`. LaTeX 2_ε se pronuncia "Lei-tegh tu íi" —aunque muchos nos empeñamos en decir "Lategh dos e"— y se puede escribir `LaTeX2e`.

Conceptos básicos

Autor, diseñador y cajista. Normalmente, para una publicación el autor le entrega a la editorial un escrito a máquina. El diseñador de libros de la editorial decide entonces el formato del documento (longitud de los renglones, tipo de letra, espacios antes y después de cada capítulo, etc.) y le da estas instrucciones al cajista para producir este formato.

La persona que diseña estos libros intenta averiguar las intenciones del autor mientras ha realizado el escrito. Entonces decide sobre el modo de presentar los títulos de capítulos, citas, ejemplos, fórmulas, etc., basándose en su saber profesional y en el contenido del escrito.

En un entorno de LaTeX, LaTeX realiza el papel del diseñador de libros y emplea a TeX como cajista. Pero LaTeX *sólo* es un programa y, por tanto, necesita más ayuda para sus decisiones que un diseñador humano de libros. El autor tiene que proporcionarle información adicional que describa la estructura lógica del texto. Esta información se indica dentro del texto a través de las *instrucciones* u *órdenes* de LaTeX.

Esto es bastante diferente del enfoque WYSIWYG[1] de la mayoría de los procesadores de textos tales como *Microsoft Word* o *FrameMaker*. Con estas aplicaciones, el autor establece el formato del texto con la entrada interactiva al introducirlo en el ordenador. En cada momento, el autor ve en pantalla el aspecto que tendrá el trabajo final cuando lo imprima.

Por regla general, al emplear LaTeX el autor no ve, al redactar el texto, cómo va a resultar la composición final. Sin embargo, existen herramientas que permiten mostrar en pantalla lo que finalmente se va a obtener tras procesar los ficheros con LaTeX. Con éllas puedes siempre realizar correcciones antes de enviar el documento final a la impresora.

Diseño del formato. El diseño tipográfico es una artesanía que se debe aprender. Los autores inexpertos cometen con frecuencia graves errores de diseño. Muchos profanos creen erróneamente que el diseño tipográfico es, ante todo, una cuestión de estética: si el documento presenta un buen aspecto desde el punto de vista artístico, entonces creen que está bien "diseñado". Sin embargo, ya que los documentos se van a leer y no a colgarse en un museo, es más importante presentar una mayor legibilidad y una mejor comprensión que un aspecto más agradable.

Por ejemplo, algunas reglas básicas que deberían seguirse son las siguientes:

[1] Siglas que significan *What you see is what you get,* lo que se ve es lo que se obtiene.

- Se debe elegir el tamaño de las letras y la numeración de los títulos de modo que la estructura de los capítulos y las secciones sea fácilmente reconocible.
- Se debe elegir la longitud de los renglones de modo que se evite el cansancio por el movimiento de los ojos del lector y no para que rellenen, a ser posible, las páginas con un aspecto estéticamente bueno.

Con los sistemas WYSIWYG los autores producen, en general, documentos estéticamente bonitos pero con una estructura muy escasa o inconsistente. LaTeX evita estos errores de formato, ya que con LaTeX el autor está obligado a indicar la estructura *lógica* del texto. Entonces LaTeX elige el formato más apropiado para éste.

Ventajas e inconvenientes. Una cuestión que se discute a menudo cuando la gente del mundo WYSIWYG se encuentra con la gente que utiliza LaTeX es sobre "las ventajas de LaTeX sobre un procesador de textos normal" o al revés. Cuando uno se ve ante una discusión como ésta, lo mejor que se puede hacer es mantener una postura de asentimiento, ya que las cosas suelen salirse de control. Pero a veces no se puede huir...

En estas circunstancias hay que recordar algunas de las principales ventajas de LaTeX sobre los procesadores de texto normales, que son las siguientes:

- Existe mayor cantidad de diseños de texto profesionales a disposición, con los que realmente se pueden crear documentos como si fueran "de imprenta".
- Se facilita la composición de fórmulas con un cuidado especial.
- El usuario sólo necesita introducir instrucciones sencillas de entender con las que se indica la estructura del documento. Casi nunca hace falta preocuparse por los detalles de creación con técnicas de impresión.
- También las estructuras complejas como notas a pie de página, bibliografía, índices, tablas y muchas otras se pueden producir sin gran esfuerzo.
- Existen paquetes adicionales sin coste alguno para muchas tareas tipográficas que no se facilitan directamente por el LaTeX básico. Por ejemplo, existen paquetes para incluir gráficos en formato PostScript o para componer bibliografías conforme a determinadas normas. Muchos de estos paquetes se describen en *The LaTeX Companion* [?].
- LaTeX hace que los autores tiendan a escribir textos bien estructurados porque así es como trabaja LaTeX, o sea, indicando su estructura.
- TeX, la máquina de composición de LaTeX 2_ε, es altamente transportable y gratis. Por esto, el sistema funciona prácticamente en cualquier en cualquier plataforma.

LaTeX tiene, naturalmente, también inconvenientes:

- Para hacer funcionar un sistema de LaTeX, se necesitan más recursos (memoria, espacio de disco y potencia de procesamiento, y espacio de almacenamiento) que para un procesador de texto simple. Aunque no es menos cierto que las cosas van siendo cada vez mejores, y las distintas versiones que cada vez van apareciendo de *Microsoft Word* necesita cada vez más espacio de disco que un sistema de LaTeX normal. Cuando analizamos el uso del procesador, podemos ver que LaTeX supera en prestaciones cualquier sistema WYSIWYG ya que necesita mucha cantidad de CPU pero únicamente cuando el documento se procesa, mientras que los paquetes WYSIWYG tienen ocupada la CPU continuamente.
- Si bien se pueden ajustar algunos parámetros de un diseño de documento predefinido, la creación de un diseño entero es difícil y lleva mucho tiempo[2].

13.2.2. Ficheros de entrada de LaTeX

La entrada para LaTeX es un fichero de texto en formato `ASCII`. Se puede crear con cualquier editor de textos. Contiene tanto el texto que se debe imprimir como las "instrucciones", con las cuales se le puede indicar a LaTeX cómo debe disponer el texto.

[2] Los rumores dicen que este es uno de los puntos claves sobre el que se hará hincapié en el próximo sistema LaTeX 3.

Signos de espacio. Los caracteres "invisibles", como el espacio en blanco, el tabulador y el final de línea, son tratados por LaTeX como signos de espacio propiamente dichos. *Varios* espacios seguidos se tratan como *un* único espacio en blanco. Generalmente, un espacio en blanco al comienzo de una línea se ignora, y *varios* renglones en blanco se tratan como un renglón en blanco.

Un renglón en blanco entre dos líneas de texto definen el final de un párrafo. *Varias* líneas en blanco se tratan como *una sola* línea en blanco. El texto que mostramos a continuación es un ejemplo. A la derecha se encuentra el texto del fichero de entrada y a la izquierda la salida formateada.

No importa si introduces varios espacios tras una palabra.

Con una línea vacía se empieza un párrafo nuevo.

```
No importa si introduces
varios        espacios tras
una palabra.

Con una línea vacía se
empieza un párrafo nuevo.
```

Caracteres especiales. Los símbolos siguientes son caracteres reservados que tienen un significado especial para LaTeX o que no están disponibles en todos los tipos. Si los introduces en su fichero directamente es muy probable que no se impriman o que fuercen a LaTeX a hacer cosas que probablemente te parecerán extrañas.

```
$ & % # _ { } ~ ^ \
```

Como podemos ver, estos caracteres se pueden incluir en sus documentos anteponiendo el carácter \ (*barra invertida*):

$ & % # _ { }

```
\$ \& \% \# \_ \{ \}
```

Los restantes símbolos y otros muchos caracteres especiales se pueden imprimir en fórmulas matemáticas o como tildes con órdenes específicas.

Las órdenes de LaTeX. En las órdenes de LaTeX se distinguen las letras mayúsculas y las minúsculas. Toman uno de los dos formatos siguientes:

- Comienzan con una *barra invertida* \ y tienen un nombre compuesto sólo por letras. Los nombres de las órdenes acaban con uno o más espacios en blanco, un carácter especial o una cifra.
- Se compone de una *barra invertida* y un carácter especial.

LaTeX ignora los espacios en blanco que van tras las órdenes. Si deseas introducir un espacio en blanco tras una instrucción, se debe poner o bien `{}` y un espacio, o bien una instrucción de espaciado después de la orden. Con `{}` se fuerza a LaTeX a dejar de ignorar el resto de espacios que se encuentren después de la instrucción.

He leído que Knuth distingue a la gente que trabaja con TeX en TeXnicos y TeXpertos.
Hoy es 8 de septiembre de 2003.

```
He leído que Knuth distingue a la
gente que trabaja con \TeX{} en
\TeX{}nicos y \TeX pertos.\\
Hoy es \today.
```

Algunas instrucciones necesitan un parámetro que se debe poner entre llaves `{ }` tras la instrucción. Otras órdenes pueden llevar parámetros opcionales que se añaden a la instrucción entre corchetes `[ ]` o no. El siguiente ejemplo usa algunas órdenes de LaTeX que te explicaremos más adelante.

¡Te puedes *apoyar* en mí!

```
<Te puedes \textsl{apoyar} en mí!
```

¡Por favor, comienza una nueva línea justamente aquí! Gracias.

```
<Por favor, comienza una nueva
línea justamente aquí!%
\linebreak[3] Gracias.
```

Comentarios. Cuando LaTeX encuentra un carácter % mientras procesa un fichero de entrada, ignora el resto de la línea. Esto suele ser útil para introducir notas en el fichero de entrada que no se mostrarán en la versión impresa.

Esto es un ejemplo.

```
Esto es un % tonto
% Mejor: instructivo <----
ejemplo.
```

Esto a veces puede resultar útil cuando no quieras líneas demasiado largas en el fichero fuente. Si no quisieras introducir un espacio entre dos palabras, y prefieres tener dos renglones, entonces el signo % debe ir justo al final del renglón pero pegado al último carácter. De este modo se comenta el carácter de "salto de línea", que de otro modo se hubiese tratado como un espacio en blanco.

Este es otro ejemplo.

```
Este es otro ejem% y
% ahora el resto
plo.
```

13.2.3. Estructura de un fichero de entrada

Cuando LaTeX 2ε procesa un fichero de entrada, espera de éste que siga una determinada estructura. Todo fichero de entrada debe comenzar con la orden

```
\documentclass{...}
```

Esto indica qué tipo de documento se desea crear. Tras esto, se pueden incluir órdenes que influirán sobre el estilo del documento entero o puedes cargar paquetes con las que añadir nuevas propiedades al sistema de LaTeX. Para cargar uno de estos paquetes puedes emplear la instrucción

```
\usepackage{...}
```

Cuando todo el trabajo de configuración esté realizado[3] entonces comienza el cuerpo del texto con la instrucción

```
\begin{document}
```

A partir de entonces puedes introducir el texto mezclado con las instrucciones que consideremos necesarios de LaTeX. Al finalizar el documento debes poner la orden

```
\end{document}
```

LaTeX ignorará cualquier cosa que pongas tras esta instrucción.

La figura 13.1 muestra el contenido mínimo de un fichero de LaTeX 2ε. En la figura 13.2 se expone un fichero de entrada algo más complejo.

[3] El área entre `\documentclass` y `\begin{document}` se llama *preámbulo*.

```
\documentclass{article}
\begin{document}
Lo pequeño es bello.
\end{document}
```

Figura 13.1: Un fichero mínimo de LaTeX

13.2.4. El formato del documento

Clases de documentos. Cuando procesa un fichero de entrada, lo primero que necesita saber LaTeX es el tipo de documento que quieres crear. Esto se lo puedes indicar con la instrucción `\documentclass`.

`\documentclass[`*opciones*`]{`*clase*`}`

En este caso, la *clase* indica el tipo de documento que se creará. En la tabla 13.1 se muestran las clases de documento que explicaremos aquí. La distribución de LaTeX 2_ε proporciona más clases para otros documentos, como cartas y transparencias. El parámetro de *opciones* personaliza el comportamiento de la clase de documento elegida. Las opciones se deben separar con comas. En la tabla 13.2 se indican las opciones más comunes para las clases de documento estándares.

Por ejemplo: un fichero de entrada para un documento de LaTeX se podría comenzarlo con

```
\documentclass[11pt,twoside,a4paper]{article}
```

Con esto se le indica a LaTeX que componga el documento como un *artículo* utilizando tipos del cuerpo 11 —en términos más coloquiales, con fuente de 11 puntos— y que produzca un formato para impresión a *doble cara* en *papel DIN-A4*.

Paquetes Mientras e escribe el documento, probablemente nos encuentremos en situaciones donde el LaTeX básico no basta para realizar lo que buscamos. Si deseamos incluir gráficos, texto en color o el código fuente de un fichero, se necesita mejorar

```
\documentclass[a4paper,11pt]{article}
\usepackage{latexsym}
\usepackage[activeacute,spanish]{babel}
\author{T.~Bautista}
\title{Minimizando}
\frenchspacing
\begin{document}
\maketitle
\tableofcontents
\subsection{Inicio}
Bien\ldots{} y aquí comienza mi artículo tan
estupendo.
\subsection{Fin}
\ldots{} y aquí acaba.
\end{document}
```

Figura 13.2: Ejemplo para un artículo científico (en) español.

Tabla 13.1: Clases de documentos

`article`	para artículos de revistas especializadas, ponencias, trabajos de prácticas de formación, trabajos de seminarios, informes pequeños, solicitudes, dictámenes, descripciones de programas, invitaciones y muchos otros.
`report`	para informes mayores que constan de más de un capítulo, proyectos fin de carrera, tesis doctorales, libros pequeños, disertaciones, guiones y similares.
`book`	para libros de verdad
`slide`	para transparencias. Esta clase emplea tipos grandes sans serif.

Tabla 13.2: Opciones de clases de documento

`10pt`, `11pt`, `12pt` Establecen el tamaño (cuerpo) para los tipos[a]. Si no se especifica ninguna opción, se toma `10pt`.
`a4paper`, `letterpaper`, ... Definen el tamaño del papel. Si no se indica nada, se toma `letterpaper`. Aparte de éstos se puede elegir `a5paper`, `b5paper`, `executivepaper` y `legalpaper`.
`fleqn` Con esta opción las ecuaciones se disponen hacia la izquierda en vez de centradas.
`leqno` Coloca el número de las ecuaciones a la izquierda en vez de a la derecha.
`titlepage`, `notitlepage` Indica si se debe comenzar una página nueva tras el título del documento o no. Si no se indica otra cosa, la clase `article` no comienza una página nueva, mientras que `report` y `book` sí.
`twocolumn` Le dice a LaTeX que componga el documento en dos columnas.
`twoside, oneside` Especifica si se debe generar el documento a una o a dos caras. En caso de no indicarse otra cosa, las clases `article` y `report` son a una cara y la clase `book` es a dos.
`openright, openany` Hace que los capítulos comiencen o bien sólo en páginas a la derecha, o bien en la próxima que esté disponible. Esto no funciona con la clase `article`, ya que en esta clase no existen capítulos. De modo predeterminado, la clase `report` comienza los capítulos en la próxima página disponible y la clase `book` las inicia en las páginas a la derecha.

[a] El término adecuado para designar lo que normalmente se conoce como *fuente* es el de *tipo*

las capacidades de LaTeX. Tales mejoras se realizan con ayuda de los llamados *paquetes*. Los paquetes se activan con la orden

`\usepackage[`*opciones*`]{`*paquete*`}`

donde *paquete* es el nombre del paquete y *opciones* es una lista de palabras clave que activan funciones especiales del paquete, a las que LaTeX les añade las opciones que previamente hayamos podido indicar en la orden `\documentclass`. Algunos paquetes vienen con la distribución básica de LaTeX 2ε (véase la tabla 13.3). Otros se proporcionan por separado. En la Guía local (si existe) se puede encontrar más información sobre los paquetes disponibles en la instalación a la que se tenga acceso. Una buena fuente de información sobre LaTeX es *The LaTeX Companion* [?]. Contiene descripciones de cientos de paquetes, así como información sobre cómo escribir sus propias extensiones a LaTeX 2ε.

Tabla 13.3: Algunos paquetes distribuidos con LaTeX

doc Permite la documentación de paquetes y otros ficheros de LaTeX.
Se describe en `doc.dtx` y en *The LaTeX Companion* [?].

exscale Proporciona versiones escaladas de los tipos adicionales para matemáticas.
Descrito en `ltexscale.dtx`.

fontenc Especifica qué codificación de tipo debe usar LaTeX.
Descrito en `ltoutenc.dtx`.

ifthen Proporciona instrucciones de la forma
'si... entonces... si no...'
Descrito en `ifthen.dtx` y en *The LaTeX Companion* [?].

latexsym Para que LaTeX acceda al tipo de símbolos, se debe usar el paquete `latexsym`.
Descrito en `latexsym.dtx` y en *The LaTeX Companion* [?].

makeidx Proporciona instrucciones para producir índices de materias.
Descrito en el *The LaTeX Companion* [?].

syntonly Procesa un documento sin componerlo.
Se describe en `syntonly.dtx` y en *The LaTeX Companion* [?]. Es útil para la verificación rápida de errores.

inputenc Permite la especificación de una codificación de entrada como ASCII (con la opción ascii), ISO Latin-1 (con la opción latin1), ISO Latin-2 (con la opción latin2), páginas de código de 437/850 IBM (con las opciones cp437 y cp580, respectivamente), Apple Macintosh (con la opción applemac), Next (con la opción next), ANSI-Windows (con la opción ansinew) o una definida por el usuario. Está descrito en `inputenc.dtx`.

Estilo de página Con LaTeX existen tres combinaciones predefinidas de cabeceras y pies de página, a las que se llaman estilos de página. El parámetro *estilo* de la instrucción

`\pagestyle{`*estilo*`}`

define cuál queremos que LaTeX emplee. La tabla 13.4 muestra los estilos de página predefinidos.

Tabla 13.4: Estilos de página predefinidos en LaTeX

`plain`	imprime los números de página en el centro del pie de las páginas. Este es el estilo de página que se toma si no se indica otro.
`headings`	en la cabecera de cada página imprime el capítulo que se está procesando y el número de página, mientras que el pie está vacío.
`empty`	deja tanto la cabecera como el pie de las páginas vacíos.

Es posible cambiar el estilo de página de la página actual con la instrucción

`\thispagestyle{`*estilo*`}`

En *The LaTeX Companion* [**?**] existe una descripción de cómo crear nuetras propias cabeceras y pies de página[4].

13.2.5. Proyectos grandes

Cuando trabajes con documentos grandes, podrías, si lo consideramos oportuno, dividir el fichero de entrada en varias partes. LaTeX tiene dos instrucciones que te ayudan a realizar esto.

`\include{`*fichero*`}`

se puede utilizar en el cuerpo del documento para introducir el contenido de otro fichero. En este caso, LaTeX comenzará una página nueva antes de procesar el texto del *fichero*.

La segunda instrucción sólo se puede emplear en el preámbulo. Permite indicarle a LaTeX que sólo tome la entrada de algunos ficheros de los indicados con `\include`.

`\includeonly{`*fichero*`,`*fichero*`,...}`

Una vez se encuentre esta instrucción en el preámbulo del documento, sólo se procesarán las instrucciones `\include` con los ficheros indicados en el argumento de la orden `\includeonly`. Vigila que no hayan espacios entre los nombres de los ficheros y las comas.

13.3. Composición del texto

Tras leer este tema ya deberíamos conocer los elementos básicos de los que se compone un documento de LaTeX 2$_\varepsilon$. En este tema completaremos la estructura sobre la que normalmente se trabaja para componer documentos reales.

[4] Uno de los paquetes más socorridos para este tipo de cosas es `fancyhdr`. Lo más recomendable sería mirar su documentación, fundamentalmente la que existe en el fichero `fancyhdr.dtx`.

13.3.1. Salto de línea y de página

13.3.1.1. Párrafos justificados

Normalmente los libros se suelen componer con todos los renglones del mismo tamaño. LaTeX inserta los saltos de línea y los espacios entre las palabras optimizando el contenido de los párrafos enteros. Si es necesario, también introduce guiones, dividiendo las palabras que no encajen bien al final de los renglones. El modo de componer los párrafos depende de la clase de documento. Normalmente se introduce una sangría horizontal en la primera línea de un párrafo y no se introduce espacio adicional entre cada dos párrafos. Para más información véase el apartado 13.5.2.2.

En casos especiales se le puede ordenar a LaTeX que introduzca un salto de línea.

`\\ o \newline`

comienza una línea nueva sin comenzar un párrafo nuevo.

`\\*`

además prohíbe que se produzca un salto de página tras el salto de línea.

`\newpage`

comienza una página nueva.

`\linebreak[`*n*`]`, `\nolinebreak[`*n*`]`, `\pagebreak[`*n*`]` and `\nopagebreak[`*n*`]`

hacen lo que indican sus nombres: salto de línea, ningún salto de línea, salto de página y ningún salto de página. Además nos permite influir sobre las acciones a través del argumento opcional *n*. Se puede establecer a un valor entre cero y cuatro. Al poner *n* menor de 4 se le deja a LaTeX la posibilidad de ignorar la orden si el resultado resulta muy malo.

LaTeX siempre intenta realizar los saltos de línea lo mejor posible. Si no puede encontrar ninguna posibilidad satisfactoria para producir los bordes de los párrafos totalmente rectos, cumpliendo con las reglas impuestas, entonces dejará un renglón demasiado largo. En este caso LaTeX producirá el correspondiente mensaje de advertencia ("`overfull box`") mientras procesa el fichero de entrada. Esto sucede en especial si no se encuentra un lugar apropiado para introducir un guión de divisón entre las sílabas. Si se introduce la orden `\sloppy`, LaTeX será menos severo en sus exigencias y evita tales renglones con longitudes mayores, aumentando la separación entre las palabras —si bien el resultado final no es de lo mejor—. En este caso se dan mensajes de advertencia ("`underfull hbox`"). El resultado suele ser aceptable la mayoría de las veces. La orden `\fussy` actúa en sentido contrario. Esto podría hacerlo en caso que desee ver a LaTeX quejarse en todos los sitios.

13.3.1.2. Silabeo

LaTeX silabea las palabras cuando resulta necesario. Si el algoritmo de silabeo no produce los resultados correctos, entonces se puede remediar esta situación con órdenes como las que presentamos a continuación. Esto suele ser especialmente necesario en palabras compuestas o de idiomas extranjeros.

La instrucción

`\hyphenation{`*lista de palabras*`}`

da lugar a que las palabras mencionadas en ella se puedan dividir en cualquier momento únicamente en los lugares indicados con "-". Esta orden debería aparecer en el preámbulo del fichero de entrada y debería contener solamente palabras construidas sin caracteres especiales.

No se hacen distinciones entre las letras mayúsculas y minúsculas de las palabras a las que se refiera esta orden. El ejemplo siguiente permitirá localizar las sílabas de "fichero" y "Fichero" del mismo modo, e impedirá que en las palabras "FORTRAN", "Fortran" y "fortran" se introduzcan guiones. No se permiten caracteres con acentos o símbolos en el argumento.

Ejemplo:

```
\hyphenation{FORTRAN fi-che-ro}
```

Dentro de una palabra, la instrucción `\-` establece un sitio donde colocar un guión si fuese necesario. Además, éstos se convierten en los únicos lugares donde se permite introducir los guiones en esta palabra. Esta instrucción es especialmente útil para las palabras que contienten caracteres especiales (como, p.ej., los caracteres con acento ortográfico), ya que LaTeX no silabea de modo automático las palabras que contienen estos caracteres.

Me parece que esto es: supercalifragilisticoexpialidoso

```
Me parece que esto es: su\-per\-%
ca\-li\-fra\-gi\-lis\-ti\-co\-%
ex\-pia\-li\-do\-so
```

También se pueden mantener varias palabras en el mismo renglón con la orden

`\mbox{`*texto*`}`

Hace que su argumento se mantenga siempre unido bajo cualquier circunstancia, es decir, que no se puede dividir.

Dentro de poco tendré otro teléfono. Será el (040) 3783-225.

El parámetro *nombre de fichero* debe contener el nombre del fichero.

```
Dentro de poco tendré otro teléfono.
Será el \mbox{(040) 3783-225}.

El parámetro \mbox{\emph{nombre
de fichero}} debe contener el nombre
del fichero.
```

13.3.2. Caracteres especiales y símbolos

13.3.2.1. Comillas

Para las comillas no se debe utilizar el carácter de comillas que se usa en las máquinas de escribir. Para las publicaciones se suelen utilizar caracteres especiales, tanto para abrir como para cerrar comillas. En LaTeX se usan dos ` para abrir comillas y dos ' para cerrarlas.

"Por favor, pulse la tecla 'x.' "

```
``Por favor, pulse la tecla `x.'\,''
```

13.3.2.2. Guiones y rayas

LaTeX reconoce cuatro tipos de guiones. Para tener acceso a tres de éstos se pone una cantidad diferente de guiones consecutivos. El cuarto tipo es el signo matemático 'menos':

psico-terapéutico
10–18 horas
Madrid – Barcelona
¿Sí? —dijo ella—
0, 1 y −1

```
psico-terapéutico \\
10--18~horas \\
Madrid -- Barcelona \\
?`Sí? ---dijo ella--- \\
0, 1 y $-1$
```

13.3.2.3. Puntos suspensivos ('...')

En una máquina de escribir, tanto para la coma como para el punto se les da el mismo espaciado que a cualquier otro carácter. En la impresión de libros, estos caracteres sólo ocupan un pequeño espacio y se colocan muy próximos al carácter que les precede. Por eso, los "puntos suspensivos" no se pueden introducir con tres puntos normales, ya que no tendrían el espaciado correcto. Para estos puntos existe una instrucción especial llamada

`\ldots`

No así ... sino así:
New York, Tokyo, Budapest...

```
No así ... sino así:\\
New York, Tokyo, Budapest\ldots
```

13.3.2.4. Ligaduras

Algunas combinaciones de letras no se componen con las distintas letras que la forman, sino que, de hecho, se emplean símbolos especiales.

ff fi fl ffi... en lugar de ff fi fl ffi ...

Estas ligaduras se pueden evitar intercalando `\mbox{}` entre el par de letras en cuestión.

13.3.2.5. Tildes y caracteres especiales

LaTeX permite el uso de tildes y caracteres especiales de numerosos idiomas. En la tabla 13.5 puedes ver todos los tipos de tildes que se pueden aplicar a la letra *o*. Naturalmente, también funciona con otras letras.

Para colocar la tilde sobre una *i* o una *j* se debe eliminar el puntito superior de estas letras. Esto se consigue con las instrucciones `\i` y `\j`.

Hôtel, naïve, èlève,
smørrebrød, ¡Señorita!,
Schönbrunner Schloß Straße

```
H\^otel, na\"\i ve, \`el\`eve,\\
sm\o rrebr\o d, <Se\~norita!,\\
Sch\"onbrunner Schlo\ss{}
Stra\ss e
```

13.3.3. Facilidades para lenguajes internacionales

Si necesitamos escribir documentos en otros idiomas distintos del inglés, LaTeX debe utilizar otras reglas de silabeo para producir un resultado correcto.

Para muchos idiomas, estos cambios se pueden llevar a cabo utilizando el paquete babel de Johannes L. Braams. Para emplear este paquete, el sistema LaTeX debe estar configurado de modo especial. En la Guía local deberíamos encontrar más información sobre este particular.

Si tu sistema se configura de modo apropiado, entonces podrás activar el paquete babel con la instrucción

```
\usepackage[idioma]{babel}
```

tras la orden `\documentclass`. En la Guía local también debería aparecer un listado de los *idiomas* que acepta tu sistema.

Para algunos idiomas, babel también define nuevas instrucciones con las que se simplifica la entrada de caracteres especiales. En el idioma español, por ejemplo, se utilizan letras con acento ortográfico. Con babel y el estilo spanish, se puede introducir *í* con `'i` en vez de `\'{\i}`[5].

Además, con babel se vuelven a definir los títulos que producen algunas instrucciones de LaTeX, que normalmente son en inglés. Por ejemplo, si introduces la orden `\tableofcontents` aparecerá en el resultado final el índice del documento. Sin embargo, el título de este índice dependerá del idioma seleccionado ('*Table of contents*' si es inglés, '*Índice*' si es español, '*Inhaltverzeichnis*' si es alemán, etc.)

Con babel también se modifica la definición de la instrucción `\today` para que introduzca la fecha del día en el idioma elegido.

Algunos sistemas de ordenadores permiten introducir caracteres especiales directamente desde el teclado. LaTeX puede manejar esos caracteres. Desde la versión básica de LaTeX 2_ε de diciembre de 1994, se da la posibilidad de la utilización de diversos codificaciones de entrada. Para esta facilidad véase el paquete inputenc. Si usamos este paquete deberíamos considerar que otra gente puede no ser capaz de ver sus ficheros en su ordenador de forma correcta porque utilizan una codificación diferente. Por ejemplo, el símbolo alemán *ä* tiene en un PC el código 132 y en algunos sistemas Unix que emplean ISO-LATIN 1 tiene el código 228. Por lo tanto, se recomienda utilizar esta facilidad con sumo cuidado.

13.3.4. Distancias entre palabras

Para conseguir un margen derecho recto en la salida, LaTeX inserta cantidades variables de espacios entre las palabras. Al final de una oración, introduce unos espacios algo mayores que favorecen la legibilidad del texto. LaTeX presupone que las frases acaban con puntos, signos de interrogación y de admiración. Si hay un punto tras una letra mayúscula, entonces esto no se considera el fin de una oración ya que los puntos tras las letras mayúsculas normalmente se utilizan para abreviaturas.

Cualquier excepción a estas reglas deberemos indicarla. Una *barra invertida* \ antes de un espacio en blanco produce un espacio en blanco que no se ensanchará. Un carácter de tilde '~' genera un espacio que no se puede ensanchar pero en el que además no se puede producir ningún cambio de renglón. Si antes de un punto aparece la instrucción `\@`, significa que este punto acaba una oración, aunque se encuentre tras una letra mayúscula.

[5] En este caso particular de los acentos ortográficos, al paquete babel *también* debe pasársele la opción activeacute.

Tabla 13.5: Tildes y caracteres especiales

ò	`\‘o`	ó	`\’o`	ô	`\^o`	õ	`\~o`
ō	`\=o`	ȯ	`\.o`	ö	`\"o`		
ŏ	`\u o`	ǒ	`\v o`	ő	`\H o`	ǫ	`\c o`
ọ	`\d o`	o͟	`\b o`	o͡o	`\t oo`		
œ	`\oe`	Œ	`\OE`	æ	`\ae`	Æ	`\AE`
å	`\aa`	Å	`\AA`				
ø	`\o`	Ø	`\O`	ł	`\l`	Ł	`\L`
ı	`\i`	ȷ	`\j`	¡	`<`	¿	`?‘`

En la fig. 1 del cap. 1...
El Dr. López se encuentra
con Dña. Pérez.
... 5 m de ancho.
Necesito vitamina C. ¿Y tú?

```
En la fig.\ 1 del cap.\ 1\dots \\
El Dr.~López se encuentra \\
con Dña.~Pérez. \\
\dots\ 5~m de ancho. \\
Necesito vitamina~C\@. ?‘Y tú?
```

Este tratamiento especial para los espacios al final de las oraciones se puede evitar con la instrucción

`\frenchspacing`

que le indica a LaTeX que *no* inserte más espacios tras un punto que tras cualquier otro carácter. Esto es muy común en diversos idiomas, como es el caso del español. En este caso la instrucción `\@` no es necesaria.

13.3.5. Títulos, capítulos y apartados

Para ayudar al lector a seguir cómodamente el tema de su trabajo, se debería dividirlo en capítulos, apartados y subapartados. LaTeX lo facilita con instrucciones especiales que toman el título de la sección como su argumento. De nosotros depende emplearlos en el orden correcto.

Para la clase `article` existen las siguientes órdenes de seccionado:

```
\section{...}           \paragraph{...}
\subsection{...}        \subparagraph{...}
\paragraph{...}     \appendix
```

Con las clases `report` y `book` puedes utilizar dos instrucciones de seccionado adicionales:

```
\part{...}              \chapter{...}
```

Ya que la clase `article` no sabe de capítulos, es bastante sencillo añadir los artículos como capítulos de un libro. LaTeX coloca automáticamente el espaciado entre secciones, la numeración y los tipos de los títulos.

Dos de las instrucciones de seccionado son un poco especiales:

- La orden `\part` no influye en la secuencia de numeración de los capítulos.
- La orden `\appendix` no toma ningún argumento. Simplemente cambia la modo de numeración de los capítulos[6] a letras.

LaTeX crea un índice tomando las cabeceras de las distintas secciones y los números de página del último tratamiento del fichero de entrada. La instrucción

`\tableofcontents`

introduce este índice en el lugar donde coloquemos esta orden. Un documento nuevo se debe procesar dos veces para obtener un índice correcto. En algunos casos puede llegar a ser necesario compilar el documento una tercera vez. LaTeX nos lo indicará en caso necesario.

De todas las órdenes de seccionado que se han indicado también existen versiones modificadas, que se construyen añadiéndoles un asterisco `*` al nombre de la instrucción. Producen encabezados de sección que no aparecen en el índice y no se numeran. La instrucción `\subsection{Ayuda}` podría, por ejemplo, convertirse en `\subsection*{Ayuda}`.

Normalmente los encabezados de las secciones aparecen en el índice exactamente tal como los introducimos en el texto. En determinadas ocasiones esto no es posible porque el título es demasiado largo para caber en el índice. Entonces se puede especificar la entrada para el índice con un argumento opcional antes del encabezado real.

```
\chapter[<Léelo! Te gustará]{Esto es un título largo
                y que puede aburrir a mucha gente}
```

[6] Para el estilo de artículo lo que cambia es la forma de numerar los apartados.

El título de todo el documento se genera con la instrucción

`\maketitle`

El contenido del título se debe definir con las órdenes

`\title{...}`, `\author{...}` y opcionalmente `\date{...}`

antes de llamar a `\maketitle`. En el argumento de `\author` se pueden proporcionar varios nombres separados con la orden `\and`.

Un ejemplo de algunas de las instrucciones mencionadas las podemos encontrar en la fig. 13.2 de la página 188.

Además de las instrucciones de seccionado que se han indicado, LaTeX 2_ε inserta 3 instrucciones adicionales para su uso con la clase `book`:

`\frontmatter`, `\mainmatter` y `\backmatter`

Son útiles para dividir tu publicación. Estas instrucciones cambian los encabezados de los capítulos y la numeración de las páginas del mismo modo que en un libro normal.

13.3.6. Referencias cruzadas

En los libros, informes y artículos existen, a menudo, referencias cruzadas a figuras, tablas y segmentos especiales de texto que se hallan en otros lugares del documento. LaTeX proporciona las siguientes instrucciones para producir referencias cruzadas:

`\label{`*marcador*`}`, `\ref{`*marcador*`}` y `\pageref{`*marcador*`}`

donde *marcador* es un identificador que podemos usar en cualquier momento para hacer referencia al elemento al que lo asociemos. LaTeX reemplaza `\ref` por el número del apartado, subapartado, figura, tabla o teorema donde se insertó la instrucción `\label` correspondiente. La orden `\pageref` imprime el número de página donde se produce la orden `\label` con igual argumento. Aquí también se utilizan los números del procesamiento anterior.

Una referencia a este subapartado aparecería como:
"vea el apartado 13.3.6 en la página 197."

```
Una referencia a este subapartado
\label{sec:este} aparecería como:

``vea el apartado~\ref{sec:este} en
la página~\pageref{sec:este}.''
```

13.3.7. Notas a pie de página

Con la instrucción

`\footnote{`*texto de la nota al pie*`}`

se imprime una nota en el pie de la página actual.

Las notas a pie de página[a] son utilizadas con frecuencia por la gente que usa LaTeX.

[a] Esta es una nota a pie de página

```
Las notas a pie de página%
\footnote{Esta es una nota a pie
de página} son utilizadas con
frecuencia por la gente que usa
\LaTeX.
```

También existe una variante de esta instrucción, que es

`\footnote[`*número*`]{`*texto de la nota al pie*`}`

De esta forma para la nota al pie correspondiente se empleará para el marcador el *número* que se ha indicado en vez del valor del contador de notas al pie. Esta variante *sólo* se puede emplear dentro de los párrafos.

13.3.8. Palabras resaltadas

En los escritos a máquina, para resaltar determinados segmentos de texto normalmente se subrayan. En los libros impresos estas palabras se *resaltan* o se *destacan*. La orden para cambiar a un tipo de letra *resaltado* es

`\emph{`*texto*`}`

Su argumento es el texto que se debe resaltar.

Si está empleando resalte *en un texto ya resaltado, entonces LaTeX utiliza* redonda *para volver a resaltar texto.*

```
\emph{Si está empleando
\emph{resalte} en un texto
ya resaltado, entonces \LaTeX{}
utiliza \emph{redonda} para volver
a resaltar texto.}
```

13.3.9. Entornos

Para componer textos con un propósito especial LaTeX define muchos tipos de entornos para toda clase de diseños:

`\begin{`*nombre*`}` *texto* `\end{`*nombre*`}`

donde *nombre* es el nombre del entorno. Los entornos son "grupos" o "agrupaciones". También podemos cambiar a un nuevo entorno dentro de otro, en cuyo caso hay que prestar especial atención a la secuencia:

```
\begin{aaa}...\begin{bbb}...\end{bbb}...\end{aaa}
```

En los apartados siguientes explicaremos todos los entornos importantes.

13.3.9.1. Listas y descripciones (`itemize`, `enumerate`, `description`)

El entorno `itemize` es adecuado para las listas sencillas, el entorno `enumerate` para relaciones numeradas y el entorno `description` para descripciones.

1. Puedes mezclar los entornos de listas a tu gusto:
   - Pero podría comenzar a parecer incómodo.
   - Si abusa de ellas.
2. Por lo tanto, recuerda:

   Lo innecesario no va a resultar adecuado porque lo coloques en una lista.

   Lo adecuado, sin embargo, se puede presentar agradablemente en una lista.

```
\begin{enumerate}
\item Puedes mezclar los entornos
de listas a tu gusto:
\begin{itemize}
\item Pero podría comenzar a
parecer incómodo.
\item Si abusa de ellas.
\end{itemize}
\item Por lo tanto, recuerda:
\begin{description}
\item[Lo innecesario] no va a
resultar adecuado porque
lo coloques en una lista.
\item[Lo adecuado,] sin embargo,
se puede presentar agradablemente
en una lista.
\end{description}
\end{enumerate}
```

13.3.9.2. Justificaciones y centrado (`flushleft`, `flushright`, `center`)

Los entornos `flushleft` y `flushright` producen párrafos justificados a la izquierda y a la derecha (sin nivelación de bordes).

El entorno `center` genera texto centrado. Si no se introduce `\\` para dividir los renglones, entonces LaTeX lo hará automáticamente.

Este texto está
justificado a la izquierda. LaTeX no intenta forzar que todas las líneas tengan longitud.

```
\begin{flushleft}
Este texto está\\ justificado a
la izquierda. \LaTeX{} no intenta
forzar que todas las líneas
tengan longitud.
\end{flushleft}
```

Este texto está
justificado a la derecha. LaTeX no intenta forzar que todas las líneas tengan igual longitud.

```
\begin{flushright}
Este texto está\\ justificado a
la derecha. \LaTeX{} no intenta
forzar que todas las líneas
tengan igual longitud.
\end{flushright}
```

En el centro
de la tierra

```
\begin{center}
En el centro\\de la tierra
\end{center}
```

13.3.9.3. Citas (`quote`, `quotation`, `verse`)

El entorno `quote` sirve para citas pequeñas, ejemplos y para resaltar oraciones.

Una regla de oro en tipografía para la longitud de los renglones dice:

> Ningún renglón debe contener más de 66 letras.

Por esto se suelen utilizar varias columnas en los periódicos.

```
Una regla de oro en tipografía
para la longitud de los renglones
dice:
\begin{quote}
Ningún renglón debe contener
más de 66~letras.
\end{quote}
Por esto se suelen utilizar varias
columnas en los periódicos.
```

Hay dos entornos muy parecidos: el entorno `quotation` y el entorno `verse`. El entorno `quotation` es adecuado para citas mayores que consten de varios párrafos. El entorno `verse` es apropiado para poemas en los que la separación de los renglones es esencial. Los versos (los renglones) se dividen con `\\` y las estrofas con renglones en blanco.

Soberano gofio en polvo,
sustento de mi barriga,
el día que no te como
para mí no hay alegría.

```
\begin{flushleft}
\begin{verse}
Soberano gofio en polvo,\\
sustento de mi barriga,\\
el día que no te como\\
para mí no hay alegría.
\end{verse}
\end{flushleft}
```

13.3.9.4. Edición directa (verbatim, verb)

El texto que se coloque entre `\begin{verbatim}` y `\end{verbatim}` aparecerá tal como se ha escrito, redactado a partir de una máquina de escribir, con todos los espacios en blanco y cambios de línea y sin interpretación de las instrucciones de LaTeX.

Dentro de un párrafo se puede lograr el mismo efecto con

`\verb+`*text*`+`

El `+` sólo es un ejemplo de carácter delimitador. Podemos emplear cualquier carácter excepto las letras, `*` o caracteres en blanco.

La instrucción `\ldots`...

```
10 PRINT "HELLO WORLD ";
20 GOTO 10
```

```
La␣version␣con␣estrella␣del
entorno␣␣␣␣␣␣␣␣␣␣verbatim
destaca␣los␣espacios␣␣␣␣␣en
el␣␣␣texto␣con␣un␣símbolo
␣␣especial.
```

```
La instrucción \verb|\ldots|%
\ldots

\begin{verbatim}
10 PRINT "HELLO WORLD ";
20 GOTO 10
\end{verbatim}

\begin{verbatim*}
La version con estrella del
entorno          verbatim
destaca los espacios     en
el  texto con un símbolo
  especial.
\end{verbatim*}
```

La instrucción `\verb` se puede usar exactamente del mismo modo, con un asterisco:

```
%\verb*|de esta   manera :-) |
```

El entorno `verbatim` y la instrucción `\verb` no pueden utilizarse como parámetros de otras instrucciones.

13.3.9.5. Estadillos (tabular)

El entorno `tabular` sirve para crear estadillos, con líneas horizontales y verticales según cómo deseemos. LaTeX determina la anchura de las columnas de modo automático.

El argumento *especificaciones del estadillo* de la instrucción

`\begin{tabular}{`*especificaciones del estadillo*`}`

define el diseño del estadillo. Se puede utilizar `l` para una columna con texto justificado a la izquierda, `r` para justificar el texto a la derecha, `c` para texto centrado, `p{`*ancho*`}` para una columna que contenga texto con saltos de línea y `|` para una línea vertical.

Dentro de un entorno `tabular`, `&` salta a la próxima columna, `\\` separa los renglones y `\hline` introduce una línea horizontal.

Cantidad	Base
7C0	hexadecimal
3700	octal
11111000000	binario
1984	decimal

```
\begin{tabular}{|r|l|}
\hline
Cantidad & Base \\
\hline
\hline
7C0 & hexadecimal \\
3700 & octal \\
11111000000 & binario \\
\hline \hline
1984 & decimal \\
\hline
\end{tabular}
```

Bienvenido al párrafo del Sr. Cajón. Esperamos que disfrute del espectáculo.

```
\begin{tabular}{|p{4.7cm}|}
\hline
Bienvenido al párrafo del Sr.\
Cajón. Esperamos que disfrute
del espectáculo.\\
\hline
\end{tabular}
```

Con la construcción `@{...}` se puede especificar el separador de columnas. Esta construcción elimina el espacio entre columnas y lo reemplaza con lo que hayamos introducido entre los paréntesis. Un uso muy frecuente de esta construcción se explica más adelante con el problema de la alineación de la coma decimal. Otra posible utilización es para eliminar el espacio que antecede y precede a los renglones de una tabla con `@{}`.

ningún espacio a la izquierda ni derecha

```
\begin{tabular}{@{} l @{}}
\hline
ningún espacio a la izquierda
ni derecha\\\hline
\end{tabular}
```

espacios a la izquierda y a la derecha

```
\begin{tabular}{l}
\hline
espacios a la izquierda
y a la derecha\\
\hline
\end{tabular}
```

incorporado para alinear columnas numéricas sobre la coma decimal [7], se podría "imitarlo" utilizando dos columnas: un entero alineado a la derecha y luego los decimales a la izquierda. La instrucción `@{,}` en el argumento de `\begin{tabular}` reemplaza el espacio normal entre columnas con una ",", dando la apariencia de una única columna justificada por la coma decimal. ¡No debemos olvidarnos de reemplazar la coma decimal en sus números con un separador de columna (`&`)! Se puede colocar una etiqueta sobre nuestra "columna" numérica empleando la instrucción `\multicolumn`.

Expresión en pi	Valor
π	3,1416
π^{π}	36,46
$(\pi^{\pi})^{\pi}$	80662,7

```
\begin{tabular}{cr@{,}l}
Expresión en pi & \multicolumn{2}{c}{Valor} \\
\hline
$\pi$ & 3&1416 \\
$\pi^{\pi}$ &
36&46 \\
$(\pi^{\pi})^{\pi}$ & 80662&7
\\ \end{tabular}
```

13.3.10. Elementos flotantes

Hoy en día, la mayoría de las publicaciones contienen muchas ilustraciones y tablas. Estos elementos necesitan un tratamiento especial porque no se pueden cortar entre páginas. Un método podría ser comenzando una página nueva cada vez que una ilustración o una tabla sea demasiado larga para caber en la página actual. Este enfoque deja páginas parcialmente vacías, lo que resulta poco estético.

La solución a este problema es hacer que cualquier ilustración o tabla que no quepa en la página actual 'flote' hasta una página posterior mientras se rellena la página actual con el texto del documento.

LaTeX ofrece dos entornos para los elementos flotantes. Uno para las tablas y otro para las ilustraciones. Para aprovechar completamente estos dos entornos es importante entender aproximadamente cómo maneja LaTeX estos objetos flotantes internamente. Si no, los objetos flotantes se pueden convertir en una fuente de frustaciones porque LaTeX nunca los pone dónde se desea situar.

Primeramente, echemos un vistazo a las instrucciones que LaTeX proporciona para objetos flotantes.

[7] Si se halla instalado el conjunto 'tools' en su sistema, sería recomendable echar un vistazo al paquete dcolumn.

Cualquier cosa que se incluya en un entorno `figure` o `table` se tratará como un objeto flotante. Ambos entornos flotantes proporcionan un parámetro opcional

`\begin{figure}[`*designador de colocado*`]` o
`\begin{table}[`*designador de colocado*`]`

llamado el *designador de colocado*. Este parámetro se emplea para indicarle a LaTeX los lugares donde le permitimos que vaya colocado el objeto flotante. Un *designador de colocado* se construye con una cadena de *permisos de colocación flotante*. Los podemos encontrar en la tabla 13.6.

Una tabla se podría comenzar con, por ejemplo, la siguiente línea:

```
\begin{table}[!hbp]
```

El designador de colocado `[!hbp]` le permite a LaTeX colocar la tabla justamente aquí (`h`) o al final (`b`) de alguna página o en alguna página especial para elementos flotantes, y en cualquier parte si no queda tan bien (`!`). Si no indicamos ningún designador de colocado, entonces se sobreentienden las clases normalizadas `[tbp]`.

LaTeX colocará todos los objetos flotantes que encuentre según los designadores de colocado que hayamos indicado. Si un objeto flotante no se puede colocar en la página actual entonces se aplaza su colocación, para lo cual se introduce en una cola[8] de *tablas* o *figuras* (ilustraciones). Cuando se comienza una nueva página, lo primero que LaTeX hace es confirmar si se puede construir una página especial con los objetos flotantes que se hallan en las colas. Si no es posible, entonces se trata el primer objeto que se encuentra en las colas como si lo acabásemos de introducir. Entonces LaTeX vuelve a intentar colocar el objeto según sus designadores de colocado (eso sí, sin tener en cuenta la opción 'h', que ya no es posible). Cualquier objeto flotante nuevo que aparezca en el texto se introduce en la cola correspondiente. LaTeX mantiene estrictamente el orden original de apariciones de cada tipo de objeto flotante.

Ésta es la razón por la que una ilustración que no se puede colocar ya que desplaza al resto de las figuras al final del documento. Por lo tanto:

> Si LaTeX no coloca los objetos flotantes como esperaba, suele deberse únicamente a un objeto flotante que está atascando una de las dos colas de objetos flotantes.

Además, existen algunas cosas más que se deben indicar sobre los entornos `table` y `figure`. Con la instrucción

`\caption{`*texto de título*`}`

se puede definir un título para el objeto flotante. LaTeX le añadirá la cadena "Figura" o "Tabla" y un número de secuencia.

Las dos instrucciones

`\listoffigures` y `\listoftables`

funcionan de modo análogo a la orden `\tableofcontents`, imprimiendo un índice de figuras o de tablas respectivamente. En estas listas se repetirán los títulos completos. Debeimos tener presente que si tendemos a utilizar títulos largos, es recomendable tener una versión de estos títulos más cortos para utilizarlos para estos índices. Esto se consigue dando la versión corta entre corchetes tras la orden `\caption`.

[8] Son de tipo *fifo*: lo que se introdujo primero es lo primero en extraerse.

Tabla 13.6: Permisos de colocación flotante

Designador	Permiso para colocar el objeto flotante...
`h`	aquí (*here*), muy próximo al lugar en el texto donde se ha introducido. Es útil, principalmente, para objetos flotantes pequeños.
`t`	en la parte superior de una página (*top*).
`b`	en la parte inferior de una página (*bottom*).
`p`	en una *página* especial que sólo contenga elementos flotantes.
`!`	sin considerar la mayoría de los parámetros internos[a] que impedirían a este objeto flotante que se colocase.

[a] Como el número máximo de elementos flotantes en una página.

```
\caption[Corto]{LLLLLaaaaaaaaarrrrrrrrgggggooooooo}
```

Con `\label` y `\ref` se pueden crear referencias a un objeto flotante dentro del texto.

El siguiente ejemplo dibuja un cuadrado y lo inserta en el documento. Podría utilizar esto si desea reservar espacios para imágenes que vaya a pegar en el documento acabado.

```
La ilustración~\ref{blanco} es un ejemplo del Pop-Art.
\begin{figure}[!hbp]
\makebox[\textwidth]{\framebox[5cm]{\rule{0pt}{5cm}}}
\caption{$5\times 5$ centímetros} \label{blanco}
\end{figure}
```

En el ejemplo anterior[9] LaTeX intentará *por todos los medios* (`!`) colocar la ilustración exactamente *aquí* (`h`). Si no puede, intentará colocarla en la *parte inferior* (`b`) de la página. Si no consigue colocar esta figura en la página actual, determina si es posible crear una página (`p`) con elementos flotantes exclusivamente que contenga esta ilustración y algunas tablas que pudieran haber en la cola de tablas. Si no hay material suficiente para una página especial de objetos flotantes, entonces LaTeX comienza una página nueva y otra vez trata la figura como si acabase de aparecer en el texto.

Bajo determinadas condiciones podría ser necesario emplear la orden

`\clearpage`

Le ordena a LaTeX que coloque *inmediatamente* todos los objetos flotantes que se encuentren en las colas y después comenzar una página nueva.

13.3.11. Añadiendo instrucciones y entornos nuevos

En el primer apartado explicamos que LaTeX necesita información sobre la estructura lógica del texto para elegir el formato adecuado. Éste es un concepto muy bien cuidado. Pero en la práctica solemos chocar con las limitaciones que esto nos impone, ya que LaTeX simplemente no tiene el entorno especializado o la orden que deseamos para un propósito específico.

Una solución es emplear varias órdenes de LaTeX para producir el diseño que se tiene en mente. Si tenemos que hacer esto una vez, no hay ningún problema. Pero si esto sucede repetidamente, entonces lleva mucho tiempo. Si alguna vez deseáramos cambiar el formato tendríamos que revisar el fichero de entrada entero y editar todos los elementos en cuestión.

Para resolver este problema, LaTeX nos permite definir nuestras propias instrucciones y entornos.

13.3.11.1. Instrucciones nuevas

Para añadir nuestras propias instrucciones utilizaremos la orden

`\newcommand{`*nombre*`}[`*num*`]{`*definición*`}`

Básicamente, la instrucción necesita dos argumentos: el *nombre* de la instrucción que queremos crear y la *definición* de la instrucción. El argumento entre corchetes *num* es opcional. Podemos usarlo para crear órdenes nuevas que tomen hasta 9 argumentos.

Los dos ejemplos siguientes deberían ayudar a captar la idea. El primer ejemplo define una instrucción nueva llamada `\udl`. Ésta es una forma abreviada de introducir "Una Descripción de LaTeX 2ε". Una orden como ésta sería muy útil si tuviéramos que escribir el título de este tema una y otra vez.

"Una Descripción de LaTeX 2ε" ... "Una Descripción de LaTeX 2ε"

```
\newcommand{\udl}
    {Una Descripción de \LaTeXe}
% en el cuerpo del documento :
``\udl'' \ldots{} ``\udl''
```

El ejemplo siguiente ilustra cómo usar el argumento *num*. La secuencia `#1` encuentra un sustituto en el argumento que especifiquemos. Si quisiéramos más de un argumento, emplearemos `#2` y así sucesivamente.

[9] suponiendo que la cola de figuras esté vacía.

- Una Descripción *no tan* Pequeña de LaTeX 2ε
- Una Descripción *muy* Pequeña de LaTeX 2ε

```
\newcommand{\txsit}[1]
    {Una Descripción \emph{#1}
     Pequeña de \LaTeXe}
% en el cuerpo del documento:
\begin{itemize}
\item \txsit{no tan}
\item \txsit{muy}
\end{itemize}
```

LaTeX no nos permitirá crear una instrucción nueva con un nombre que ya existe. Si queremos ignorar de modo explícito una instrucción existente debemos utilizar `\renewcommand`. Aparte de su nombre, utiliza la misma sintaxis que la instrucción `\newcommand`. En determinados casos podríamos utilizar la instrucción `\providecommand`. Funciona como `\newcommand`, pero si ya hay una instrucción definida con este nombre, entonces LaTeX 2ε simplemente ignora esta otra definición que acabáramos de indicar.

13.3.11.2. Entornos nuevos

De modo análogo a la instrucción `\newcommand` existe una orden para crear tus propios entornos. Cuando estábamos escribiendo este tema, hemos creado entornos especiales para estructuras que se empleaban repetidamente en toda la descripción: "ejemplos", "segmentos de código" y "cajas de definición de instrucciones". La instrucción `\newenvironment` utiliza la siguiente sintaxis:

`\newenvironment{`*nombre*`}[`*num*`]{`*antes*`}{`*después*`}`

Al igual que la instrucción `\newcommand`, se puede usar `\newenvironment` con o sin argumento opcional. Lo que especifiquemos en el argumento *antes* se procesa antes que el texto dentro del entorno. Lo que indiquemos en el argumento *después* se procesa cuando se encuentra la instrucción `\end{`*nombre*`}`.

El siguiente ejemplo ilustra el uso de la instrucción `\newenvironment`.

Mis humildes vasallos...

```
\newenvironment{king}
    {\begin{quote}}{\end{quote}}
% use esto en el cuerpo
\begin{king}
Mis humildes vasallos\ldots
\end{king}
```

El argumento *num* se utiliza igual que la instrucción `\newcommand`. LaTeX se asegura de que no definamos un entorno que ya existía. Si alguna vez deseamos cambiar una entorno existente, entonces podemos utilizar la instrucción `\renewenvironment`. Tiene la misma sintaxis que la instrucción `\newenvironment`.

13.4. Composición de fórmulas matemáticas

¡Ahora a admirarse con este apartado! Vamos a abordar el punto fuerte de TeX: la composición matemática. Pero advertimos que este apartado sólo muestra la superficie. Mientras lo que aquí explicamos es suficiente para mucha gente, no hay que desesperarase si no se puede encontrar una solución a nuestras necesidades de composición de expresiones matemáticas. Es muy probable que en esos casos nuestro problema ya esté abordado en AMS-LaTeX 2ε[10] o en algún otro paquete.

13.4.1. Generalidades

LaTeX posee un modo especial para componer expresiones matemáticas. En un párrafo, el texto matemático se introduce entre `\(` y `\)`, entre `$` y `$` o entre `\begin{math}` y `\end{math}`.

[10] `CTAN:/tex-archive/macros/latex/packages/amslatex`

Siendo a y b los catetos y c la hipótenusa de un triángulo rectángulo, entonces $c^2 = a^2 + b^2$ (Teorema de Pitágoras).

```
Siendo $a$ y $b$ los catetos
y $c$ la hipótenusa
de un triángulo rectángulo,
entonces $c^{2}=a^{2}+b^{2}$
(Teorema de Pitágoras).
```

TEX se pronuncia como $\tau\epsilon\chi$.

100 m^2 de área útil

De mi ♡.

```
\TeX{} se pronuncia como
 $\tau\epsilon\chi$.\\[6pt]
100~m$^{2}$ de área útil \\[6pt]
De mi $\heartsuit$.
```

Las fórmulas matemáticas mayores o las ecuaciones suelen quedar mejor en renglones separados del texto. Para ello se ponen entre `\[` y `\]` o entre `\begin{displaymath}` y `\end{displaymath}`. Esto produce fórmulas sin número de ecuación. Si quieres que LaTeX las enumere, entonces puedes emplear el entorno `equation`.

Siendo a y b los catetos y c la hipótenusa de un triángulo rectángulo, entonces

$$c = \sqrt{a^2 + b^2}$$

(Teorema de Pitágoras).

```
Siendo $a$ y $b$ los catetos
y $c$ la hipótenusa
de un triángulo rectángulo,
entonces
\begin{displaymath}
c = \sqrt{  a^{2}+b^{2}  }
\end{displaymath}
(Teorema de Pitágoras).
```

Con `\label` y `\ref` se puede hacer referencia a una ecuación del documento.

$$\epsilon > 0 \tag{13.1}$$

De (13.1) se deduce...

```
\begin{equation} \label{eq:eps}
\epsilon > 0
\end{equation}
De (\ref{eq:eps}) se deduce\ldots
```

Fíjate que las expresiones se componen con un estilo diferente al disponerlas en párrafos separados del texto:

$\lim_{n \to \infty} \sum_{k=1}^{n} \frac{1}{k^2} = \frac{\pi^2}{6}$

```
$\lim_{n \to \infty}
\sum_{k=1}^n \frac{1}{k^2}
= \frac{\pi^2}{6}$
```

$$\lim_{n \to \infty} \sum_{k=1}^{n} \frac{1}{k^2} = \frac{\pi^2}{6}$$

```
\begin{displaymath}
\lim_{n \to \infty}
\sum_{k=1}^n \frac{1}{k^2}
= \frac{\pi^2}{6}
\end{displaymath}
```

Existen diferencias entre el *modo matemático* y el *modo texto*. Por ejemplo, en el *modo matemático*:

1. Los espacios en blanco y los cambios de línea no tienen ningún significado. Todos los espacios se determinarán a partir de la lógica de la expresión matemática o se deben indicar con instrucciones especiales como `\,`, `\quad`, `\qquad`, `\:`, `\;`, `\ ` y `\!`.

$$\forall x \in \mathbf{R}: \qquad x^2 \geq 0 \tag{13.2}$$

```
\begin{equation}
\forall x \in \mathbf{R}:
\qquad x^{2} \geq 0
\end{equation}
```

2. Los renglones en blanco están prohibidos. Sólo puede haber un párrafo por fórmula.

3. Cada letra en particular se tendrá en cuenta como el nombre de una variable y se pondrá como tal (cursiva con espacios adicionales). Para introducir texto normal dentro de un texto matemático (con escritura en redondilla y con espacios entre palabras) debes incluirlo dentro de la orden `\textrm{...}`.

$$x^2 \geq 0 \qquad \text{para todo } x \in \mathbf{R} \tag{13.3}$$

```
\begin{equation}
x^{2} \geq 0\qquad
\textrm{para todo }x\in\mathbf{R}
\end{equation}
```

Los matemáticos pueden ser muy exigentes con los símbolos que se emplean: aquí sería más *convencional* emplear '*blackboard bold*' que se obtienen con `\mathbb` del paquete amsfonts o amssymb. El último ejemplo se convierte en

$$x^2 \geq 0 \qquad \text{para todo } x \in \mathbb{R}$$

```
\begin{displaymath}
x^{2} \geq 0\qquad
\textrm{para todo }x\in\mathbb{R}
\end{displaymath}
```

13.4.2. Agrupaciones en modo matemático

En modo matemático la mayoría de las instrucciones sólo afecta al carácter siguiente. Si se desea que una instrucción influya sobre varios caracteres, entonces debes agruparlos empleando llaves (`{...}`).

$$a^{x} + y \neq a^{x+y} \tag{13.4}$$

```
\begin{equation}
a^x+y \neq a^{x+y}
\end{equation}
```

13.4.3. Elementos de las fórmulas matemáticas

A continuación presentamos las instrucciones más importantes que se utilizan en las fórmulas matemáticas.

Las letras griegas minúsculas se introducen como `\alpha`, `\beta`, `\gamma`..., y las mayúsculas[11] se introducen como `\Gamma`, `\Delta`...

$\lambda, \xi, \pi, \mu, \Phi, \Omega$

```
$\lambda,\xi,\pi,\mu,\Phi,\Omega$
```

Los exponentes y los subíndices se pueden indicar empleando el carácter `^` y el carácter `_`.

$a_1 \qquad x^2 \qquad e^{-\alpha t} \qquad a^3_{ij}$
$e^{x^2} \neq e^{x^2}$

```
$a_{1}$ \qquad $x^{2}$ \qquad
$e^{-\alpha t}$ \qquad
$a^{3}_{ij}$\\
$e^{x^2} \neq {e^x}^2$
```

El **signo de raíz cuadrada** se introduce con `\sqrt`, y la raíz n-ésima con `\sqrt[`n`]`. LaTeX elige automáticamente el tamaño del signo de raíz. Si sólo necesitamos el signo de la raíz utiliza `\surd`.

$\sqrt{x} \qquad \sqrt{x^2+\sqrt{y}} \qquad \sqrt[3]{2}$
$\surd[x^2 + y^2]$

```
$\sqrt{x}$ \qquad
$\sqrt{ x^{2}+\sqrt{y} }$
\qquad $\sqrt[3]{2}$\\[3pt]
$\surd[x^2 + y^2]$
```

Las instrucciones `\overline` y `\underline` producen **líneas horizontales** directamente encima o debajo de una expresión.

$\overline{m+n}$

```
$\overline{m+n}$
```

Las órdenes `\overbrace` y `\underbrace` crean **llaves horizontales** largas encima o bien debajo de una expresión.

[11] No hay definida ninguna Alfa mayúscula en LaTeX 2_ε porque tiene el mismo aspecto que la redondilla A. Una vez que se haga la nueva codificación matemática, esto cambiará.

$\underbrace{a+b+\cdots+z}_{26}$

```
$\underbrace{ a+b+\cdots+z }_{26}$
```

Para poner acentos matemáticos, como pequeñas flechas o tildes a las variables, se pueden utilizar las órdenes que aparecen en la tabla 13.13. Los ángulos y las tildes que abarcan varios caracteres se obtienen con `\widetilde` y `\widehat`. Con el símbolo `'` se introduce el signo de prima.

$y = x^2 \qquad y' = 2x \qquad y'' = 2$

```
\begin{displaymath}
y=x^{2}\qquad y'=2x\qquad y''=2
\end{displaymath}
```

Tabla 13.7: Acentos en modo matemático

$\hat{a}$	`\hat{a}`	$\check{a}$	`\check{a}`	$\tilde{a}$	`\tilde{a}`	$\acute{a}$	`\acute{a}`
$\grave{a}$	`\grave{a}`	$\dot{a}$	`\dot{a}`	$\ddot{a}$	`\ddot{a}`	$\breve{a}$	`\breve{a}`
$\bar{a}$	`\bar{a}`	$\vec{a}$	`\vec{a}`	$\widehat{A}$	`\widehat{A}`	$\widetilde{A}$	`\widetilde{A}`

Con frecuencia los **vectores** se indican añadiéndoles símbolos de flecha pequeños encima de la variable. Esto se realiza con la orden `\vec`. Para designar al vector que va desde A hasta B resultan adecuadas las instrucciones `\overrightarrow` y `\overleftarrow`.

$\vec{a} \quad \overrightarrow{AB}$

```
\begin{displaymath}
\vec a\quad\overrightarrow{AB}
\end{displaymath}
```

Existen funciones matemáticas (seno, coseno, tangente, logaritmos...) que se presentan con redondilla y *nunca* en itálica. Para éstas, LaTeX proporciona las siguientes instrucciones:

```
\arccos  \cos   \csc  \exp  \ker    \limsup  \min  \sinh
\arcsin  \cosh  \deg  \gcd  \lg     \ln       \Pr   \sup
\arctan  \cot   \det  \hom  \lim    \log      \sec  \tan
\arg     \coth  \dim  \inf  \liminf  \max      \sin  \tanh
```

$$\lim_{n \rightarrow 0} \frac{\sin x}{x} = 1$$

```
\[\lim_{n \rightarrow 0}
\frac{\sin x}{x}=1\]
```

Para la función módulo existen dos órdenes distintas: `\bmod` para el operador binario, como en "a mód b", y `\pmod` para expresiones como "$x \equiv a \pmod{b}$".

Un **quebrado** o **fracción** se pone con la orden `\frac{...}{...}`. Para los quebrados sencillos a veces suele ser preferible utilizar el operador `/`,

$1\frac{1}{2}$ horas

$\frac{x^2}{k+1} \qquad x^{\frac{2}{k+1}} \qquad x^{1/2}$

```
$1\frac{1}{2}$~horas
\begin{displaymath}
\frac{ x^2 }{ k+1 }\qquad
x^{ \frac{2}{k+1} }\qquad
x^{ 1/2 }
\end{displaymath}
```

Los **coeficientes de los binomios** y estructuras similares se pueden componer con la instrucción `{... \choose...}`— o `{... \atop ...}`. Con la segunda orden se consigue lo mismo pero sin paréntesis.

$\binom{n}{k} \qquad \begin{matrix} x \\ y+2 \end{matrix}$

```
\begin{displaymath}
{n \choose k}\qquad {x \atop y+2}
\end{displaymath}
```

El **signo de integral** se obtiene con `\int` y el **signo de sumatorio** con `\sum`. Los límites superior e inferior se indican con `^` y `_`, tal como se hace para los superíndices y subíndices.

$$\sum_{i=1}^{n} \qquad \int_0^{\frac{\pi}{2}}$$

```
\begin{displaymath}
\sum_{i=1}^{n} \qquad
\int_{0}^{\frac{\pi}{2}} \qquad
\end{displaymath}
```

Para las **llaves** y otros delimitadores tenemos todos los tipos de símbolos de TEX (p. ej. [⟨ ‖ ↕). Los paréntesis y los corchetes se introducen con las teclas correspondientes, las llaves con \{ y \}, y el resto con instrucciones especiales (p. ej. \updownarrow).

$$a, b, c \neq \{a, b, c\}$$

```
\begin{displaymath}
{a,b,c}\neq\{a,b,c\}
\end{displaymath}
```

Para que LaTeX elija de modo automático el tamaño apropiado se pone la orden \left delante del delimitador de apertura y \right delante del que cierra. Debemos tener en cuenta que debemos cerrar cada \left con el \right correspondiente. Si no deseas nada en la derecha, entonces emplea '\right.'.

$$1 + \left(\frac{1}{1 - x^2} \right)^3$$

```
\begin{displaymath}
1 + \left( \frac{1}{ 1-x^{2} }
    \right) ^3
\end{displaymath}
```

En algunos casos es necesario fijar de modo explícito el tamaño correcto del delimitador matemático. Para esto se pueden utilizar las instrucciones \big, \Big, \bigg y \Bigg como prefijos de la mayoría de las órdenes de delimitadores[12].

$$\Big((x+1) (x-1) \Big) ^{2}$$
$$\big(\Big(\bigg(\Bigg($$
$$\big\}\Big\}\bigg\}\Bigg\}$$
$$\big\|\Big\|\bigg\|\Bigg\|$$

```
$\Big( (x+1) (x-1) \Big) ^{2}$\\
$\big(\Big(\bigg(\Bigg($\quad
$\big\}\Big\}\bigg\}\Bigg\}$\quad
$\big\|\Big\|\bigg\|\Bigg\|$
```

Para poner los **puntos suspensivos** en una ecuación existen varias órdenes. \ldots coloca los puntos en la línea base y \cdots los pone en la zona media del renglón. Ademas de éstos, también existen las instrucciones \vdots para puntos verticales y \ddots para puntos en diagonal.

En el apartado 13.4.5 existe otro ejemplo.

$$x_1, \ldots, x_n \qquad x_1 + \cdots + x_n$$

```
\begin{displaymath}
x_{1},\ldots,x_{n} \qquad
x_{1}+\cdots+x_{n}
\end{displaymath}
```

13.4.4. Espaciado en modo matemático

Si no estamos satisfechos con los espaciados que TEX elige dentro de una fórmula, éstos se pueden alterar con instrucciones especiales. Las más importantes son \, para un espacio muy pequeño, carácter en blanco), \quad y \qquad para espaciados grandes y \! para la disminución de una separación.

$$\iint_D g(x, y)\,\mathrm{d}x\,\mathrm{d}y$$

en lugar de

$$\int\int_D g(x, y)\mathrm{d}x\mathrm{d}y$$

```
\newcommand{\rd}{\mathrm{d}}
\begin{displaymath}
\int\!\!\!\int_{D} g(x,y)
  \, \rd x\, \rd y
\end{displaymath}
en lugar de
\begin{displaymath}
\int\int_{D} g(x,y)\rd x \rd y
\end{displaymath}
```

[12] Estas instrucciones pueden no funcionar del modo deseado si hemos utilizado una instrucción de cambio del tamaño del tipo, o si se ha especificado la opción 11pt o 12pt. Los paquetes exscale o amstex sirven para corregir esta anomalía.

Observemos que la 'd' en la diferencial se compone de modo convencional en redondilla[13].

13.4.5. Colocación de signos encima de otros

Para componer **matrices** y similares se tiene el entorno `array`. Éste funciona de modo similar al entorno `tabular`. Para dividir los renglones se utiliza la instrucción `\\`.

$$\mathbf{X} = \left(\begin{array}{ccc} x_{11} & x_{12} & \ldots \\ x_{21} & x_{22} & \ldots \\ \vdots & \vdots & \ddots \end{array} \right)$$

```
\begin{displaymath}
\mathbf{X} =
\left( \begin{array}{ccc}
x_{11} & x_{12} & \ldots \\
x_{21} & x_{22} & \ldots \\
\vdots & \vdots & \ddots
\end{array} \right)
\end{displaymath}
```

También se puede usar el entorno `array` para componer expresiones de funciones que tienen "." como delimitador invisible derecho, es decir, `\right.`.

$$y = \left\{ \begin{array}{ll} a & \text{si } d > c \\ b + x & \text{por la mañana} \\ l & \text{el resto del día} \end{array} \right.$$

```
\begin{displaymath}
y = \left\{ \begin{array}{ll}
 a & \textrm{si $d>c$}\\
 b+x & \textrm{por la mañana}\\
 l & \textrm{el resto del día}
  \end{array} \right.
\end{displaymath}
```

Para las ecuaciones que ocupen varios renglones o para los sistemas de ecuaciones se pueden emplear los entornos `eqnarray` y `eqnarray*`. En `eqnarray` cada renglón contiene un número de ecuación. Con `eqnarray*` no se produce ninguna numeración.

Los entornos `eqnarray` y `eqnarray*` funcionan como una tabla de 3 columnas con la disposición `{rcl}`, donde la columna central se utiliza para el signo de igualdad, desigualdad o cualquier otro signo que deba ir. La instrucción `\\` divide los renglones.

$$\begin{aligned} f(x) &= \cos x && (13.5) \\ f'(x) &= -\sin x && (13.6) \\ \int_0^x f(y)\mathrm{d}y &= \sin x && (13.7) \end{aligned}$$

```
\begin{eqnarray}
f(x) & = & \cos x       \\
f'(x) & = & -\sin x     \\
\int_{0}^{x} f(y) \mathrm{d}y &
 = & \sin x
\end{eqnarray}
```

Observemos que existe demasiado espacio a cada lado de la columna central, donde se encuentran los signos. Para reducir estas separaciones se puede emplear `\setlength\arraycolsep{2pt}` como en el ejemplo siguiente.

Las **ecuaciones largas** no se dividen automáticamente. Somos nosotros los que determinamos en qué lugares se deben fraccionar y cuánto se debe sangrar. Los dos métodos siguientes son las variantes más utilizadas para esto.

$$\begin{aligned} \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \\ - \frac{x^7}{7!} + \cdots \end{aligned} \qquad (13.8)$$

```
{\setlength\arraycolsep{2pt}
\begin{eqnarray}
\sin x & = & x -\frac{x^{3}}{3!}
     +\frac{x^{5}}{5!}-{}
                          \nonumber\\
 & & {}-\frac{x^{7}}{7!}+{}\cdots
\end{eqnarray}}
```

[13]En este ejemplo la 'dén redondilla se ha introducido a través de la orden `\rd`, que previamente se ha definido con `\newcommand{\rd}{\mathrm{d}}`. De esta forma se evita estar introduciendo la secuencia `\mathrm{d}` repetidamente.

$$\cos x = 1 - \frac{x^2}{2!} + \\ + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots \qquad (13.9)$$

```
\begin{eqnarray}
\lefteqn{ \cos x = 1
     -\frac{x^{2}}{2!} +{} }
                          \nonumber\\
 & & {}+\frac{x^{4}}{4!}
     -\frac{x^{6}}{6!}+{}\cdots
\end{eqnarray}
```

La instrucción `\nonumber` impide que LaTeX coloque un número para la ecuación en la que está colocada la orden.

13.4.6. Tamaño del tipo para ecuaciones

En el modo matemático TeX selecciona el tamaño del tipo según el contexto. Los superíndices, por ejemplo, se ponen en un tipo más pequeño. Si quiere introducir un texto en redondilla en una ecuación y utilizamos la instrucción `\textrm`, el mecanismo de cambio del tamaño del tipo no funcionará, ya que `\textrm` conmuta de modo temporal al modo de texto. Entonces se debe emplear `\mathrm` para que se mantenga activo el mecanismo de cambio de tamaño. Pero hemos de tener cuidado, ya que `\mathrm` sólo funcionará bien con cosas pequeñas. Los espacios no son aún activos y los caracteres con acentos no funcionan[14].

$$2^{\text{o}} \quad 2^{\mathrm{o}} \qquad (13.10)$$

```
\begin{equation}
2^\textrm{o} \quad
2^\mathrm{o}
\end{equation}
```

Sin embargo, a veces resulta necesario indicarle a LaTeX el tamaño del tipo correcto. En modo matemático el tamaño del tipo se fija con las cuatro instrucciones:

`\displaystyle` (123), `\textstyle` (123), `\scriptstyle` (123) y `\scriptscriptstyle` (123).

El cambio de estilo también afecta al modo de presentar los límites.

$$\mathrm{corr}(X,Y) = \frac{\displaystyle\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\displaystyle\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2\sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

```
\begin{displaymath}
\mathrm{corr}(X,Y)=
 \frac{\displaystyle
   \sum_{i=1}^n(x_i-\bar x)
   (y_i-\bar y)}
  {\displaystyle\sqrt{
 \sum_{i=1}^n(x_i-\bar x)^2
\sum_{i=1}^n(y_i-\bar y)^2}}
\end{displaymath}
```

Éste es uno de los ejemplos en los que se necesitan corchetes mayores que los normalizados que proporciona `\left[` y `\right]`.

13.4.7. Descripción de variables

Para algunas de sus ecuaciones podríamos querer añadir una sección donde se describan las variables utilizadas. Para esto nos podría servir de ayuda el siguiente ejemplo:

[14] El paquete AMS-LaTeX hace que la orden `\textrm` funcione bien con el cambio de tamaños.

$$a^2 + b^2 = c^2$$

donde: a, b son los adjuntos del ángulo recto de un triángulo rectángulo.

c es la hipotenusa del triángulo

```
\begin{displaymath}
a^2+b^2=c^2
\end{displaymath}
{\settowidth{\parindent}
   {donde:\ }

\makebox[0pt][r]
 {donde:\ }$a$, $b$ son
los adjuntos del ángulo recto
de un triángulo rectángulo.

$c$ es la hipotenusa
del triángulo}
```

Si necesitamos componer a menudo segmentos de texto como éste, ahora es el momento idóneo para practicar la instrucción `\newenvironment`. Es recomendable emplearla para crear un entorno especializado para describir variables. Revisa la descripción al final del tema anterior.

13.4.8. Teoremas, leyes...

Cuando se escriben documentos matemáticos, probablemente necesitemos un modo para componer "lemas", "definiciones", "axiomas" y estructuras similares. LaTeX facilita esto con la orden

`\newtheorem{`*nombre*`}[`*contador*`]{`*texto*`}[`*sección*`]`

El argumento *nombre* es una palabra clave corta que se utiliza para identificar el "teorema". Con el argumento *texto* se define el nombre del "teorema" que aparecerá en el documento final.

Los argumentos entre corchetes son opcionales. Ambos se emplean para especificar la numeración utilizada para el "teorema". Con el argumento *contador* se puede especificar el *nombre* de un "teorema" declarado previamente. El nuevo "teorema" se numerará con la misma secuencia. El argumento *sección* nos permite indicar la unidad de sección con la que te gustaría numerar nuestro "teorema".

Tras ejecutar la instrucción `\newtheorem` en el preámbulo del documento, dentro del texto se puede usar la instrucción siguiente:

`\begin{`*nombre*`}[`*texto*`]`
Este es un teorema interesante
`\end{`*nombre*`}`

He aquí otro ejemplo de las posibilidades de este entorno:

Ley 1 *No se esconda en la caja testigo*

Jurado 2 (Los doce) *Podría ser Vd. Por tanto, tenga cuidado y vea la ley 1*

Ley 3 *No, No, No*

```
% Definiciones para el documento.
% Preámbulo
\newtheorem{ley}{Ley}
\newtheorem{jurado}[ley]{Jurado}
% En el documento
\begin{ley} \label{law:box}
No se esconda en la caja testigo
\end{ley}
\begin{jurado}[Los doce]
Podría ser Vd. Por tanto, tenga
cuidado y vea la ley
\ref{law:box}\end{jurado}
\begin{ley}No, No, No\end{ley}
```

El teorema "Jurado" emplea el mismo contador que el teorema "Ley". Por ello, toma un número que está en secuencia con las otras "Leyes". El argumento que está entre corchetes se utiliza para especificar un título o algo parecido para el teorema.

Ley de Murphy 13.4.1 *Si algo puede ir mal, irá mal.*

```
\newtheorem{mur}{Ley de Murphy}[section]
\begin{mur} Si algo puede ir mal,
irá mal.
\end{mur}
```

El teorema "Ley de Murphy" obtiene un número que está ligado con el apartado actual. También se podría utilizar otra unidad, como, por ejemplo, un capítulo o un subapartado.

13.4.9. Símbolos en negrita

Es bastante difícil obtener símbolos en negrita en LaTeX. Probablemente esto sea intencionado ya que los compositores de texto aficionados tienden a abusar de ellos. La orden de cambio de tipo `\mathbf` produce letras en negrita, pero éstas son redondillas mientras que los símbolos matemáticos normalmente van en versalita. Existe una orden `\boldmath`, pero *sólo se puede emplear fuera del modo matemático.* También funciona con los símbolos.

$\mu, M \qquad \mathbf{M} \qquad \boldsymbol{\mu, M}$

```
\begin{displaymath}
\mu, M \qquad \mathbf{M} \qquad
\mbox{\boldmath $\mu, M$}
\end{displaymath}
```

Observa que la coma también está en negrita, lo cual puede que a veces no los creas adecuado.

El paquete amsbsy (incluido por amsmath) hace esto mucho más fácil. Incluye una orden `\boldsymbol` y una "negrita del hombre pobre" `\pmb` (*"poor man's bold"*), que opera de forma análoga a las máquinas de escribir, que para poner un texto en negrita se escribe encima del texto ya escrito.

$\mu, M \qquad \boldsymbol{\mu, M} \qquad \pmb{\mu, M}$

```
\begin{displaymath}
\mu, M \qquad
\boldsymbol{\mu}, \boldsymbol{M}
\qquad \pmb{\mu}, \pmb{M}
\end{displaymath}
```

13.5. Especialidades

Llegados a este punto, si yanos sentimos lo sucifientemente seguro de nosotros mismos, entonces ahora ya podemos comenzar a escribir nuestros documentos en LaTeX. El propósito de este tema es añadir algunas 'especias' a nuestros conocimientos de LaTeX. En el Manual de *LaTeX* [?] y *The LaTeX Companion* [?] podremos encontrar una descripción más completa de las especialidades y de las posibles mejoras que podemos realizar con LaTeX.

13.5.1. Tipos y tamaños

LaTeX elige el tipo y el tamaño de los tipos basándose en la estructura lógica del documento (apartados, notas al pie...). En algunos casos podríamos querer cambiar directamente los tipos y los tamaños. Para realizar esto se pueden usar las instrucciones de las tablas 13.8 y 13.9. El tamaño real de cada tipo es cuestión de diseño y depende de la clase de documento y de sus opciones.

Los pequeños y **gordos** romanos dominaron toda la grande *Italia.*

```
{\small Los pequeños y
\textbf{gordos} romanos dominaron}
{\Large toda la grande
\textit{Italia}.}
```

Una característica importante de LaTeX 2_ε es que los atributos de los tipos son independientes. Esto significa que se puede llamar a instrucciones de cambio de tamaño o incluso de tipo y aún así se mantienen los atributos de negrita o inclinado que se establecieron previamente.

En el *modo matemático* se pueden emplear instrucciones de cambio de tipos para salir temporalmente del *modo matemático* e introducir texto normal. Si para componer las ecuaciones prefiriésemos utilizar otro tipo existe un conjunto especial de instrucciones para ello. Consulta para esto la tabla 13.10.

Tabla 13.8: Tipos

`\textrm{...}`	redonda	`\textsf{...}`	sin línea de pie
`\texttt{...}`	`de máquina de escribir`		
`\textmd{...}`	media	`\textbf{...}`	**negrita**
`\textup{...}`	vertical	`\textit{...}`	*itálica*
`\textsl{...}`	*inclinada*	`\textsc{...}`	VERSALITA
`\emph{...}`	*resaltada*	`\textnormal{...}`	tipo del documento

Tabla 13.9: Tamaños de los tipos

`\tiny`	letra diminuta	`\Large`	letra mayor
`\scriptsize`	letra muy pequeña	`\LARGE`	muy grande
`\footnotesize`	letra bastante pequeña	`\huge`	enorme
`\small`	letra pequeña	`\Huge`	la mayor
`\normalsize`	letra normal		
`\large`	letra grande		

Tabla 13.10: Tipos matemáticos

Orden	*Ejemplo*	*Resultado*
`\mathcal{...}`	`$\mathcal{B}=c$`	$\mathcal{B} = c$
`\mathrm{...}`	`$\mathrm{K}_2$`	K_2
`\mathbf{...}`	`$\sum x=\mathbf{v}$`	$\sum x = \mathbf{v}$
`\mathsf{...}`	`$\mathsf{G\times R}$`	$\mathsf{G} \times \mathsf{R}$
`\mathtt{...}`	`$\mathtt{L}(b,c)$`	$\mathtt{L}(b, c)$
`\mathnormal{...}`	`$\mathnormal{R_1}=R_1$`	$R_1 = R_1$
`\mathit{...}`	`$eficaz\neq\mathit{eficaz}$`	$eficaz \neq \mathit{eficaz}$

Conjuntamente con las instrucciones de los tamaños de los tipos, las llaves juegan un papel significativo. Se utilizan para construir agrupaciones o *grupos*. Los grupos limitan el ámbito de la mayoría de las instrucciones de LaTeX.

A él le gustan las {\LARGE letras grandes y las letras {\small pequeñas}}.

```
A él le gustan las {\LARGE
letras grandes y las letras
{\small pequeñas}}.
```

Las instrucciones de tamaño del tipo también alteran el espaciado entre renglones, pero sólo si el párrafo termina dentro del ámbito de la orden de tamaño del tipo. Por ello, la llave de cierre } no debería aparecer antes de lo indicado. Fijémonos la posición de la instrucción `\par` en los dos ejemplos siguientes.

{\Large <No leas esto! No es cierto. <Créeme!\par}

```
{\Large <No leas esto! No es
cierto. <Créeme!\par}
```

{\Large Esto no es cierto. Pero recuerda que digo mentiras.}\par

```
{\Large Esto no es cierto.
Pero recuerda que digo
mentiras.}\par
```

Para concluir este viaje al mundo de los tipos y los tamaños de tipos, debemos tener presente un pequeño consejo:

¡Recuerda! que *Cuántos* MÁS tipos utilices en un documento, *más* LEGIBLE y *agradable resultará*.[15]

13.5.2. Separaciones

13.5.2.1. Separaciones entre renglones

Si queremos emplear mayores separaciones entre renglones, podemos cambiar su valor poniendo la orden

`\linespread{`*factor*`}`

en el preámbulo de su documento. Utiliza `\linespread{1.3}` para textos a espacio y medio y `\linespread{1.6}` para textos a doble espacio. Normalmente los renglones no se separan tanto, por lo que, a no ser que se indique otra cosa, el factor de separación entre renglones es 1.

13.5.2.2. Diseño de los párrafos

En LaTeX existen dos parámetros que influyen en el formato de los párrafos. Si se pone una definición como

```
\setlength{\parindent}{0pt}
\setlength{\parskip}{1ex plus 0.5ex minus 0.2ex}
```

en el preámbulo del fichero de entrada[16] se puede cambiar el aspecto de los párrafos. Estas dos líneas pueden aumentar el espacio entre dos párrafos y dejarlos sin sangrías. En la Europa continental, a menudo se separan los párrafos con algún espacio y no se le pone sangría. Pero hay que tener cuidado, ya que esto también tiene efecto en el índice general, haciendo que nuestras líneas queden más separadas.

Si quisioérmos sangrar un párrafo que no tiene sangría, entonces utilizaremos

`\indent`

al comienzo del párrafo[17]. Esto sólo funcionará cuando `\parindent` no esté puesto a cero.

[15] ¡Ojo!, que se trata de una pequeña sátira. ¡Espero que te des cuenta!
[16] El preámbulo es la zona del fichero de entrada entre las instrucciones `\documentclass` y `\begin{document}`.
[17] Para sangrar el primer párrafo después de cada cabecera de apartado, emplea el paquete indentfirst del conjunto 'tools'.

Para crear un párrafo sin sangría se emplea

`\noindent`

como primera orden del párrafo. Esto podría resultar útil cuando comencemos un documento con texto y sin ninguna instrucción de seccionado.

13.5.2.3. Separaciones horizontales

LaTeX determina automáticamente las separaciones entre palabras y oraciones. Para producir otras separaciones horizontales se utiliza:

`\hspace{`*longitud*`}`

Cuando se debe producir una separación como ésta, incluso si cae al final o al comienzo de un renglón, emplea `\hspace*` en vez de `\hspace`. La indicación de la distancia consta, en el caso más simple, de un número más una unidad. En la tabla 13.11 encontrarás las unidades más importantes.

Este es un espacio de 1.5 cm.

```
Este\hspace{1.5cm}es un espacio
de 1.5 cm.
```

La instrucción

`\stretch{`*n*`}`

produce una separación especial elástica. Se alarga hasta que el espacio que resta en un renglón se llena. Si dos instrucciones `\hspace{\stretch{`*n*`}}` aparecen en el mismo renglón, los espaciados crecen según sus 'factores de alargamiento'.

x x x

```
x\hspace{\stretch{1}}
x\hspace{\stretch{3}}x
```

13.5.2.4. Separaciones verticales especiales

LaTeX determina de modo automático las separaciones entre dos párrafos, apartados, subapartados... En casos especiales se pueden forzar separaciones adicionales *entre dos párrafos* con la orden

`\vspace{`*longitud*`}`

Esta orden se debería indicar siempre entre dos renglones vacíos. Cuando esta separación se debe introducir aunque vaya al principio o al final de una página, entonces en vez de `\vspace` se debe utilizar `\vspace*`.

Se puede utilizar la orden `\stretch` conjuntamente con `\pagebreak` para llevar texto al borde inferior de una página o para centrarlo verticalmente.

```
Algo de texto \ldots

\vspace{\stretch{1}}
Esto va en el último renglón de la página.\pagebreak
```

Las separaciones adicionales entre dos renglones *del mismo párrafo* o dentro de una tabla se consiguen con la orden

`\\[`*longitud*`]`

Tabla 13.11: Unidades de TeX

mm	milímetro ≈ 1/25 pulgada
cm	centímetro = 10 mm
in	pulgada ≈ 25 mm
pt	punto ≈ 1/72 pulgada ≈ $\frac{1}{3}$ mm
em	aprox. el ancho de una `m` en el tipo actual
ex	aprox. la altura de una `x` en el tipo actual

13.5.3. Inclusión de gráficos EPS

Con los entornos `figure` y `table` LaTeX proporciona las facilidades básicas para trabajar con objetos flotantes, entre los que se incluyen las imágenes y los gráficos.

Existen varias alternativas para generar gráficos con el LaTeX básico o un paquete de extensiones de LaTeX mediante órdenes. Por desgracia, la mayoría de los usuarios los encuentran difíciles de entender. Por esto, no las vamos a explicar aquí. Si realmente se desea buscar más información sobre este particular siempre se puede echar mano de textos más específicos y extensos, como *The LaTeX Companion* [?] y el Manual de *LaTeX* [?].

Un modo más sencillo de poner gráficos en un documento es produciéndolos con un paquete de *software* especializado[18] e incluir los gráficos dentro del documento. En este punto, también hay paquetes de LaTeX que ofrecen muchas alternativas. En esta descripción sólo mostraremos el uso de gráficos en PostScript Encapsulado (EPS), ya que es un método muy sencillo y utilizado ampliamente. Para utilizar dibujos en formato EPS, debemos disponer una impresora PostScript[19] para imprimir.

Un buen conjunto de órdenes para la inclusión de gráficos se proporciona con el paquete graphicx de D. P. Carlisle. Forma parte de todo un conjunto de paquetes que se llama el conjunto "graphics"[20].

Suponiendo que dispongamos de una impresora PostScript para imprimir y utilicemos el paquete graphicx, se puede seguir la siguiente lista de pasos para incluir un dibujo dentro de nuestros documentos:

1. Exportar el dibujo desde tu programa de gráficos en formato EPS.
2. Cargar el paquete graphicx en el preámbulo del fichero de entrada con

   `\usepackage[`*driver*`]{graphicx}`

   driver es el nombre de su conversor "de *dvi* a PostScript"[21]. El paquete necesita esta información porque la inclusión de los gráficos la realiza el *driver* de la impresora. Una vez que se conozca el *driver*, el paquete graphicx inserta las órdenes correctas en el fichero `.dvi` para incluir el gráfico que se desea con el *driver* de impresora.
3. Utilizar la orden

   `\includegraphics[`*clave*`=`*valor*`, ...]{`*fichero*`}`

   para incluir *fichero* en tus documentos. El parámetro opcional acepta una lista de *claves* separadas por comas y sus *valores* asociados. Las *claves* se pueden emplear para modificar el ancho, la altura y el giro del gráfico incluido. En la tabla 13.12 puedes encontrar las claves más importantes.

El siguiente ejemplo podrá ayudar a aclarar algunas de estas ideas:

```
\begin{figure}
\begin{center}
\includegraphics[angle=90, width=10cm]{test.eps}
\end{center}
\end{figure}
```

[18] Tales como XFig, CorelDraw!, Freehand, Gnuplot, Tgif, Paint Shop Pro, Gimp...

[19] Otra posibilidad para imprimir PostScript es con el programa de GNU GhostScript, que puede encontrar, p. ej., en `CTAN:/tex-archive/support/ghostscript`.

[20] `CTAN:/tex-archive/macros/latex/packages/graphics`.

[21] El programa más utilizado para esto se llama `dvips`.

Tabla 13.12: Nombres de las claves para el paquete graphicx

`width`	escalado gráfico al ancho indicado
`height`	escalado gráfico a la altura indicada
`angle`	giro del gráfico en el sentido de las agujas del reloj

Este código inserta el gráfico que se encuentra en el fichero `test.eps`. El gráfico se gira *primero* 90° y *después* se escala hasta lograr los 10 cm de ancho. La relación de aspecto es de 1.0 porque no se ha indicado ninguna altura especial.

Para buscar más información sobre este paquete y sus posibilidades siempre podemos acudir a su documentación en [**?**].

13.6. Herramientas de LaTeX

Copiamos esta fórmula en algún fichero, por ejemplo `ejemplo.tex` y veamos los pasos necesarios para *compilarlo*. Para compilar el fuente lo único que hay que hacer es ejecutar LaTeX seguido del nombre del fichero fuente LaTeX. En nuestro ejemplo, nos colocaremos en el directorio donde tenemos el `ejemplo.tex` y teclearemos:

```
$ latex ejemplo.tex
```

Esto generará varios ficheros, pero el que realmente interesa en estos momentos es `ejemplo.dvi`. Este fichero es el resultado de la compilación y lo podemos ver usando por ejemplo `xdvi`. Para verlo teclearemos en el mismo directorio en donde está el fichero

```
$ xdvi ejemplo.dvi
```

Dado que el formato `DVI` (DeVice Independent) no está muy extendido se han generado conversores de formato, de tal manera que podemos pasar desde dvi a formato `PostScript` y a formato `PDF`, ambos mucho más extendidos que el dvi. Para pasar un fichero generado por LaTeX en formato `DVI` a un fichero `PostScript` usaremos el comando `dvips` de la siguiente manera:

```
$ dvips ejemplo.dvi -o ejemplo.ps
```

Ahora podemos ver el resultado con cualquier visor de ficheros PostScript, por ejemplo `gv`:

```
$ gv ejemplo.ps
```

Una vez comprobado que todo va bien intentemos pasar este último fichero en formato `PostScript` a formato `PDF`. Para ello usaremos el comando `ps2pdf` como se explica a continuación:

```
$ ps2pdf ejemplo.ps ejemplo.pdf
```

Y una vez hecho esto ya podemos usar el visor de PDF para leer el `ejemplo.pdf`.

Existe también la posibilidad de convertir el código LaTeX de un documento al estándar HTML, lo que resulta muy útil a la hora de publicar en internet. El programa apropiado para esto es el `GNU LaTeX2HTML`, y su uso básico es sumamente sencillo:

```
$ latex2html ejemplo.tex
```

Al terminar latex2html encontramos un nuevo directorio llamado `ejemplo` que contiene la conversión a HTML de nuestro ejemplo de LaTeX, incluidos los gráficos correspondientes a las fórmulas matemáticas. El resultado obtenido no está nada mal. Además, esta conversión puede personalizarse en gran medida mediante hojas de estilo.

13.7. Tablas de símbolos matemáticos

En las tablas siguientes se indican todos los símbolos que normalmente se pueden utilizar en el *modo matemático*.

Para usar los símbolos de las tablas 13.24–13.28[22], se debe cargar el paquete amssymb en el preámbulo del documento y además deberán encontrarse en el sistema los tipos matemáticos de la *American Mathematical Society* (AMS). Si no están instalados el paquete y los tipos de la AMS, entonces eche un vistazo a

`CTAN:/tex-archive/macros/latex/packages/amslatex`

Tabla 13.13: Acentos en modo matemático

$\hat{a}$	`\hat{a}`	$\check{a}$	`\check{a}`	$\tilde{a}$	`\tilde{a}`	$\acute{a}$	`\acute{a}`
$\grave{a}$	`\grave{a}`	$\dot{a}$	`\dot{a}`	$\ddot{a}$	`\ddot{a}`	$\breve{a}$	`\breve{a}`
$\bar{a}$	`\bar{a}`	$\vec{a}$	`\vec{a}`	$\widehat{A}$	`\widehat{A}`	$\widetilde{A}$	`\widetilde{A}`

Tabla 13.14: Letras griegas mayúsculas

Γ	`\Gamma`	Λ	`\Lambda`	Σ	`\Sigma`	Ψ	`\Psi`
Δ	`\Delta`	Ξ	`\Xi`	Υ	`\Upsilon`	Ω	`\Omega`
Θ	`\Theta`	Π	`\Pi`	Φ	`\Phi`		

[22] Estas tablas provienen de `symbols.tex` y luego se hicieron muchas modificaciones según las sugerencias de Josef Tkadlec

Tabla 13.15: Letras griegas minúsculas

α	\alpha	θ	\theta	o	o	υ	\upsilon
β	\beta	ϑ	\vartheta	π	\pi	ϕ	\phi
γ	\gamma	ι	\iota	ϖ	\varpi	φ	\varphi
δ	\delta	κ	\kappa	ρ	\rho	χ	\chi
ϵ	\epsilon	λ	\lambda	ϱ	\varrho	ψ	\psi
ε	\varepsilon	μ	\mu	σ	\sigma	ω	\omega
ζ	\zeta	ν	\nu	ς	\varsigma		
η	\eta	ξ	\xi	τ	\tau		

Tabla 13.16: Relaciones

Puede realizar las negaciones correspondientes a estos símbolos añadiéndoles una orden \not como prefijo a las instrucciones siguientes.

$<$	<	$>$	>	$=$	=
$\leq$	\leq o \le	$\geq$	\geq o \ge	$\equiv$	\equiv
$\ll$	\ll	$\gg$	\gg	$\doteq$	\doteq
$\prec$	\prec	$\succ$	\succ	$\sim$	\sim
$\preceq$	\preceq	$\succeq$	\succeq	$\simeq$	\simeq
$\subset$	\subset	$\supset$	\supset	$\approx$	\approx
$\subseteq$	\subseteq	$\supseteq$	\supseteq	$\cong$	\cong
$\sqsubset$ [a]	\sqsubset [a]	$\sqsupset$	\sqsupset [a]	$\bowtie$	\Join [a]
$\sqsubseteq$	\sqsubseteq	$\sqsupseteq$	\sqsupseteq	$\bowtie$	\bowtie
$\in$	\in	$\ni$	\ni , \owns	$\propto$	\propto
$\vdash$	\vdash	$\dashv$	\dashv	$\models$	\models
$\mid$	\mid	$\parallel$	\parallel	$\perp$	\perp
$\smile$	\smile	$\frown$	\frown	$\asymp$	\asymp
$:$	:	$\notin$	\notin	$\neq$	\neq o \ne

[a] Para obtener este símbolo emplee el paquete latexsym

Tabla 13.17: Operadores binarios

$+$	+	$-$	-		
$\pm$	\pm	$\mp$	\mp	$\triangleleft$	\triangleleft
$\cdot$	\cdot	$\div$	\div	$\triangleright$	\triangleright
$\times$	\times	$\setminus$	\setminus	$\star$	\star
$\cup$	\cup	$\cap$	\cap	$*$	\ast
$\sqcup$	\sqcup	$\sqcap$	\sqcap	$\circ$	\circ
$\vee$	\vee , \lor	$\wedge$	\wedge , \land	$\bullet$	\bullet
$\oplus$	\oplus	$\ominus$	\ominus	$\diamond$	\diamond
$\odot$	\odot	$\oslash$	\oslash	$\uplus$	\uplus
$\otimes$	\otimes	$\bigcirc$	\bigcirc	$\amalg$	\amalg
$\bigtriangleup$	\bigtriangleup	$\bigtriangledown$	\bigtriangledown	$\dagger$	\dagger
$\lhd$ [a]	\lhd [a]	$\rhd$	\rhd [a]	$\ddagger$	\ddagger
$\unlhd$ [a]	\unlhd [a]	$\unrhd$	\unrhd [a]	$\wr$	\wr

[a] Para obtener este símbolo emplee el paquete latexsym

Tabla 13.18: Operadores "grandes"

$\sum$	\sum	$\bigcup$	\bigcup	$\bigvee$	\bigvee	$\bigoplus$	\bigoplus
$\prod$	\prod	$\bigcap$	\bigcap	$\bigwedge$	\bigwedge	$\bigotimes$	\bigotimes
$\coprod$	\coprod	$\bigsqcup$	\bigsqcup			$\bigodot$	\bigodot
$\int$	\int	$\oint$	\oint			$\biguplus$	\biguplus

Tabla 13.19: Flechas

$\leftarrow$	\leftarrow o \gets	$\longleftarrow$	\longleftarrow	$\uparrow$	\uparrow
$\rightarrow$	\rightarrow o \to	$\longrightarrow$	\longrightarrow	$\downarrow$	\downarrow
$\leftrightarrow$	\leftrightarrow	$\longleftrightarrow$	\longleftrightarrow	$\updownarrow$	\updownarrow
$\Leftarrow$	\Leftarrow	$\Longleftarrow$	\Longleftarrow	$\Uparrow$	\Uparrow
$\Rightarrow$	\Rightarrow	$\Longrightarrow$	\Longrightarrow	$\Downarrow$	\Downarrow
$\Leftrightarrow$	\Leftrightarrow	$\Longleftrightarrow$	\Longleftrightarrow	$\Updownarrow$	\Updownarrow
$\mapsto$	\mapsto	$\longmapsto$	\longmapsto	$\nearrow$	\nearrow
$\hookleftarrow$	\hookleftarrow	$\hookrightarrow$	\hookrightarrow	$\searrow$	\searrow
$\leftharpoonup$	\leftharpoonup	$\rightharpoonup$	\rightharpoonup	$\swarrow$	\swarrow
$\leftharpoondown$	\leftharpoondown	$\rightharpoondown$	\rightharpoondown	$\nwarrow$	\nwarrow
$\rightleftharpoons$	\rightleftharpoons	$\iff$	\iff (espacios mayores)	$\leadsto$ [a]	\leadsto [a]

[a] Para obtener este símbolo emplee el paquete latexsym

Tabla 13.20: Delimitadores

(	(	)	)	$\uparrow$	\uparrow	$\Uparrow$	\Uparrow
[	[o \lbrack	]	] o \rbrack	$\downarrow$	\downarrow	$\Downarrow$	\Downarrow
{	\{ o \lbrace	}	\} o \rbrace	$\updownarrow$	\updownarrow	$\Updownarrow$	\Updownarrow
$\langle$	\langle	$\rangle$	\rangle	$\vert$	| o \vert	$\Vert$	\\| o \Vert
$\lfloor$	\lfloor	$\rfloor$	\rfloor	$\lceil$	\lceil	$\rceil$	\rceil
/	/	\	\backslash		. (vacío dual)		

Tabla 13.21: Delimitadores grandes

$\lgroup$	\lgroup	$\rgroup$	\rgroup	$\lmoustache$	\lmoustache	$\rmoustache$	\rmoustache
$\arrowvert$	\arrowvert	$\Arrowvert$	\Arrowvert	$\bracevert$	\bracevert		

Tabla 13.22: Símbolos diversos

...	\dots	$\cdots$	\cdots	$\vdots$	\vdots	$\ddots$	\ddots
$\hbar$	\hbar	$\imath$	\imath	$\jmath$	\jmath	ℓ	\ell
$\Re$	\Re	$\Im$	\Im	$\aleph$	\aleph	$\wp$	\wp
$\forall$	\forall	$\exists$	\exists	$\mho$ [a]	\mho [a]	∂	\partial
′	'	$\prime$	\prime	$\emptyset$	\emptyset	∞	\infty
∇	\nabla	$\triangle$	\triangle	$\Box$ [a]	\Box [a]	$\Diamond$ [a]	\Diamond [a]
$\bot$	\bot	$\top$	\top	$\angle$	\angle	$\surd$	\surd
$\diamondsuit$	\diamondsuit	$\heartsuit$	\heartsuit	$\clubsuit$	\clubsuit	$\spadesuit$	\spadesuit
$\neg$	\neg o \lnot	$\flat$	\flat	$\natural$	\natural	$\sharp$	\sharp

[a] Para obtener este símbolo emplee el paquete latexsym

Tabla 13.23: Símbolos no matemáticos

Los siguientes símbolos también se pueden utilizar en modo texto.

†	\dag	§	\S	©	\copyright
‡	\ddag	¶	\P	£	\pounds

Tabla 13.24: Delimitadores de la AMS

⌜	\ulcorner	⌝	\urcorner	⌞	\llcorner	⌟	\lrcorner

Tabla 13.25: Símbolos griegos y hebreos de la AMS

F	\digamma	$\varkappa$	\varkappa	ב	\beth	ד	\daleth	ג	\gimel

Tabla 13.26: Relaciones binarias de la AMS

⩿	\lessdot	⩾	\gtrdot	≒	\doteqdot o \Doteq
⩽	\leqslant	⩾	\geqslant	≓	\risingdotseq
⪕	\eqslantless	⪖	\eqslantgtr	≒	\fallingdotseq
≦	\leqq	≧	\geqq	≡	\eqcirc
⋘	\lll o \llless	⋙	\ggg o \gggtr	≗	\circeq
≶	\lesssim	≷	\gtrsim	≜	\triangleq
⪅	\lessapprox	⪆	\gtrapprox	≏	\bumpeq
≶	\lessgtr	≷	\gtrless	≎	\Bumpeq
⋚	\lesseqgtr	⋛	\gtreqless	∼	\thicksim
⪋	\lesseqqgtr	⪌	\gtreqqless	≈	\thickapprox
≺	\preccurlyeq	≻	\succcurlyeq	≊	\approxeq
≼	\curlyeqprec	≽	\curlyeqsucc	∽	\backsim
≾	\precsim	≿	\succsim	⋍	\backsimeq
⪷	\precapprox	⪸	\succapprox	⊨	\vDash
⫅	\subseteqq	⫆	\supseteqq	⊩	\Vdash
⋐	\Subset	⋑	\Supset	⊪	\Vvdash
⊏	\sqsubset	⊐	\sqsupset	϶	\backepsilon
∴	\therefore	∵	\because	∝	\varpropto
∣	\shortmid	∥	\shortparallel	≬	\between
⌣	\smallsmile	⌢	\smallfrown	⋔	\pitchfork
◁	\vartriangleleft	▷	\vartriangleright	◀	\blacktriangleleft
⊴	\trianglelefteq	⊵	\trianglerighteq	▶	\blacktriangleright

Tabla 13.27: Flechas de la AMS

⇠	\dashleftarrow	⇢	\dashrightarrow	⊸	\multimap
⇇	\leftleftarrows	⇉	\rightrightarrows	⇈	\upuparrows
⇆	\leftrightarrows	⇄	\rightleftarrows	⇊	\downdownarrows
⇚	\Lleftarrow	⇛	\Rrightarrow	↿	\upharpoonleft
↞	\twoheadleftarrow	↠	\twoheadrightarrow	↾	\upharpoonright
↢	\leftarrowtail	↣	\rightarrowtail	⇂	\downharpoonleft
⇋	\leftrightharpoons	⇌	\rightleftharpoons	⇃	\downharpoonright
↰	\Lsh	↱	\Rsh	⇝	\rightsquigarrow
↫	\looparrowleft	↬	\looparrowright	↭	\leftrightsquigarrow
↶	\curvearrowleft	↷	\curvearrowright		
↺	\circlearrowleft	↻	\circlearrowright		

Tabla 13.28: Relaciones binarias y flechas negadas de la AMS

≮	\nless	≯	\ngtr	⊊	\varsubsetneqq
≨	\lneq	≩	\gneq	⊋	\varsupsetneqq
≰	\nleq	≱	\ngeq	⊈	\nsubseteqq
≰	\nleqslant	≱	\ngeqslant	⊉	\nsupseteqq
≨	\lneqq	≩	\gneqq	∤	\nmid
≨	\lvertneqq	≩	\gvertneqq	∦	\nparallel
≰	\nleqq	≱	\ngeqq	∤	\nshortmid
≨	\lnsim	≩	\gnsim	∦	\nshortparallel
≨	\lnapprox	≩	\gnapprox	≁	\nsim
⊀	\nprec	⊁	\nsucc	≇	\ncong
⋠	\npreceq	⋡	\nsucceq	⊬	\nvdash
⋨	\precneqq	⋩	\succneqq	⊭	\nvDash
⋨	\precnsim	⋩	\succnsim	⊮	\nVdash
⋨	\precnapprox	⋩	\succnapprox	⊯	\nVDash
⊊	\subsetneq	⊋	\supsetneq	⋪	\ntriangleleft
⊊	\varsubsetneq	⊋	\varsupsetneq	⋫	\ntriangleright
⊈	\nsubseteq	⊉	\nsupseteq	⋬	\ntrianglelefteq
⊈	\subsetneqq	⊉	\supsetneqq	⋭	\ntrianglerighteq
↚	\nleftarrow	↛	\nrightarrow	↮	\nleftrightarrow
⇍	\nLeftarrow	⇏	\nRightarrow	⇎	\nLeftrightarrow

Tabla 13.29: Operadores binarios de la AMS

∔	\dotplus	∙	\centerdot	⊺	\intercal
⋉	\ltimes	⋊	\rtimes	⋇	\divideontimes
⋓	\Cup o \doublecup	⋒	\Cap o \doublecap	∖	\smallsetminus
⊻	\veebar	⊼	\barwedge	⩞	\doublebarwedge
⊞	\boxplus	⊟	\boxminus	⊝	\circleddash
⊠	\boxtimes	⊡	\boxdot	⊚	\circledcirc
⋋	\leftthreetimes	⋌	\rightthreetimes	⊛	\circledast
⋎	\curlyvee	⋏	\curlywedge		

Tabla 13.30: Símbolos diversos de la AMS

$\hbar$	`\hbar`	$\hbar$	`\hslash`	$\Bbbk$	`\Bbbk`
$\square$	`\square`	$\blacksquare$	`\blacksquare`	Ⓢ	`\circledS`
$\vartriangle$	`\vartriangle`	$\blacktriangle$	`\blacktriangle`	$\complement$	`\complement`
$\triangledown$	`\triangledown`	$\blacktriangledown$	`\blacktriangledown`	$\Game$	`\Game`
$\lozenge$	`\lozenge`	$\blacklozenge$	`\blacklozenge`	$\bigstar$	`\bigstar`
$\angle$	`\angle`	$\measuredangle$	`\measuredangle`	$\sphericalangle$	`\sphericalangle`
$\diagup$	`\diagup`	$\diagdown$	`\diagdown`	$\backprime$	`\backprime`
$\nexists$	`\nexists`	$\Finv$	`\Finv`	$\varnothing$	`\varnothing`
$\eth$	`\eth`	$\mho$	`\mho`		

Tabla 13.31: Alfabetos matemáticos

Ejemplo	Instrucción	Paquete necesario
ABCdef	`\mathrm{ABCdef}`	
$ABCdef$	`\mathit{ABCdef}`	
$ABCdef$	`\mathnormal{ABCdef}`	
$\mathcal{ABC}$	`\mathcal{ABC}`	
	`\mathcal{ABC}`	euscript con opción mathcal
	`\mathscr{ABC}`	euscript con opción mathscr
$\mathfrak{ABCdef}$	`\mathfrak{ABCdef}`	eufrak
$\mathbb{ABC}$	`\mathbb{ABC}`	amsfonts o amssymb

MÓDULO IV

Herramientas matemáticas

TEMA 14

GNU Octave

Octave se puede definir como un lenguaje de alto nivel inspirado en un software comercial llamado MATLAB® (MATrix LABoratory). MATLAB® estuvo pensado inicialmente para álgebra numérica lineal (matrices, vectores y sus operaciones), y con el tiempo se le ha sacado partido a esta forma de trabajo. De la misma forma, Octave empezó siendo un software para que los alumnos de Ingeniería Química de las Universidades de Wisconsin-Madison y Texas calcularan reacciones químicas.

A partir de ese momento, las contribuciones de los usuarios han hecho evolucionar este software y han añadido librerías y funcionalidades. Ahora, las aplicaciones de Octave ya no se limitan a simple trabajo con matrices y vectores, como una mera calculadora, sino que ahora aparte de aplicaciones puramente matemáticas o numéricas, es válido para otros campos de ciencias e ingenierías. Entre ellos, el procesamiento de señales (sonido), de imágenes (filtrados, análisis, etc), estadística, geometría, redes neuronales, sistemas de control realimentados y hasta dibujo vectorial. Intentaremos poner ejemplos de cada una de estas aplicaciones en la medida de lo posible para mostrar la versatilidad de Octave.

Estas librerías se pueden programar de forma interpretada, usando el propio lenguaje de octave, o de forma binaria, usando cualquiera de los lenguajes que soporte gcc como C/C++, pascal o fortran (recordemos que todo el código objeto era intercambiable). Además, también se puede hacer a la inversa, es decir, traducir programas de octave a c++ usando una librería llamada liboctave. Con esto se elimina la etapa de interpretación al ejecutarlo con lo que se gana velocidad cuando ésta sea determinante. Parece que hay bastantes cosas por ver, así que vamos a empezar.

14.1. Entorno

Octave tiene una filosofía de uso semejante a la de muchas otras aplicaciones de este libro: una interfaz en forma de shell, con una linea de comandos potente con muchos atajos y facilidades, para problemas sencillos, y la posibilidad de poder agrupar muchos comandos en ficheros de scripts, organizados en funciones, para enfrentarse a problemas complejos o para realizar automatizaciones.

Para comenzar a ver el manejo básico vamos a ejecutar Octave de manera interactiva. Con este método de trabajo, si cometemos un error al entrar una línea, podremos corregirlo sobre la marcha. Para ejecutarlo, abre un terminal y en la línea de comandos teclea `octave`. Tras un mensaje de bienvenida, Octave te muestra un prompt que indica que está preparado y a la espera de comandos. En algunas distribuciones, Octave puede tener su icono en uno de los menúes del sistema, en la zona de aplicaciones matemáticas. Teclear `octave` en la consola es más rápido y funciona el 100 % de las veces. Esto es lo que se nos muestra:

```
$ octave
GNU Octave, version 2.1.34 (i386-pc-linux-gnu).
Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001 John W. Eaton.
This is free software with ABSOLUTELY NO WARRANTY.
For details, type 'warranty'.

octave:1>
```

Cuando quieras salir de Octave teclea `exit`, `quit` o `C-D` y volverás al shell de partida.

La ayuda completa de octave la puedes obtener desde el prompt tecleando `help -i`. También puedes visualizar la misma ayuda desde el shell tecleando `info octave`. Luego, la documentación para cada función y variables se obtienen tecleando `help nombredelafuncion`. Por ejemplo:

```
octave:9> help coth
coth is the user-defined function from the file
/usr/share/octave/2.1.34/m/elfun/coth.m

 - Mapping Function:  coth (X)
     Compute the hyperbolic cotangent of each element of X.
```

La mayoría de los comandos de Octave disponen de esta ayuda. Vemos que se nos dice una descripción de los parámetros y lo que realiza la función, lo cual es suficiente para que podamos utilizarla.

Cuando se invoca sin argumentos se obtiene un listado de todas las operaciones, funciones y variables incorporadas definidas en el sistema. Para conseguir esta información, Octave rastrea por los directorios donde están instaladas las funciones; de ahí su peculiar forma de organizar esta salida, que nos muestra las funciones clasificadas por temas, lo que puede ayudarnos a mirar y probar funciones. Además, aquí podemos encontrarnos funciones que no están pasadas a la documentación.

Octave usa la librería *GNU readline* para la edición en línea de comandos, al igual que *bash* y otros programas *GNU*. Contiene un historial que se puede leer con las flechas arriba y abajo, muchas combinaciones de teclas para hacer muchas cosas. Para más información y explicación sobre estas características teclea desde un shell `info rluserman`. No entraremos más en este tema.

Cada vez que te equivoques en la sintaxis, octave te indicará la posición donde cree que está el fallo con un angulillo `^`. A veces no se puede fiar uno completamente, y sólo te ayuda a saber más o menos donde se localiza. Veámoslo aquí:

```
octave:13>  functon y = f (x) y = x^2; endfunction
parse error:

>>>  functon y = f (x) y = x^2; endfunction
                ^
```

Otro tipo de errores pueden ocurrir dentro de funciones. En este caso, son errores en tiempo de ejecución, porque ocurren por un fallo en la ejecución del programa. En este caso, lo que aparece es la línea y posición dentro de la función y en la función que lo llamó y en las siguientes. Por ejemplo, en este hipotético caso:

```
octave:13> f ()
error: 'x' undefined near line 1 column 24
error: evaluating expression near line 1, column 24
error: evaluating assignment expression near line 1, column 22
error: called from 'f'
```

En este caso, la función f se compone de unas funciones que se llaman a otras. El error está en la línea 1 con una x mal definida. La función que contiene este error, según octave, formaba parte de una expresión en la linea 1, que a su vez formaba parte de una asignación tambien en la línea 1. Obsérvese que la función es la que definimos en el anterior ejemplo y el error es que hace falta pasarle un parámetro a la función.

Los comentarios dentro del código de octave se preceden con el carácter `#` o `%` y abarcan desde ese carácter hasta el final de la línea. Si ponemos en octave un código como éste:

```
function xdot = f (x, t)

# usage: f (x, t)
#
# This function defines the right hand
# side functions for a set of nonlinear
```

```
# differential equations.

  r = 0.25;
  ...
endfunction
```

Octave interpreta los comentarios después de la función como el texto de ayuda. De esta forma, cuando hagamos `help xdot` nos mostrará como ayuda el texto que hemos definido en el ejemplo.

Cuando una línea se hace demasiado larga se puede añadir al final de la linea una barra invertida \ o unos puntos suspensivos ... y continuar en la siguiente línea.

```
x = long_variable_name ...
    + longer_variable_name \
    - 42
```

Otra cosa interesante es que por defecto se muestra el resultado de la operación al realizarla salvo cuando se añade un ; al final de la operación. También se pueden hacer varias operaciones en una misma linea separándolas con ;.

```
octave:1> a=sqrt(3)
a = 1.7321
octave:2> b=sqrt(5);
octave:3> b
b = 2.2361
octave:4> sqrt(7)
ans = 2.6458
```

Como última nota, hay que añadir que cuando no capturamos el valor de retorno aparece la palabra `ans`, que representa el último resultado, y que se puede usar como variable en la siguiente línea. Continuando el ejemplo anterior:

```
octave:5> ans*ans
ans = 7.0000
octave:6> ans*ans
ans = 49.000
```

14.2. Tipos de datos

En Octave hay tres tipos de datos: numéricos (escalares, vectores y matrices), cadenas (strings) y estructuras. Como es de suponer, los numéricos son los más usados, mientras que cadenas sólo se usan para presentar mensajes. Las estructuras son algo que nos pueden dar una buena base para organizar tareas más complicadas. A continuación se muestra la forma de definir cada uno de ellos.

```
octave:13> a=[1 2 3]
a =
  1  2  3

octave:14> b=[1,2,3;4,5,6]
b =
  1  2  3
  4  5  6

octave:15> c="hola mundo"
c = hola mundo

octave:16> d.vector=[1;2];
octave:17> d.matriz=[1 2; 3 4];
```

```
octave:18> d.texto="titulo";
octave:19> d
d =
{
  texto = titulo
  vector =
    1
    2

  matriz =
    1  2
    3  4

}
octave:20> d.vector
d.vector =
    1
    2
```

Se observará que las definiciones de matrices o vectores, las filas van separadas por comas `(,)` o espacios `( )` mientras que las columnas se separan por punto y coma `(;)`. También observamos que una estructura no es más que un *paquete* donde se pueden almacenar varias variables relacionadas juntas. Se usan principalmente para reducir el número de argumentos de las funciones agrupando todos los argumentos relacionados en estructuras y pasando éstas.

En relación con matrices y vectores, las siguientes funciones nos informan sobre las dimensiones de estos elementos. Por ejemplo:

```
octave:25> length(a)
ans = 3
octave:26> columns(b)
ans = 3
octave:27> rows(b)
ans = 2
octave:28> size(b)
ans =

  2  3
```

Donde `length` da la dimensión de un vector fila o columna, `columns` y `rows` dan las columnas o filas de una matriz y `size` devuelve un vector fila con las dimensiones.

Ya hemos visto la forma más simple de definir una matriz. También se puede definir una matriz en base a otras matrices existentes. Por ejemplo, continuando de lo anterior, podemos construir una matriz concatenando un vector fila junto a otro o encima de otro:

```
octave:34> g=[a a]
g =

  1  2  3  1  2  3

octave:35> g=[a;a]
g =

  1  2  3
  1  2  3
```

Como última nota sobre vectores y matrices, sólo queda comentar que hay una forma taquigráfica de definir un vector secuencial. `m:n` devuelve un vector de números consecutivos desde m hasta n de uno en uno, ambos inclusive. Por ejemplo:

```
octave:36> -2:3
ans =

  -2  -1   0   1   2   3
```

De igual manera, `m:s:n` devuelve un vector de números consecutivos que van desde m hasta n de s en s. Por ejemplo:

```
octave:37> 7:-2:1
ans =

  7  5  3  1
```

De las operaciones con cadenas diremos que se tratan como vectores y que hay muchas funciones para su manejo pero que no nombraremos aquí. Las operaciones con matrices y vectores son las usuales. De resto, comentar que también existen tipos de datos booleanos.

14.3. Variables y expresiones

Octave te permite llamar a las variables con secuencias de cualquier longitud de letras, números y subrayados, pero sin empezar en dígito. Los nombres son sensibles a mayúsculas/minúsculas. Ya hemos tenido ejemplos anteriormente.

Una vez que ha sido definida una variable, puede ser útil conocer el comando `who` que nos da una lista de variables definidas, y `whos` que nos muestra más información.

```
octave:15> who

*** local user variables:

a  b  c  d  g

octave:16> whos

*** local user variables:

prot  type                     rows   cols  name
====  ====                     ====   ====  ====
 rwd  matrix                      1      3  a
 rwd  matrix                      2      3  b
 rwd  string                      1     10  c
 rwd  struct                      -      -  d
 rwd  matrix                      2      3  g
```

En caso de que queramos eliminar una variable de memoria usaremos la orden `clean nombredevariable`. En caso de que queramos borrar todas las variables simplemente teclearemos `clear` sin argumentos.

La primera expresión que podemos aprender es la *extracción* de valores desde una matriz. En el caso más sencillo de extraer un elemento, por ejemplo, de la fila 1 y columna 2, escribimos `a(1,2)`. El operador `:` (dos puntos) nos vale de comodín, y nos valdrá para extraer, por ejemplo, toda la fila 1 escribiendo `a(1,:)` o toda la columna 2 escribiendo `a(:,2)`. También podemos incluir rangos en los argumentos, por ejemplo, para sacar las columnas 2 y 3 sería `a(:,[2 3])`.

Cabe señalar que las matrices en octave se crean dinámicamente, bajo demanda, es decir, que no hay que asignarles previamente un tamaño prefijado. Por ejemplo, este código al final contendrá un vector de 10 elementos:

```
for i = 1:10
  a (i) = sqrt (i);
endfor
```

Aunque no debemos hacer uso de esta creación dinámica salvo en casos irremediables, pues en el anterior ejemplo hemos reasignado memoria 10 veces, pues hemos creado primero un vector de tamaño 1, luego otro de tamaño 2, etc. En cambio, si hubiésemos previamente creado un vector de ceros de tamaño 10, con `zeros(1,10)`, el bucle no necesitaría cambiar el tamaño del vector. Aunque la mejor medida para conseguir velocidad es aprovechar la notación matricial de octave, y en vez del anterior ejemplo, escribir `a=sqrt(1:10);`, que hace lo mismo en una sola línea y es aún más rápido. Dejemos el inciso y sigamos.

Las expresiones más usadas son las aritméticas, que detallamos como:

X+Y y X-Y La suma y la resta. Si ambos operandos son matrices, el número de filas y columnas deben coincidir. Si uno es un número, su valor es añadido o restado respectivamente a todos los elementos del otro operando.

X.+Y y X.-Y Suma o resta elemento a elemento. Son equivalentes a las anteriores.

X*Y Producto de matrices. El número de columnas de X debe coincidir con el de filas de Y.

X.*Y Producto de dos elementos. Si ambos operandos son matrices, sus dimensiones deben coincidir.

X/Y División por la derecha. Esto es conceptualmente equivalente a la expresión `(inverse (y´) * x´)´` usada para resolver sistemas lineales, pero que se calcula sin hacer la inversa de `y´`.

X./Y División por la derecha elemento a elemento.

X\Y División por la izquierda. Esto es conceptualmente equivalente a la expresión `inverse (x) * y` pero que se calcula sin hacer la inversa de `x`.

X.\Y División por la izquierda elemento a elemento.

X^Y o XY** Potencia. Si X e Y son escalares, devuelve X elevado a la potencia de Y. Si X es escalar e Y es una matriz cuadrada, se calcula usando autovalores. Si X es una matriz cuadrada e Y es un escalar, la matriz se calcula con repetidas multiplicaciones si Y es entero y usando autovalores si no es entero.

X.^Y o X.Y** Potencia elemento a elemento. Si ambos sin matrices, sus dimensiones deben coincidir.

X.´ Traspuesta.

X´ Traspuesta compleja conjugada. Para valores reales es equivalente a la traspuesta. Para valores complejos, es equivalente a la expresión `conj(X.´)`.

Luego tenemos los operadores de comparación, que pueden ser < menor, <= menor o igual, == igual, >= mayor o igual, > mayor y cualquiera de estos `!=`, `~=` o <> distinto. Aplicados entre matrices devuelven una matriz de unos y ceros, con unos en los elementos donde la condición se cumpla y cero donde no se cumpla. También tenemos operadores booleanos y de incrementación y decrementación, pero que no los vamos a nombrar.

14.4. Control de flujo

Las sentencias para el control de flujo son las típicas de cualquier lenguaje de programación. Pondremos un simple ejemplo de cada una para que sirva de referencia en el caso que tuvieras que usarlas. Todas las sentencias se caracterizan por finalizar con una sentencia `end*`.

14.4.1. if

El caso más general de `if` es la estructura if-elseif-else-endif, que se muestra aquí.

```
if (rem (x, 2) == 0)
  printf ("x es par\n");
elseif (rem (x, 3) == 0)
  printf ("x es impar y divisible por 3\n");
else
  printf ("x es impar\n");
endif
```

14.4.2. switch

Es de reciente implantación así que se considera experimental (con respecto a la versión 2.0.5). Es una mera traducción de un bloque if, así que las condiciones siempre se miran de arriba a abajo y se ejecuta el código de la primera que sea verdadera y luego se sale. La forma de uso es la siguiente:

```
switch x
  case (x>=5)
    printf("x es mayor o igual que 5\n");
  case (x>=2)
    printf("x es mayor o igual que 2 y menor que 5\n");
  otherwise
    printf("x es menor que 2\n");
endswitch
```

14.4.3. while

Primero se comprueba la condición, y si es válida se ejecuta el cuerpo y el proceso se repite. Si no es válida se sale.

```
fib = ones (1, 10);
i = 3;
while (i <= 10)
  fib (i) = fib (i-1) + fib (i-2);
  i++;
endwhile
```

14.4.4. do-until

Es el mismo caso que el `while`, pero comprobando la condición al final, después de haber ejecutado una vez el cuerpo. Es decir, se ejecuta el cuerpo y se comprueba la condición, y si es válida se ejecuta el cuerpo de nuevo y el proceso se repite. Si no es válida se sale.

```
fib = ones (1, 10);
i = 2;
do
  i++;
  fib (i) = fib (i-1) + fib (i-2);
until (i == 10)
```

14.4.5. for

La variable del bucle se asigna consecutivamente a todos los elementos de un vector. En el caso de ejemplo, la variable i toma los valores 3, 4, ..., 9 y 10, ejecutando luego el cuerpo.

```
fib = ones (1, 10);
for i = 3:10
  fib (i) = fib (i-1) + fib (i-2);
endfor
```

14.4.6. break/continue

La sentencia break permite salir de un bucle `for` o `while` y seguir la ejecución del programa en la sentencia siguiente al bucle. La sentencia `continue` simplemente se salta todo el cuerpo y realiza la siguiente iteración del bucle sin salirse.

14.4.7. unwind_protect/try

Octave permite dos formas limitadas de manejo de excepciones. Más información en el manual.

14.5. Funciones

Pongamos un ejemplo de como se define una función en octave. En este ejemplo, a la función se le pasa un argumento que debe ser un vector, y se devuelve la media de sus componentes:

```
function retval = avg (v)
  # ayuda: la funcion avg(v) devuelve el promedio
  # de las componentes del vector v

  retval = sum (v) / length (v);
endfunction
```

Como se puede apreciar, es una función con un argumento y un valor de retorno. En octave, los argumentos de la función son locales, es decir, que los argumentos son copias de los originales y si se modifican desde dentro de la función, los cambios no son vistos por el llamante. Esto implica que la única forma que tenemos de devolver valores al llamante sea usando valores de retorno como en el ejemplo. Podemos hacer que devuelva más de un valor de retorno, disponiéndolos entre corchetes, como en este otro ejemplo que calcula el máximo de un vector.

Fichero `vmax.m`

```
function [max, idx] = vmax (v)
  # ayuda: vmax(v) devuelve el máximo valor de un vector
  # y la posición que ocupa

  idx = 1;
  max = v (1);
  for i = 2:length (v)
    if (v (i) > max)
      max = v (i);
      idx = i;
    endif
  endfor
endfunction
```

Ejemplo 14.1: La función recibe un vector y devuelve dos valores, el máximo valor encontrado en el vector y la posición donde se encuentra.

Excepto para casos muy simples, no es práctico teclear las funciones en la línea de comandos. Las funciones se suelen guardar en ficheros, que se pueden editar fácilmente y guardarlas para su uso posterior. Existe una pauta que hay que seguir, y es que el nombre del archivo debe ser el nombre de la función con la extensión `.m` en el directorio de trabajo. Por ejemplo, la función vmax que acabamos de definir la tendríamos que guardar en un fichero llamado *vmax.m* para que funcionase.

De igual forma se pueden colocar en ficheros instrucciones sueltas sin formar funciones. Esto se denomina *ficheros script*. Cuando se ejecuta uno de ellos, las instrucciones que contiene se ejecutan como si se escribieran una a una por la línea de comandos.

Cuando se le ordena a octave que ejecute una función el proceso que realiza es el siguiente: la lee y la analiza sintácticamente; en caso que no existan errores, la compila en un formato interno y pasa a ejecutarla. En caso de que el usuario le ordene a octave ejecutarla otra vez, si el fichero no ha sido modificado, octave utiliza la función ya compilada ahorrando el tiempo de lectura y análisis.

Veamos esto con un ejemplo. Cojamos el código fuente de la función anterior, guardémoslo en un fichero llamado *vmax.m* y tecleemos lo siguiente:

```
octave:1> clear
octave:2> who
octave:3> help vmax
vmax is the user-defined function from the file
/home/alberto/cvs/Libro\_CILA/ejemplos/vmax.m

ayuda: vmax(v) devuelve el máximo valor de un vector
y la posición que ocupa
...

octave:3> [max idx]=vmax([5 4 3 2 7 5 4 9 6])
max = 9
idx = 8
octave:4> who

*** currently compiled functions:

vmax

*** local user variables:

idx  max
```

Como estamos viendo, efectivamente `who` al principio, después de limpiar la memoria, no nos decía nada, pero después de ejecutar la función, nos informa que la tiene ya compilada y almacenada para posteriores usos.

Donde se hagan usos más exigentes de octave, este esquema, aunque es inteligente, resulta poco óptimo, pues lo que se hace es traducir la función a un lenguaje intermedio que luego octave interpreta cuando se le manda a ejecutar. Si nuestro uso requiere de la máxima potencia de cálculo, viene bien saber que octave es capaz de ejecutar código objeto compilado con gcc, es decir, código de c, c++, fortran o pascal. Veamos un ejemplo en c++ para darnos cuenta lo sencillo que es y de las posibilidades que nos puede abrir.

Fichero `oregonator.cc`

```
#include <octave/oct.h>

DEFUN_DLD(oregonator, args,, "El 'oregonador'.")
{
  ColumnVector dx(3);

  ColumnVector x(args(0).vector_value());

  dx(0) = 77.27 * (x(1) - x(0) * x(1) + x(0) - 8.375e-06 * pow(x(0), 2));
  dx(1) = (x(2) - x(0) * x(1) - x(1)) / 77.27;
  dx(2) = 0.161 * (x(0) - x(2));

  return octave_value(dx);
}
```

Ejemplo 14.2: Esta función recibe un vector de tres elementos como argumento de entrada y hace un cálculo con esos valores para devolver un resultado.

No vamos a analizar este ejemplo en profundidad, sino solamente dar unas pistas sobre como está hecho. En primer lugar vemos un `include <octave/oct.h>`. Lo que hace es definir todos los tipos de datos de octave y las funciones nativas como clases y métodos de C++. Con esto queremos decir que si en octave podíamos hacer `length(vector)` para obtener las dimensiones de un vector, en *C++* podremos hacer lo mismo usando `vector.lenght`. Entendido esto, el resto son meras particularidades sintácticas específicas de C++ en las que no entraremos. Para compilar este código, vayámonos a un shell y escribamos lo siguiente:

```
alberto@baifito:ejemplos\$ ls oregonator.*
oregonator.cc
alberto@baifito:ejemplos\$ mkoctfile oregonator.cc
alberto@baifito:ejemplos\$ ls oregonator.*
oregonator.cc  oregonator.o  oregonator.oct
```

El fichero *oregonator.o* es un código objeto como cualquier otro que se obtiene con gcc. El fichero *oregonator.oct* es la función recompilada para octave. Para probarla, entremos en octave y tecleemos lo siguiente:

```
octave:1> help oregonator
oregonator is the dynamically-linked function from the file
/home/alberto/cvs/Libro_CILA/ejemplos/oregonator.oct

El 'oregonador'.

...
octave:2> oregonator([1 2 3])
ans =

   77.269353
   -0.012942
   -0.322000
```

Como verás, en la ayuda de la función aparece el texto que definimos en el código fuente, así como una advertencia de que la función está dinámicamente linkada a las librerías de octave. En el segundo paso vemos que la ejecución de la función tambien funciona. Eso sí, si no pasamos los parámetros correctamente veremos como un *Segmentation Fault* cierra nuestro octave. El problema es que no comprobamos el tipo de dato en el código de C++, pero añadiendo las comprobaciones pertinentes podremos manejar estos casos excepcionales y no se dará este problema.

La ejecución de código compilado como éste puede llegar a ser hasta 10 veces más rápido que el código interpretado, en fichero con extensión `.m`. Además, podemos enlazar (link) funciones de otros lenguajes simplemente escribiendo una capa tal como hemos visto que se encargue de leer/escribir los datos en estructuras de octave. Y como en esos otros lenguajes se pueden hacer ventanas gráficas o acceder a periféricos, eso significa que en octave también se podrá. Las rutinas de procesamiento de imágenes que veremos (o las de sonido, que no veremos) son un ejemplo de ello. Para más ejemplos, consultar los ficheros con extensión `.cc` de la distribución de octave.

14.6. Representación gráfica

En los siguientes ejemplos entraremos en el campo de la representación gráfica, que también es sencillo (NOTA: no olvidarse los puntos y comas al final de línea pues los vectores son algo largos para estarlos visualizando, y pulsar `q` para cerrar las gráficas).

Para hacer representaciones gráficas deberás haber ejecutado Octave desde un shell dentro de las X puesto que la representación gráfica se realiza usando Gnuplot, cuya forma de funcionar por defecto es en entorno X-Window. No nos adentraremos demasiado pues describiremos Gnuplot en otro capítulo.

Presentación en una dimensión

La función `plot(vector)` o `plot(x,y)` es muy sencilla de usar. La diferencia entre ambas llamadas es que cuando presentamos un vector, el eje x se numera automáticamente de 1 en adelante, mientras que la segunda forma de llamarla, el valor del eje x está definido por nosotros. Veamos el siguiente ejemplo que presenta un período de una senoidal.

```
octave:25> x=[0:0.01:1];
octave:26> y=sin(2*pi*x);
octave:27> plot(x) # presentamos una recta
octave:28> plot(y) # presentamos una senoidal
octave:29> plot(x,y) # senoidal, pero con eje x bien puesto
```

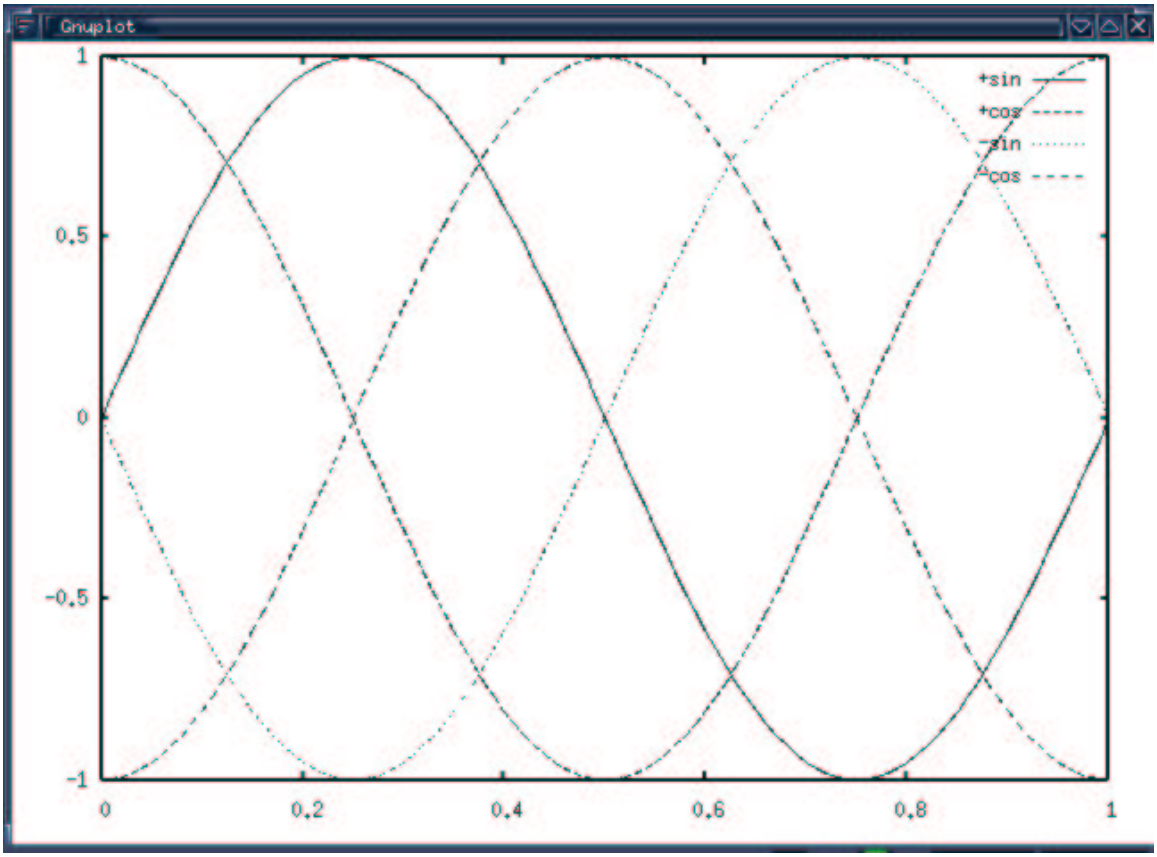


Figura 14.1: Múltiples líneas por gráfica.

Se pueden presentar varias gráficas en una usando la opción `hold on` y se pueden añadir títulos a las gráficas con un tercer párametro a la función `plot`. Veamos un ejemplo:

Fichero `sinus.m`

```
#!/usr/bin/octave -qf
x=[0:0.01:1];       # rango variacion eje x
clearplot;          # borramos grafica
subplot(1,1,1);     # un unico plot centrado
axis("auto","normal");     # ejes automaticos
hold on;            # superpone siguientes graficos
plot(x, +sin(2*pi*x), ';+sin;')
plot(x, +cos(2*pi*x), ';+cos;')
plot(x, -sin(2*pi*x), ';-sin;')
plot(x, -cos(2*pi*x), ';-cos;')
```

Ejemplo 14.3: Mostramos gráficas sobreimpresas entre ellas y les asignamos títulos.

Octave y Gnuplot permiten diferentes estilos en las gráficas. Por ejemplo, en polares, de la forma `polar(`θ`,`ρ`)`:

Fichero `polares.m`

```
#!/usr/bin/octave -qf
x=[0:0.01:2*pi];  # rango variacion angulo
clearplot;        # borra grafico
axis([-1 1 -1 1], "square");   # ejes manuales
hold off;         # no superpone siguientes graficos
subplot(2,2,1);   # cuadrante 1 de 4
polar(x,ones(size(x)));       # circulo
```

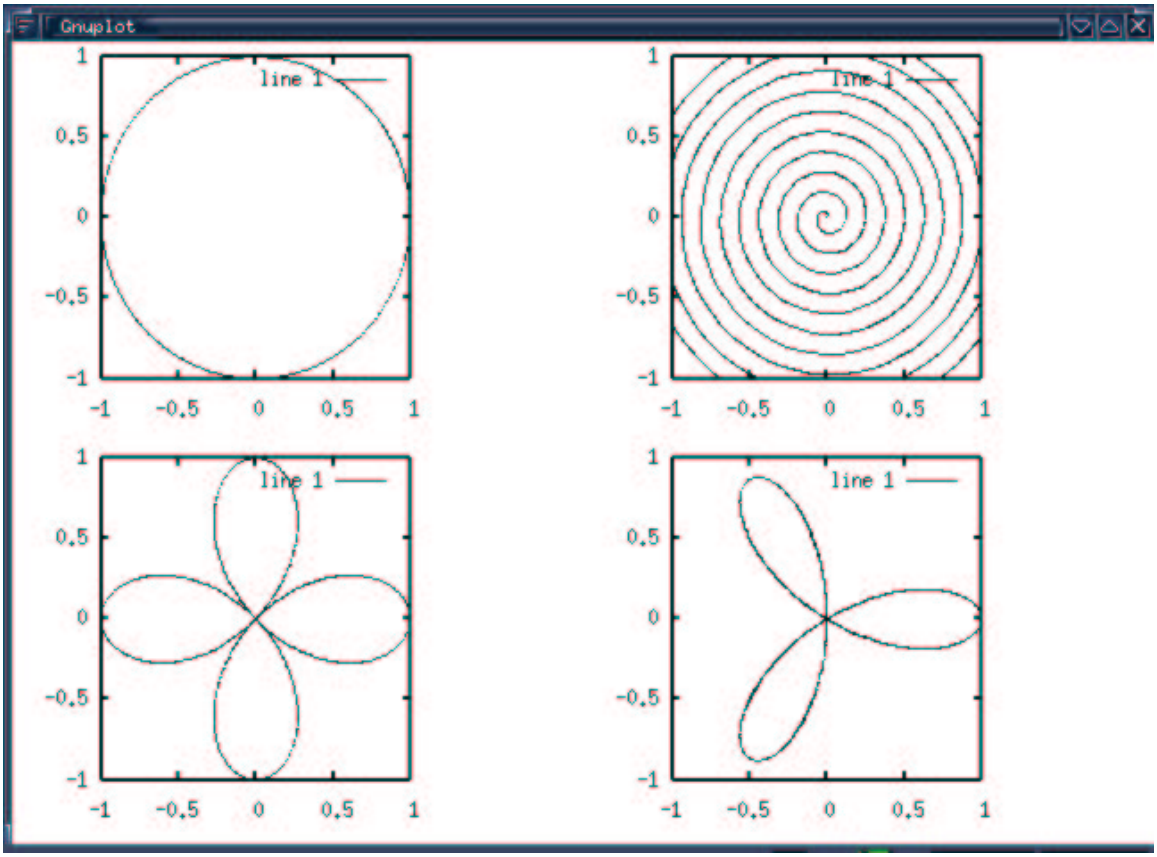


Figura 14.2: Diagramas en coordenadas polares

```
subplot(2,2,2);   # cuadrante 2 de 4
polar(10*x,x/5);    # espiral de arquímedes
subplot(2,2,3);   # cuadrante 3 de 4
polar(x,cos(2*x));# rosa de cuatro petalos
subplot(2,2,4);   # cuadrante 4 de 4
polar(x,cos(3*x));# rosa de tres petalos
```

Ejemplo 14.4: Mostramos cuatro gráficas diferentes en polares en cuatro cuadrantes.

También en forma de histograma, adornando las gráficas con títulos.

Fichero `histo.m`

```
#!/usr/bin/octave -qf
y=randn(100,1);   # matriz num aleatorios normal
clearplot;        # borra grafico
axis("auto","normal"); # ejes automáticos
hold off;         # no superpone siguientes graficos
subplot(1,2,1);   # cuadrante 1 de 2
title("datos aleatorios en eje y");
plot(y,'*');      # plotea los datos aleatorios con *
subplot(1,2,2);   # cuadrante 2 de 2
title("histograma");
hist(y,20,1);     # histograma de 20 barras normalizado a 1
```

Ejemplo 14.5: Mostramos un histograma y sus datos de partida, con títulos sobre las gráficas.

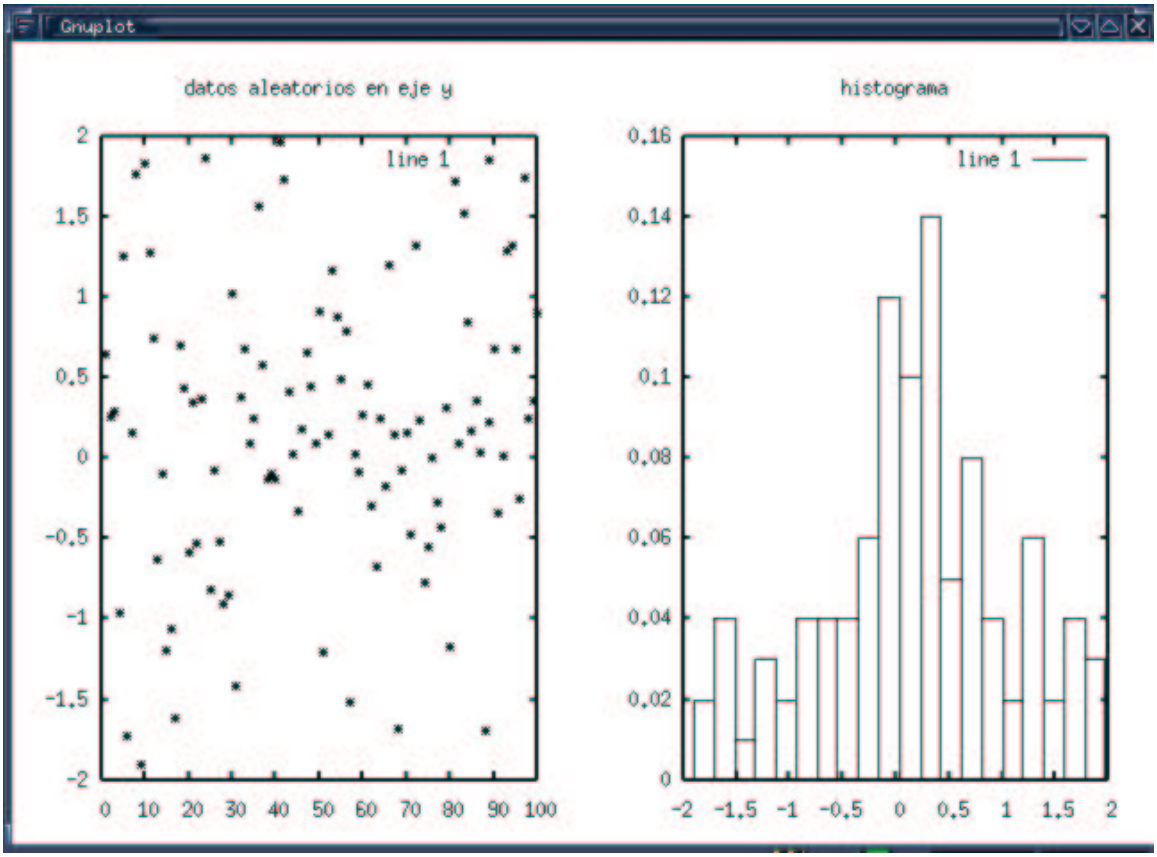


Figura 14.3: Representación de histogramas

Representación en 3D

La función `mesh(x,y,z)` hace una representación 3D dados dos vectores `x` e `y` para los ejes y una matriz bidimensional `z` que será la coordenada Z en un espacio tridimensional. Otra función llamada `contour(x,y,z)` con los mismos argumentos que `mesh()`, dibujará las curvas de nivel de la superficie. En este ejemplo, lo más complicado será generar una matriz `z` bonita. Una vez tenemos la matriz y los ejes, las dos llamadas son directas.

Fichero `meshplot.m`

```
#!/usr/bin/octave -qf
1; # limpia memoria

function configura
    hold off;                # no superpone siguientes graficos
    clearplot();             # limpiamos
    axis("auto","normal");   # ejes automáticos
    subplot(1,1,1);          # nos ponemos en una sola ventana
    xlabel("eje x");         #
    ylabel("eje y");         # ponemos etiquetas
    zlabel("eje z");         #
endfunction

x=[-10:0.5:10];          # vector del eje x
y=[-10:0.5:10];          # vector del eje y
[mx,my]=meshgrid(x,y);   # genera matrices de ejes
mr=sqrt(mx.^2+my.^2);    # matriz que contiene el radio
mz=sin(mr)./mr;          # funcion z=sin(r)/r
configura();
subplot(1,2,1);          # cuadrante 1 de 2
```

```
title("plano eje x creciente, eje y constante");
mesh(x,y,mx);
subplot(1,2,2);         # cuadrante 2 de 2
title("plano eje x constante, eje y creciente");
mesh(x,y,my);
pause(5);               # pausar 5 segundos
configura();
subplot(1,2,1);         # cuadrante 1 de 2
title("superficie 3d");
mesh(x,y,mz);
subplot(1,2,2);         # cuadrante 2 de 2
title("curvas de nivel");
contour(x,y,mz);
```

Ejemplo 14.6: Mostramos una sinc en 3 dimensiones, así como los planos usados para generarla.

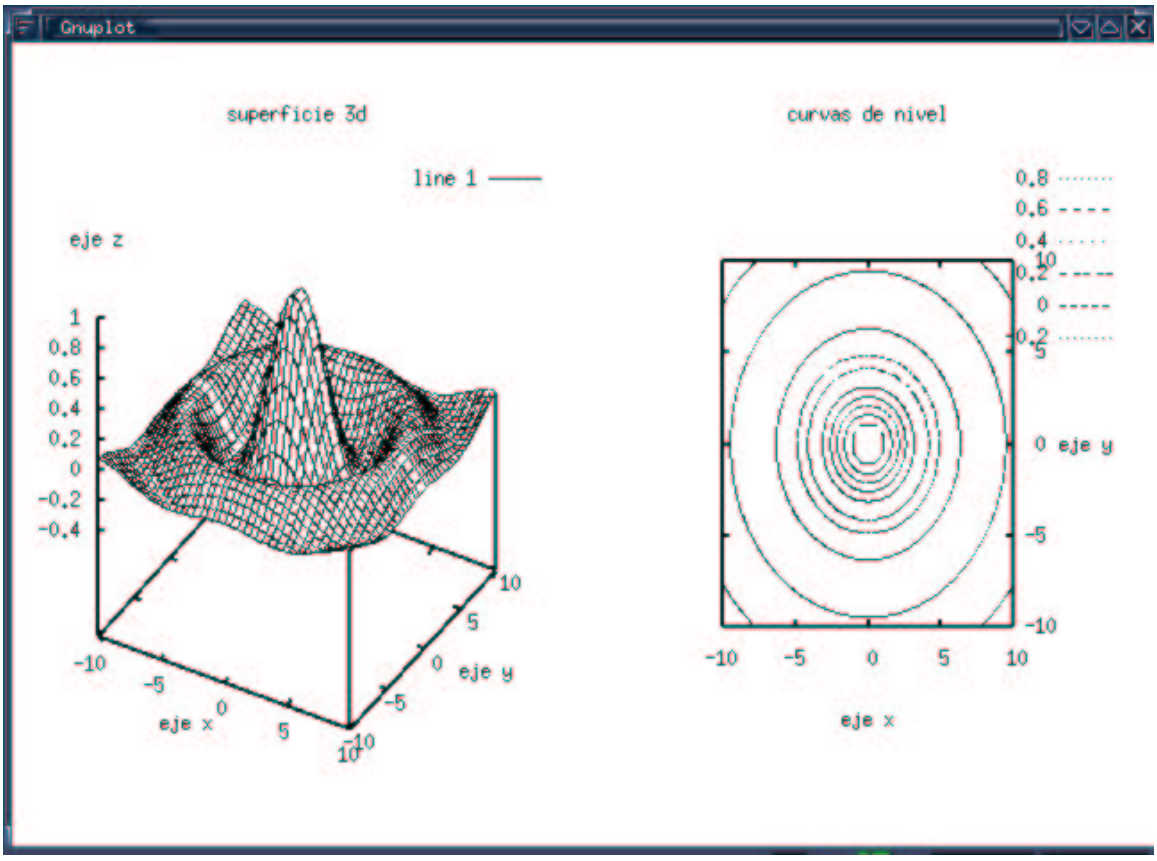


Figura 14.4: Gráficas en tres dimensiones

14.7. Matrices

Octave tiene una amplia colección de funciones para trabajar con matrices y vectores. Las veremos en un ejemplo.

```
octave:20> c=diag([1,2,3,4]) # creación de matrices diagonales
c =
  1  0  0  0
  0  2  0  0
  0  0  3  0
  0  0  0  4
```

```
octave:21> inv(c) # inversa de una matriz
ans =
  1.00000  0.00000  0.00000  0.00000
  0.00000  0.50000  0.00000  0.00000
  0.00000  0.00000  0.33333  0.00000
  0.00000  0.00000  0.00000  0.25000

octave:22> det(c) # determinante de una matriz
ans = 24

octave:23> eye(4) # matriz identidad de dimensión 4
ans =
  1  0  0  0
  0  1  0  0
  0  0  1  0
  0  0  0  1

octave:15> ones(2,3) # matriz 2x3 repleta de unos
ans =

  1  1  1
  1  1  1

octave:16> zeros(1,7) # matriz 1x7 repleta de ceros
ans =

  0  0  0  0  0  0  0

octave:24> rand(4,3) # matriz de números aleatorios 4x3
ans =
  0.85927  0.43700  0.85462
  0.88050  0.27016  0.52905
  0.58098  0.54402  0.29237
  0.41791  0.73324  0.45943

octave:5> randn(3,2) # matriz de números aleatorios gaussiana
ans =

   0.64262  -1.03740
   0.31010  -1.38565
  -0.64096   0.58650

# resta cada elemento con su anterior
octave:7> diff([1 2 4 5 7 9 11 14])
ans =

  1  2  1  2  2  2  3

# encuentra índices elementos no nulos
octave:8> find ([0 0 0 0 0 0 1 0 0 0 5 0])
ans =

   7  11
# subdivide linealmente el intervalo [1,10] en 8 puntos.
octave:9> linspace (1, 10, 8)
ans =
```

```
  1.0000  2.2857  3.5714  4.8571  6.1429  7.4286  8.7143 10.0000

# subdivide logarítmicamente el intervalo [10^0,10^3] en 6 puntos.
octave:13> logspace (0, 3, 6)
ans =

   1.0000   3.9811  15.8489  63.0957  251.1886 1000.0000

# factorización lu de una matriz
octave:26> [a,b,c]=lu([1,3,2;3,2,1;3,2,6])
a =

  1.00000  0.00000  0.00000
  0.33333  1.00000  0.00000
  1.00000  0.00000  1.00000

b =

  3.00000  2.00000  1.00000
  0.00000  2.33333  1.66667
  0.00000  0.00000  5.00000

c =

  0  1  0
  1  0  0
  0  0  1

octave:27> [a,b,c]=qr([1,3,2;3,2,1;3,2,6]) # factorización QR
a =

  -0.312348  -0.752156  -0.580259
  -0.156174  -0.561851   0.812362
  -0.937043   0.344361   0.058026

b =

  -6.40312  -3.12348  -3.59200
   0.00000  -2.69145  -1.40463
   0.00000   0.00000   2.03091

c =

  0  0  1
  0  1  0
  1  0  0
```

14.8. Ecuaciones diferenciales

Fichero ode.m

```
#!/usr/bin/octave

clear;
```

```
function xdot = f (x, t)
  xdot = zeros (3,1);
  xdot(1) = 77.27 * (x(2) - x(1)*x(2) + x(1) - 8.375e-06*x(1)^2);
  xdot(2) = (x(3) - x(1)*x(2) - x(2)) / 77.27;
  xdot(3) = 0.161*(x(1) - x(3));
endfunction

# condicion inicial
x0 = [ 4; 1.1; 4 ];
# generación del eje t para la simulacion
t = linspace (0, 50, 100);
# simulacion
y = lsode ("f", x0, t);
hold off;
subplot(3,1,1);
plot(t,y(:,1),';x(1);');
subplot(3,1,2);
plot(t,y(:,2),';x(2);');
subplot(3,1,3);
plot(t,y(:,3),';x(3);');
```

Ejemplo 14.7: Resuelve una EDO ordinaria de tercer orden y muestra por pantalla la evolución de los `x(t)`.

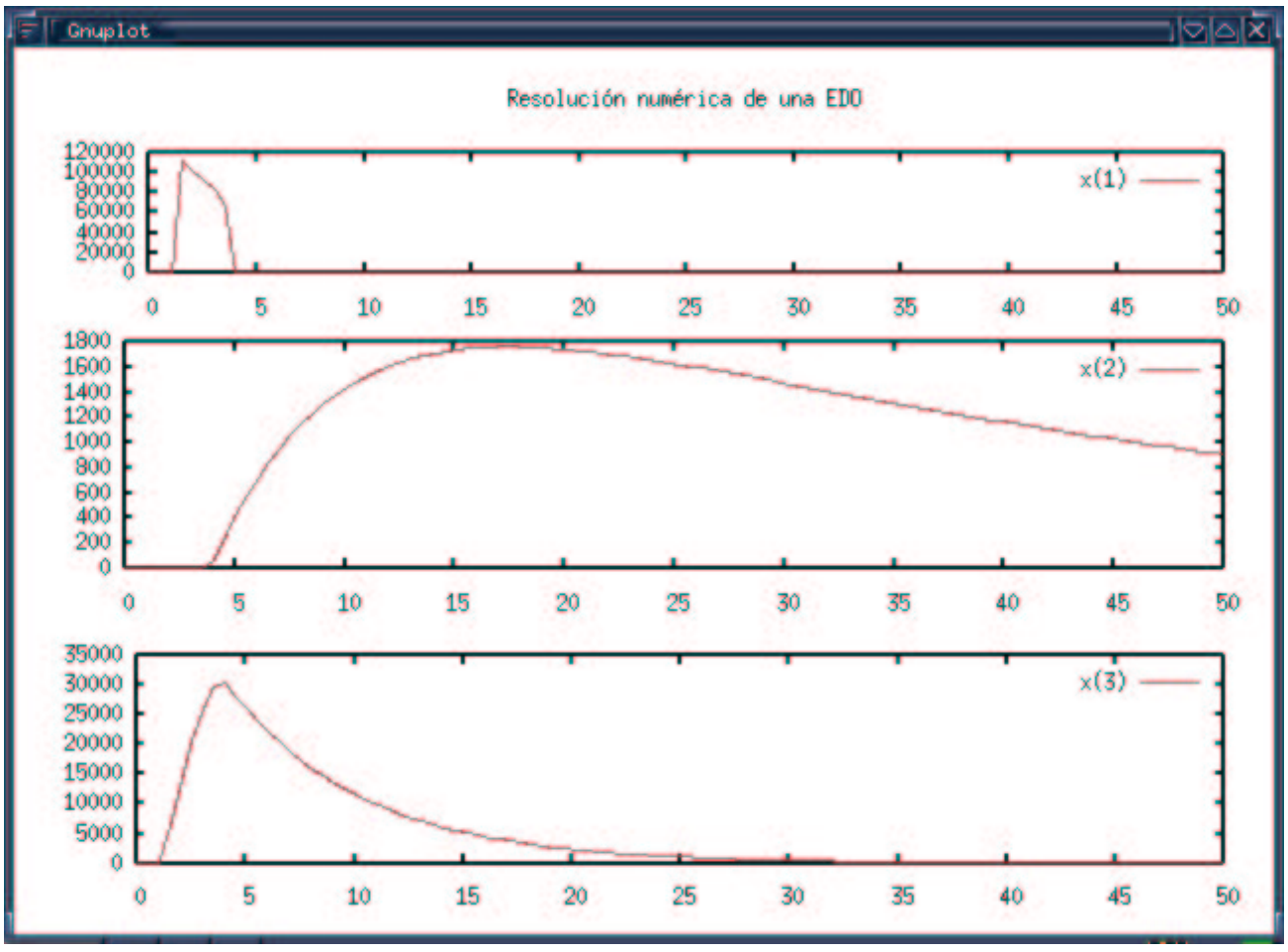


Figura 14.5: Resolución numérica de EDO

14.9. Polinomios

Un polinomio de grado r en octave se presenta como un vector de dimensiones $r + 1$. A partir de él se pueden realizar operaciones.

```
octave:10> a=[1,2,3]; # a(x)=x^2+2x+3
octave:11> b=[3,2,3,2]; #b(x)=3x^3+2x^2+3x+2
octave:19> polyout(a)
1*s^2 + 2*s^1 + 3
octave:18> polyout(b)
3*s^3 + 2*s^2 + 3*s^1 + 2
octave:20> polyout(conv(a,b)) # producto de polinomios
3*s^5 + 8*s^4 + 16*s^3 + 14*s^2 + 13*s^1 + 6
octave:13> [coc, resto]=deconv([3 8 16 14 13 6],b); # división
octave:24> polyout (coc) # el cociente de la división
1*s^2 + 2*s^1 + 3
octave:25> polyout (resto)         # el resto de la división
0*s^5 + 0*s^4 + 0*s^3 + 0*s^2 + 0*s^1 + 0
octave:26> polyout(polyderiv (b))  # la derivada de b(x)
9*s^2 + 4*s^1 + 3
octave:27> polyout(polyinteg (b))  # la integral de b(x)
0.75*s^4 + 0.666667*s^3 + 1.5*s^2 + 2*s^1 + 0
octave:17> polyval (b,2)           # b(x) evaluado en x=2
ans = 40
```

14.10. Teoría de control

La versión 2.1 de octave incorpora una Toolbox de Control. El control es una rama de la ingeniería dedicada a modelar sistemas mediante las ecuaciones diferenciales que relacionan su salida con su entrada y predecir su comportamiento frente a diferentes entradas. En la toolbox se utilizan estructuras para abstraer al usuario el concepto de función de transferencia de un sistema.

14.11. Procesamiento de señales

En esta categoría están todas las funciones dedicadas a trabajar con espectros de señales. Tanto la *FFT*, como filtros digitales *FIR* e *IIR*, diferentes tipos de ventanas, o cálculo de modelos ARMA.

La transformada de Fourier discreta, más conocida por el nombre de su algoritmo FFT (**Fast Fourier Transform**), es la versión discreta y periódica de la transformada exponencial de Fourier. Es una función muy usada en ingeniería y en la vida real. La tomaremos como la función para la ingeniería por antonomasia, y en este ejemplo veremos qué sencillo resulta obtener la transformada de Fourier discreta de una senoidal y un pulso.

Fichero `fftview.m`

```
#!/usr/bin/octave
clear;
hold off;
T=0.1;                      # periodo de muestreo
N=100;                      # numero de muestras
W=3*(2*pi/T);               # frecuencia de la senoidal
A=20*T;                     # ancho del escalon
t=T*[0:N];                  # escala temporal
f=(2*pi/T)*[-N/2:N/2]/N;    # escala en frecuencias
# la primera es la fft de un seno de 20Hz
title("Seno de 20Hz");
x1=sin(2*pi*W*t);
```

```
y1=fft(x1)/length(t);
y1=fftshift(y1);
subplot(3,1,1);
clearplot;
plot(t,x1,';sinusoidal;');
subplot(3,1,2);
clearplot;
plot(f,abs(y1),';modulo;');
subplot(3,1,3);
clearplot;
plot(f,arg(y1),';fase;');
pause(5);
# la segunda es la de un escalon centrado de ancho A sg.
x2=abs(t-T*N/2)<(A/2);
y2=fft(x2)/length(t);
y2=fftshift(y2);
subplot(3,1,1);
clearplot;
plot(t,x2,';escalon;');
subplot(3,1,2);
clearplot;
plot(f,abs(y2),';modulo;');
subplot(3,1,3);
clearplot;
plot(f,arg(y2),';fase;');
```

Ejemplo 14.8: Muestra por pantalla el módulo y la fase de dos funciones: una sinusoidal (que debe dar dos deltas de dirac) y un pulso (que debe dar una sinc).

14.12. Tratamiento de imágenes

Por último veamos un ejemplo de cómo se pueden cargar, realizar modificaciones, visualizar y salvar imágenes usando las rutinas de octave. Con esto podremos realizar muchas operaciones útiles en procesado, realzado y análisis de imágenes, útiles en muchas áreas de las ciencias.

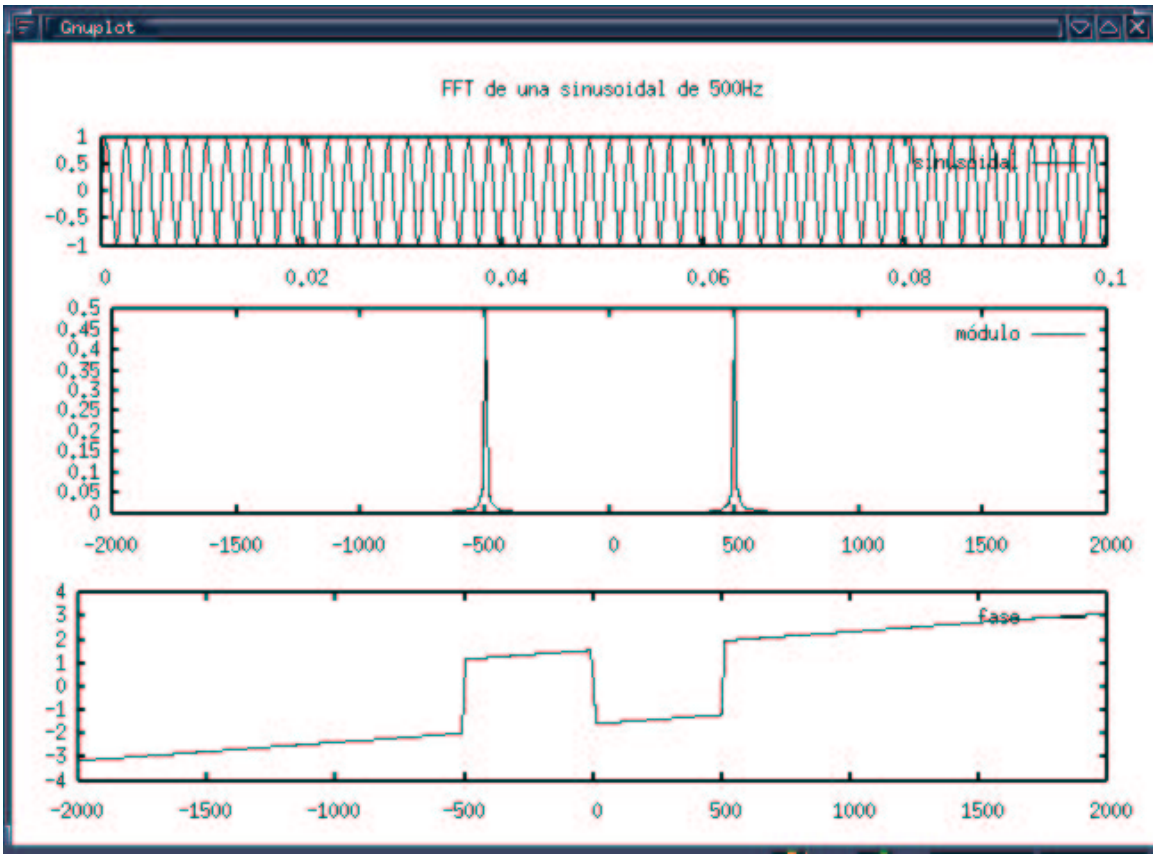
En primer lugar suponemos sabido que una imagen se puede entender como una matriz donde cada punto tiene un color diferente. Cada punto se llama *píxel*. Hay muchas formas de representar pixeles, y octave maneja tres de ellas, que son las siguientes:

escala de grises En una imagen en escala de grises cada píxel se representa por un valor de punto flotante comprendido entre 0, que significa negro, y 1, que significa blanco. Por tanto, la representación de una imagen es una matriz llena de valores comprendidos entre 0 y 1. En este caso, un valor de 0.5 representaría un color gris que tiene partes iguales de blanco y negro mientras que un valor de 0.2 representaría un color que tiene 20 % de negro y el resto de blanco.

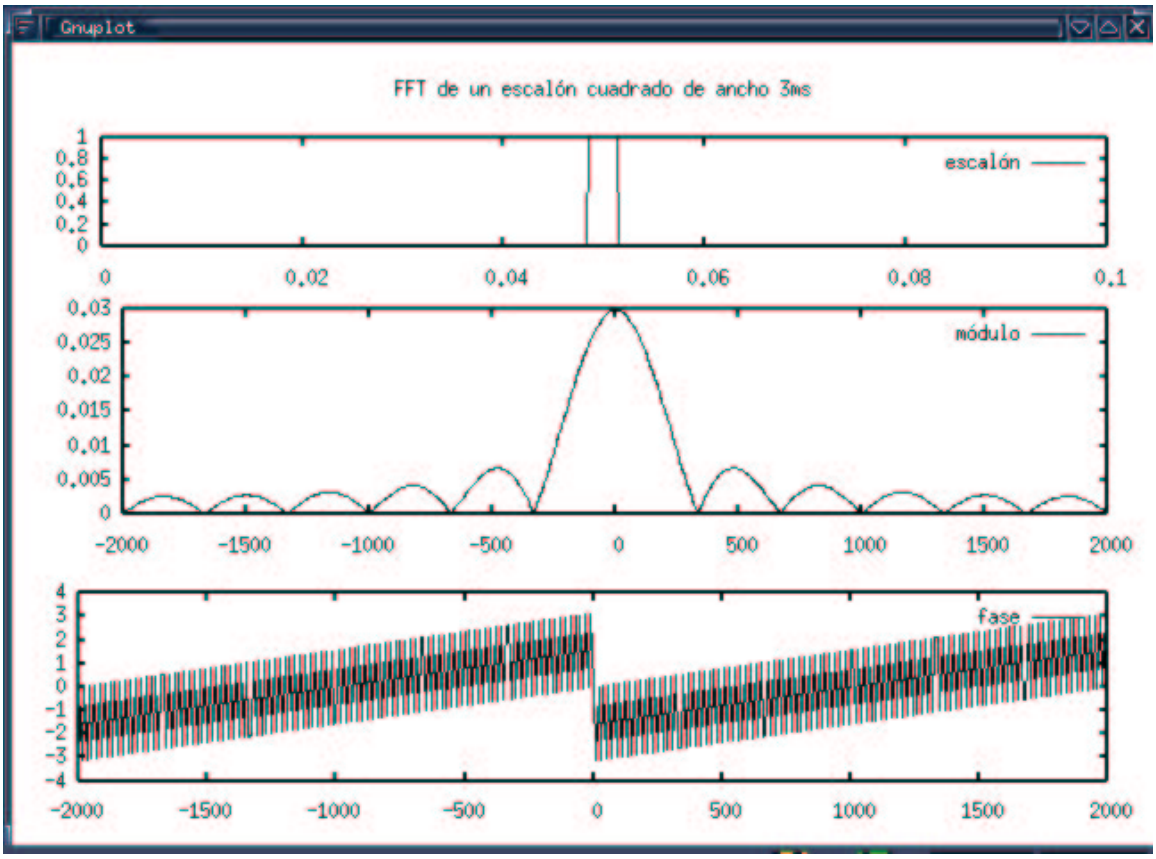
color RGB Cualquier color en la naturaleza se puede aproximar (no es una correspondencia exacta) como suma de sus tres componentes de color, a saber: rojo (R, red), verde (G, green) y azul (B, blue). De esta forma, cada píxel de una imagen en color se puede representar como un terna de valores de sus tres componentes. Por tanto, una imagen en formato RGB son tres matrices del mismo tamaño, donde cada una almacena los valores de una componente.

color indexado Como en una imagen suele haber colores repetidos, esta representación lista por un lado los colores, de manera consecutiva y sin repetirse, y por otro lado los pixels, cuyo color se almacena buscando el valor en la tabla de colores y almacenando el índice correspondiente en el vector.

La función usada para representar imágenes en pantalla es `imshow()`. En primer lugar la usaremos para representar valores en escala de grises. Vamos a probar creando una matriz que tenga variedad de valores y presentándola. Veamos el ejemplo de uso de esta función:

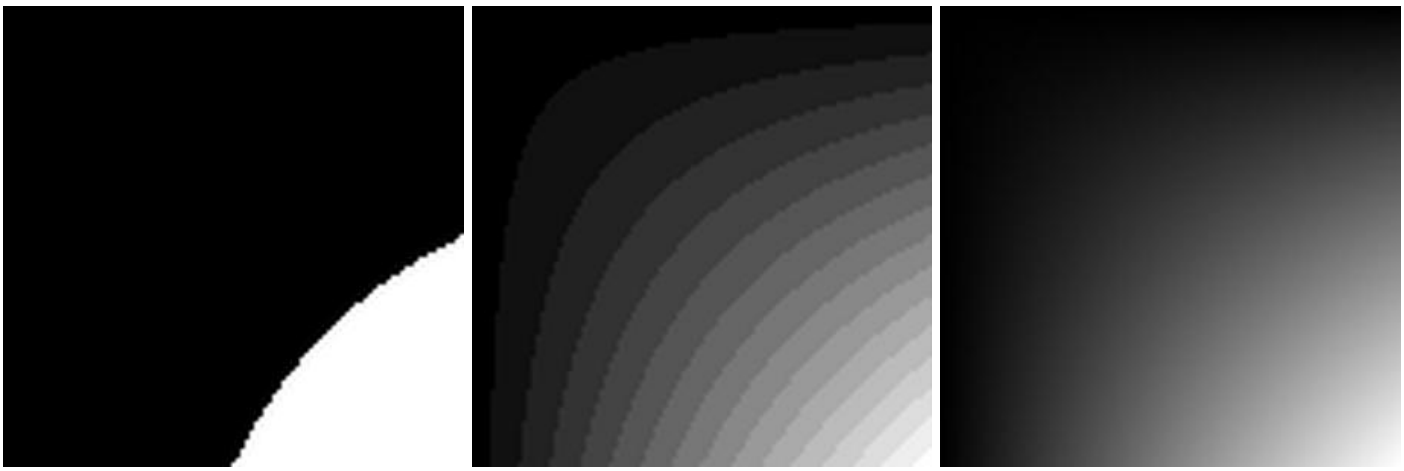


(a) Ejemplo de FFT aplicado a una sinusoidal



(b) Ejemplo de FFT aplicado a una función escalón

```
octave:15> a=0:0.01:1;
octave:16> ma=a'*a;
octave:17> imshow(ma,2)
octave:18> imshow(ma,16)
octave:19> imshow(ma,256)
```



(c) 2 niveles (d) 16 niveles (e) 256 niveles

Figura 14.6: El gradiente anterior visualizado a varios niveles de gris

Observando la matriz `ma` observamos que todos sus valores están en el rango $[0, 1]$. Si vemos su tamaño, vemos que es 100x100. Las diferentes llamadas a `imshow` se diferencian en el segundo parámetro que representa el número de colores discretos en el que presentará la imagen continua. En el primer caso veremos solamente dos colores, en la segunda 16 y en la tercera 256.

Probemos ahora a trabajar con una imagen en color. La función `loadimage()` se utiliza para cargar imágenes de un archivo. Las imágenes se devuelven en formato indexado, donde `c` representará la matriz y `cmap` representará los colores. Veamos el ejemplo:

```
octave:26> [c,cmap]=loadimage("tux.img");
octave:27> imshow(c,cmap)
```

Como verás, se visualiza con la misma función. Ahora, usemos esta imagen como base para jugar un poco. Primero, transformémosla a un formato más manejable. Empezaremos pasándola a `RGB` con la función `ind2rgb` transforma la imagen de indexada a RGB. Luego, con `imshow` la representaremos.

```
ctave:28> [r,g,b]=ind2rgb (c,cmap);
octave:29> imshow(r,g,b)
octave:30> imshow(g,r,b)
octave:31> imshow(b,g,r)
```

Observamos que cada canal contiene sus valores, y cambiando el orden hacemos creer a imshow que los valores son diferentes y nos muestra imágenes con los colores trasladados. Podemos probar también sustituyendo las matrices `r`, `g` o `b` por unos o ceros y comprobar el efecto de quitar o añadir canales. El trabajo con imágenes RGB es complejo, así que usaremos una imagen en escala de grises para seguir efectuando nuestras pruebas.

```
octave:32> g=ind2gray (c,cmap);
octave:35> imshow(g,2)
octave:36> imshow(g,4)
octave:37> imshow(g,16)
octave:65> imshow(g,256)
```

(a) `imshow(r,g,b)` (b) `imshow(g,r,b)` (c) `imshow(b,g,r)`

Figura 14.7: Los colores de la imagen vienen representados por sus componentes RGB. Si se invierten, los colores cambian.

Comenzaremos los análisis con un filtrado/realzado. Estas operaciones se realizan barriendo todos los píxels de la imagen y creando una nueva imagen donde cada píxel es una ponderación de los valores de cada píxel y sus vecinos según unas matrices preestablecidos. Las matrices

$$\frac{1}{5}\begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Filtro pasa-baja Filtro pasa-alta Detector de bordes

Ahora veamos como se aplicaría un filtro. Este filtro es el primero, que genera una imagen donde cada píxel es un promedio entre el correspondiente en la imagen original y sus cuatro vecinos por cada lado.

```
octave:41> h=zeros(256,256);
octave:42> for i=2:255;
>             for j=2:255;
>                h(i,j)=(g(i,j)+g(i-1,j)+g(i+1,j)+ \
>                       g(i,j-1)+g(i,j+1))/5;
>             endfor;
>          endfor;
octave:43> imshow(g,256)
octave:44> imshow(h,256)
```

Mostrando la imagen original y retocada comprobamos que el efecto que tiene es el de suavizar los bordes de la imagen, el de emborronarla.

Habrás notado que has tenido que esperar un rato (según la velocidad de tu máquina) durante un buen rato para ver el resultado. Este procedimiento es optimizable atendiendo a las especiales características de manejo de matrices de octave. En vez de hacer las operaciones elemento a elemento con sus superiores, inferiores, izquierdo y derecho, podemos hacerlas globalmente, simplemente trabajando con las matrices completas y operando con ellas con sus desplazadas una posición hacia arriba, abajo, izquierda y derecha. El ejemplo esta aquí y comprobarás que el resultado es notablemente más rápido.

```
octave:44> i=2:255;
octave:45> h=(g(i,i)+g(i-1,i)+g(i+1,i)+g(i,i-1)+g(i,i+1))/5;
octave:46> imshow(g,256)
octave:47> imshow(h,256)
```

Comprobemos ahora los otros dos filtros. El siguiente filtro pasa-alta se implementaría así. El resultado, al contrario que el anterior, acentúa los bordes en la imagen, resultando los pequeños detalles más destacados a simple vista.

```
octave:44> i=2:255;
octave:45> h=5*g(i,j)-g(i-1,j)-g(i+1,j)-g(i,j-1)-g(i,j+1);
octave:46> imshow(g,256)
octave:47> imshow(h,256)
```

(a) Imagen original (b) Filtro pasa-baja (c) Filtro pasa-alta

Figura 14.8: Cuando aplicamos un filtro pasa-baja se suaviza la imagen y si aplicamos un pasa-alta, sus detalles se acentúan.

Y por último, el detector de bordes, que lo que hace es presentar de color blanco las zonas que en la imagen original tienen cambios más rápidos, mientras que las zonas más homogéneas las presenta en negro. Con esto, conseguimos detectar bordes acusados en la imagen.

```
octave:44> i=2:255;
octave:45> h=5*g(i,j)-g(i-1,j)-g(i+1,j)-g(i,j-1)-g(i,j+1);
octave:46> imshow(g,256)
octave:47> imshow(h,256)
```

Figura 14.9: Imagen tras aplicar detector de bordes.

TEMA 15

GNUplot

Gnuplot es el programa encargado de hacer las gráficas 2D y 3D que se visualizaban en Octave. Gnuplot es un programa independiente de Octave, que usado por sí mismo te permite hacer representaciones de funciones continuas y de tablas de datos. Octave sólo usa un subconjunto de las funcionalidades de Gnuplot.

La primera característica de Gnuplot es que es muy similar a Octave en funcionamiento, es decir, que posee una interfaz de comandos muy poderosa que también puedes utilizar escribiendo scripts. Esta forma de trabajar tiene sus desventajas y sus ventajas. Las desventajas es que necesitas una curva de aprendizaje más lenta, donde tienes que haberte mirado por lo menos la descripción de uno de los comandos (`plot`) para poder empezar a hacer algo. Cuando estás tanteando datos mejor que uses otro programa que te permita hacer las cosas más interactivamente. Pero cuando ya tienes claro lo que tienes que hacer, por ejemplo, sobre una tabla de datos, y tienes 100 tablas de datos a las que hacer lo mismo, poder hacer un script puede ser de una gran ayuda.

La otra característica destacable de Gnuplot es la variedad de formatos de salida de que dispone, que se pueden seleccionar en el script. Te permite exportar a formatos vectoriales (xfig, TEX, postscript), formatos bitmap (png, pbm), o formatos de impresora (epson, hp, etc). Con esto puedes tener tu gráfica retocada por xfig en tu publicación en LaTeX, o bien puesta en tu página web (png) y o bien impresa directamente en una impresora.

Al ejecutar gnuplot en un shell entramos a su línea de comandos:

```
alberto@mencey:~$ gnuplot

G N U P L O T
Linux version 3.7
patchlevel 1
last modified Fri Oct 22 18:00:00 BST 1999

Copyright(C) 1986 - 1993, 1998, 1999
Thomas Williams, Colin Kelley and many others

Type ‘help‘ to access the on-line reference manual
The gnuplot FAQ is available from
<http://www.ucc.ie/gnuplot/gnuplot-faq.html>

Send comments and requests for help to <info-gnuplot@dartmouth.edu>
Send bugs, suggestions and mods to <submit@bugs.debian.org>


Terminal type set to 'x11'
gnuplot>
```

Como fuente de ayuda teclea `help` desde dentro del programa y después de una pantalla introductoria te saldrá un prompt sobre el que podrás escribir, o bien un nombre que elegirás de los topics que se te presentan, o bien un nombre de comando si quieres conocer su sintaxis.

Como has visto, el formato de salida es x11 (visualizar en las X). Para ver un listado de los diferentes tipos de salida disponibles usa `set terminal`.

15.1. Representación de expresiones analíticas

La parte más sencilla y práctica de Gnuplot es la presentación de funciones continuas, tanto en forma explícita `y=f(x)` o `z=f(x,y)`, como puede ser en forma paramétrica: curvas 2D `(x,y)=f(t)`, curvas 3D `(x,y,z)=f(u)`, superficies 3D `(x,y,z)=f(u,v)`.

Con `help functions` tenemos un listado de las funciones que admite. Una gran desventaja que tiene es que muestrea las funciones a intervalos regulares, por tanto, no hace ningún análisis de discontinuidades (lo que se nota en, por ejemplo, la función `floor`), aunque sí se puede configurar para que reduzca el intervalo. Si queremos imponer cual será el rango del eje X o el Y lo ponemos entre corchetes antes de la función. Algunos ejemplos:

```
gnuplot> plot x                        # identidad
gnuplot> plot abs(x)                   # valor absoluto
gnuplot> plot x**2                     # parábola
gnuplot> plot [-1:1] sqrt(1-x**2)      # semicircunferencia
gnuplot> plot [] [-0.1:1.1] exp(-x**2) # gaussiana
gnuplot> plot [-1:4] gamma(x)          # función gamma
gnuplot> plot floor(x)                 # función redondeo hacia abajo
gnuplot> plot x-floor(x)               # diente de sierra
gnuplot> splot x**2+y**2               # plot en 3D
gnuplot> splot sqrt(1-x**2+y**2)
gnuplot> set isosamples 20,20          # cambia la resolución
gnuplot> replot
gnuplot> set isosamples 50,50          # cambia la resolución
gnuplot> set contour        # activa líneas de nivel
gnuplot> replot
gnuplot> set parametric                # modo paramétrico

dummy variable is t for curves, u/v for surfaces
gnuplot> set samples 500                   # mejor resolución (+lento)
gnuplot> plot sin(7*t),cos(5*t)            # lissajous en 2D
gnuplot> splot sin(5*u),sin(6*u),sin(7*u)  # lissajous en 3D
gnuplot> set samples 100                   # menor resolución (+rápido)
gnuplot> splot cos(u)*cos(v),cos(u)*sin(v),sin(u)  # esfera en 3D
```

15.2. Representación de archivos de datos

Gnuplot también tiene un modo para trabajar con archivos de datos con múltiples columnas. Cuando los archivos de datos tienen 1 ó 2 columnas se presentan directamente. Si un archivo tiene más de 2 columnas se pueden presentar columnas arbitrariamente, hacer operaciones matemáticas sencillas entre columnas. Veamos esto en un ejemplo real (bastante prolijo) donde un servidor genera una línea de log de load, logins y carga de cpu, a cada hora y queremos obtener gráficas que muestren la evolución en el tiempo

```
# Ejemplo para la monitorización de carga de un servidor en el tiempo

set title "Convex     November 1-7 1989    Circadian"
set key left box
set xrange[-1:24]
plot 'gnuplot.dat' using 2:4 title "Logged in" with impulses,\
     'gnuplot.dat' using 2:4 title "Logged in" with points
pause -1 "Hit return to continue"

set xrange [1:8]
```

```
#set xdtic
set title "Convex     November 1-7 1989"
set key below
set label "(Weekend)" at 5,25 center
plot 'gnuplot.dat' using 3:4 title "Logged in" with impulses,\
     'gnuplot.dat' using 3:5 t "Load average" with points,\
     'gnuplot.dat' using 3:6 t "%CPU used" with lines
set nolabel
pause -1 "Hit return to continue"
reset
```

Como último ejemplo, vamos a probar un script donde se hacen ajustes por el método de mínimos cuadrados con Gnuplot. En el ejemplo se realizan ajustes a una recta variando los pesos, pero el método de ajuste que utiliza Gnuplot permite poner cualquier función de ajuste, simplemente definiendo las variables y constantes y dando unos valores iniciales a las constantes.

```
# ajustes por mínimos cuadrados en Gnuplot

y(x) = a*x + b   # función a la que se ajustará
a = 0.0          # valores iniciales
b = 0.0          # de los parámetros

fit y(x) 'gnuplot-fit.dat' via a, b
set title 'Ajuste sin pesar'
plot 'gnuplot-fit.dat', y(x)
pause -1 "Pulsa enter para continuar"

fit y(x) 'gnuplot-fit.dat' using 1:2:3 via a, b
set title 'Ajuste con mayor peso en bajas temperaturas'
plot 'gnuplot-fit.dat', y(x)
pause -1 "Pulsa enter para continuar"

fit y(x) 'gnuplot-fit.dat' using 1:2:4 via a, b
set title 'Ajuste con mayor peso a altas temperaturas'
plot 'gnuplot-fit.dat', y(x)
pause -1 "Pulsa enter para continuar"

fit y(x) 'gnuplot-fit.dat' using 1:2:5 via a, b
set title 'Ajuste con peso correspondiente a error experimental'
plot 'gnuplot-fit.dat' using 1:2:5 with errorbars, y(x)
pause -1 "Pulsa enter para continuar"
```

TEMA 16

GNU R

16.1. Introducción

R es a la vez un entorno y un lenguaje de programación para realizar cálculos y gráficos estadísticos. Es un proyecto GNU similiar al sistema S desarrollado en Bell Laboratories (formalmente AT&T, ahora Lucent Technologies) por John Chambers y sus colegas. R puede considerarse como una implementación diferente de S. Hay diferencias importantes entre ambos, pero mucho código escrito para el sistema S puede ejecutarse en R sin modificarlo.

R proporciona una amplia variedad de técnicas gráficas y estadísticas (regresiones lineales y no lineales, tests estadísticos clásicos, análisis de series temporales, clasifiaciones, clustering, etc.) y además es muy extensible. El lenguaje S suele ser utilizado para la investigación en metodología estadística, y R proporciona una alternativa de código abierto para esta actividad.

Uno de los puntos fuertes de R es la facilidad con la que se pueden producir gráficas de buen diseño y calidad de imprenta, incluyendo símbolos y fórmulas matemáticas donde sean necesarias. Aunque R pone un gran cuidado en las opciones por defecto para el diseño de las gráficas, el usuario puede tener control total sobre éstas.

R es Software Libre, disponible bajo los términos de la GNU General Public License de la Free Software Foundation, en forma de código fuente. Se puede compilar y ejecutar en una amplia variedad de plataformas UNIX (incluyendo Linux y FreeBSD), MacOS y Windows 9x/NT/2000.

El código fuente de R se puede descargar del sitio web del proyecto R (`http://www.r-project.org`), así como su documentación. Además de la documentación oficial (en inglés) existen otros documentos *contribuidos* entre los que se encuentra la traducción al español de "An Introduction to R" y "Gráficos Estadísticos con R". Estos y más manuales se encuentran en el CRAN (Comprehensive R Archive Network), concretamente en `http://cran.r-project.org/other-docs.html`

16.2. El entorno R

R es un conjunto integrado de utilidades para manipulación, cálculo y representación de datos. Decimos que R es un *entorno* porque es un sistema diseñado para ser completamente coherente. El entorno de R proporciona facilidades para manipulación y almacenamiento de datos, operaciones con variables indexadas (como vectores y matrices), análisis y representación de datos, un lenguaje de programación bien desarrollado (con todo lo que cabe esperar de un lenguaje de programación) y una amplia colección integrada y coherente de utilidades para análisis de datos.

Aunque mucha gente utiliza R como un sistema estadístico, sus autores prefieren considerarlo como "un entorno en el que se han implementado muchas técnicas estadísticas, clásicas y modernas". La mayoría de la estadística clásica y muchas de los últimos métodos están disponibles en R, aunque posiblemente tendrás que buscar un rato para encontrarlas.

La forma de trabajar con R es distinta a la de otros programas como SPSS. En R, un análisis estadístico se realiza en una serie de pasos, con unos resultados intermedios que se van almacenando en objetos, para ser observados o analizados posteriormente, produciendo unas salidas mínimas. En SPSS se obtendría de modo inmediato una salida copiosa para cualquier análisis. Esto puede parecer a primera vista una terrible incomodidad, pero si tuviéramos que trabajar en una máquina poco potente rápidamente nos daríamos cuenta de que puede resultar muy ventajosa la sencillez del entorno de R (un entorno de comandos) y la posibilidad de ver en cada momento exactamente lo que se necesita, sin excesos que desperdicien recursos del sistema.

Aún así, la mejor manera de trabajar con R es en un entorno gráfico, con un sistema de ventanas como X-Window, de forma que puedas ver las gráficas en el momento de generarlas.

Veamos cómo se trabaja con R usándolo. En primer lugar conviene crear un directorio y entrar en él antes de comenzar una sesión con R, que en éste almacena siempre en el directorio donde se ejecutan unos ficheros donde almacena los objetos, datos, funciones y comandos ejecutados. Esto puede sernos muy útil en trabajos largos, podemos interrumpir la sesión con R en cualquier momento y recuperarla luego donde mismo la dejamos.

Hemos considerado a lo largo de este curso que el símbolo del sistema Linux/UNIX es `$`. Vamos a considerar ahora que el símbolo del prompt R es >. Abre un emulador de terminal dentro del entorno gráfico y ejecuta los siguientes comandos:

```
$ mkdir sesion_R
$ cd sesion_R
$ R

R : Copyright 2002, The R Development Core Team
Version 1.5.1  (2002-06-17)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.

>
```

El prompt de R indica el entorno que está esperando tus órdenes para ejecutarlas. Para salir del entorno de R puedes utilizar la función `q()` o pulsar `C-d`, entonces R te preguntará si deseas guardar el *espacio de trabajo*, que es toda la información sobre la sesión que quieres cerrar:

```
> q()
Save workspace image? [y/n/c]:
```

Para salir guardando la sesión pulsa `y`, para salir sin guardarla pulsa `n` y para para cancelar la salida pulsa `c`. Si guardas la sesión al salir quedará almacenada en el directorio en el que ejecutaste `R`, y será automáticamente recuperada la próxima vez que ejecutes `R` dentro del directorio en cuestión.

R dispone de un sistema de ayuda similar a las páginas de manual de Linux/UNIX. Cuando necesites información sobre una función, por ejemplo `solve`, ejecuta la función `help()` pasándole como parámetro el nombre de la función que quieras consultar. También existe una forma más corta: `?función`:

```
> help (solve)
> ?solve
```

Normalmente puedes ver la documentación también en el navegador web. Si ejecutas `help.start()` debería abrirse una ventana de navegador con la documentación en formato HTML por la cual puedes navegar para buscar lo que necesites.

Otra forma muy interesante de buscar ayuda dentro del entorno de R es pasarle una palabra a la función `help.search()`. Por ejemplo, para buscar una función que resuelva sistemas de ecuaciones puedes empezar por buscar la palabra `solve`.

```
> help.search ("solve")

Help files with alias or title matching 'solve',
type 'help(FOO, package = PKG)' to inspect entry 'FOO(PKG) TITLE':

backsolve(base)         Solve an Upper or Lower Triangular System
qr(base)                The QR Decomposition of a Matrix
solve(base)             Solve a System of Equations
```

De un primer golpe ya sabes que el paquete `base` de R tiene funciones para resolver sistemas triangulares (superiores o inferiores), descomponer matrices en la forma QR y resolver sistemas de ecuaciones (cualesquiera).

R proporciona algunas demostraciones sobre sus capacidades. Para ir abriendo el apetito ejecuta la funciones que muestran las demostraciones con gráficos e imágenes:

```
> demo (graphics)
> demo (images)
```

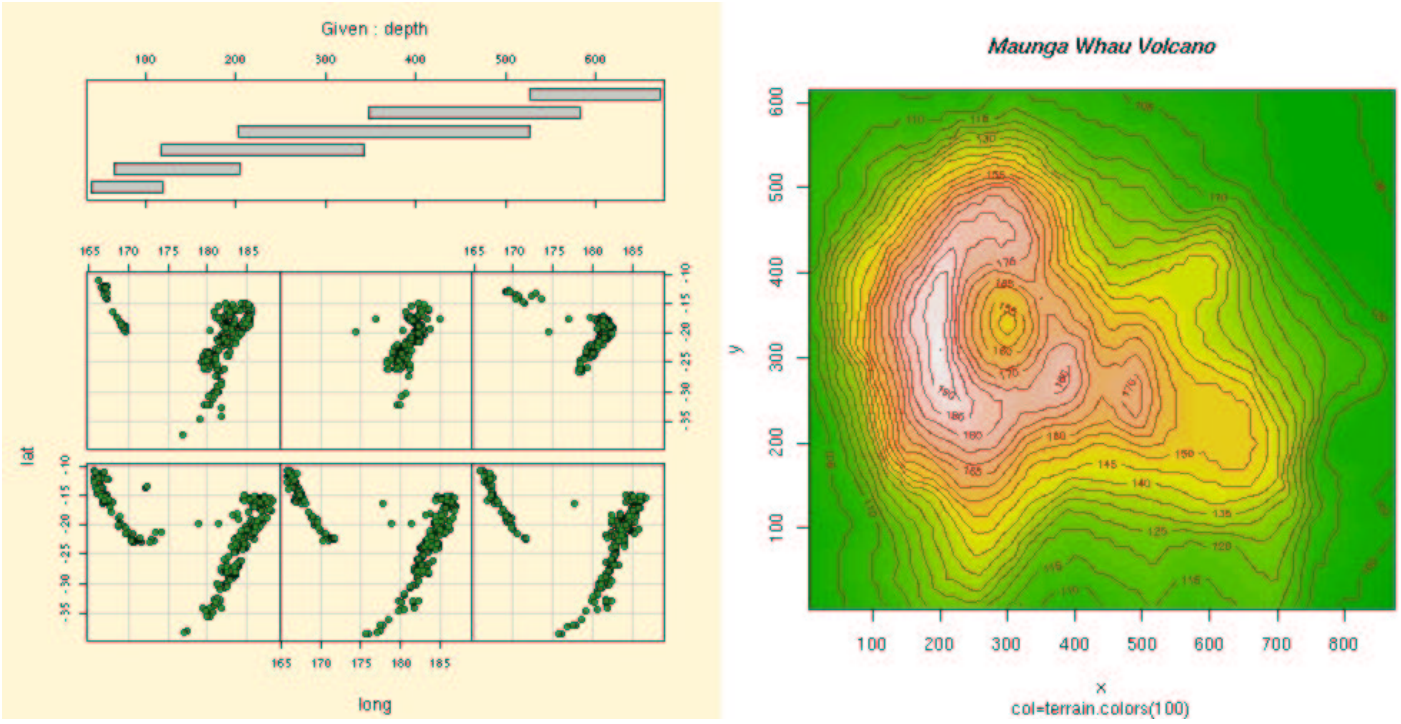


Figura 16.1: Demostraciones sobre gráficos e imágenes

16.3. El lenguaje R

R es también lenguaje de programación en toda regla, con el que puedes escribir programas que procesen ficheros de datos y generen análisis y representaciones de los datos.

Al igual que la mayoría de lenguajes basados en UNIX, el lenguaje de R distingue entre mayúsculas y minúsculas. Esto es, `esto` y `Esto` son símbolos distintos y pueden referirse a variables distintas. En R los nombres de variables pueden tener letras y números (como en casi cualquier lenguaje), aunque **no** pueden tener el subrayado (_), pero en su lugar sí pueden tener el punto. De hecho se suele utilizar el punto para separar palabras en los nombres de las variables en R, por ejemplo `hoja.de.datos`.

Las órdenes (o comandos) elementales son expresiones o asignaciones. Si ejecutas una expresión como comando ésta se evalúa y se imprime, pero su valor se pierde. En una asignación la expresión se evalúa y su valor se almacena en la varbiable, pero no se imprime.

Los comandos se separan con el caracter de punto y coma (;) o simplemente con una salto de línea. Puedes agrupar una serie de comandos encerrándolos entre llaves, de esta forma puedes definir funciones como veremos más adelante. También puedes poner comantarios, casi donde quieras[1], poniendo una almohadilla (#) y todo lo que la siga hasta el final de línea será obviado por el intérprete de R.

[1] No puedes meter un comentario dentro de una cadena de caracteres (formaría parte de la cadena), ni en medio de la lista de argumentos en la definición de una función

Si dejas un comando a medias R se dará cuenta, y en lugar de mostrarte el prompt > te mostará + para avisarte de que está esperando por el final del comando.

Otra característica interesante de R (al menos en plataformas Linux/UNIX) es la posibilidad de editar la línea de comandos. Esto incluye el típico historial de comandos, que te permite repetir los comandos sin tener que volver a teclearlos. Tan solo utiliza los cursores arriba y abajo para recuperar los comandos que hayas tecleado. Además este historial de comando queda almacenado en la sesión cuando sales del entorno R, concretamente en un fichero `.Rhistory` en el directorio donde ejecutaste en entorno R.

Pero si necesitas usar una serie un poco larga de comandos seguramente te canses de tener que darle a los cursores. Para estos casos lo que necesitas es un script en el que escribes los comandos a modo de programa. Luego para ejecutar los comandos escritos en él utilizas la función `source ()` del entorno pasándole el nombre del fichero donde escribiste los comandos:

```
> source ("script.R")
```

Del mismo modo que puedes tomar la entrada de un fichero también puedes redireccionar la salida hacia otro fichero, con la función `sink()`. Para almacenar la salida en el fichero `salida.txt` ejecuta el comando:

```
> sink ("salida.txt")
```

Para devolver la salida al intérprete utiliza la misma función `sink ()` pero sin pasarle ningún parámetro. En el fichero `iodemo.R` tienes un ejemplo de esto.

Fichero `iodemo.R`

```
sink ("salida.txt")
hoja.datos = read.table ("muestra.dat")
attach (hoja.datos)
print (summary (Edad))
sink ()
print (summary (Ingresos))
```

Ejemplo 16.1: Este script lee una tabla que está almacenada en un fichero de texto llano (`muestra.dat`, que usaremos para los ejemplos) y extrae dos variables de ella. Para ambas variables muestra un breve resumen estadístico, pero guarda uno en el fichero `salida.txt` y el otro lo muestra en el entorno R.

Entra en el directorio donde tengas el fichero y ejecuta en el entorno R

```
> source ("iodemo.R")
```

16.4. Vectores

R realiza las operaciones sobre las llamadas *estructuras de datos*, de las cuales la más simple es el vector numérico. Para crear un vector con los 5 primeros números enteros positivos utilizamos la función `c()` que combina los objetos recibidos para formas un vector:

```
> x <- c (1, 2, 3, 4, 5)
> x
[1] 1 2 3 4 5
```

En efecto, el operador de asignación no se parece para nada a un signo de igualdad. De hecho es una flecha, que está indicando que el valor de lo que hay a su derecha debe asignarse a lo que hay a su izquierda. Si le das la vuelta a la flecha funciona al revés:

```
> c (6, 7, 8, 9, 10) -> y
> y
[1]  6  7  8  9 10
```

Cuando ejecutas el nombre de un objeto como comando, R interpreta que deseas ver su contenido y entonces ejecuta la función `print()` sobre el objeto en cuestión. Esto funciona sólo cuando estás dentro del entorno R y tecleas el nombre del objeto. Cuando quieras imprimir el objeto desde un fichero de comandos (como hacemos en `iodemo.R`) has de utilizar la función `print()`.

Dado que la función `c()` combina objetos para formar vectores, también puede combinar varios vectores para formar un nuevo vector que es el resultado de concatenar los anteriores:

```
> z <- c (x, y)
> z
 [1]  1  2  3  4  5  6  7  8  9 10
```

Las operaciones usuales entre vectores se definen "elemento a elemento":

```
> x + 2
[1] 3 4 5 6 7
> 2 * x
[1]  2  4  6  8 10
> x + y
[1]  7  9 11 13 15
> x - y
[1] -5 -5 -5 -5 -5
> x * y
[1]  6 14 24 36 50
> x / y
[1] 0.1666667 0.2857143 0.3750000 0.4444444 0.5000000
> y / x
[1] 6.000000 3.500000 2.666667 2.250000 2.000000
> x ^ y
[1]       1     128    6561  262144 9765625
> y ^ x
[1]      6     49    512   6561 100000
```

Pero entonces ¿cómo se multiplican escalarmente dos vectores? Para esto hay una función llamada `crossprod`, que permite multiplicar dos vectores (o matrices). El operador `%*%` es una forma abreviada de este producto. Formalmente `crossprod(x,y)` es equivalente a `t(x)%*% y`, pero más rápido. La función `t()` es la trasposición de matrices:

```
> x
[1] 1 2 3 4
> y
[1] 4 5 6 7
> crossprod(x,y)
     [,1]
[1,]   60
> x %*% y
     [,1]
[1,]   60
```

Además de para guardar datos, los vectores suelen usarse para definir series o secuencias. Para definir una secuencia de números enteros consecutivos basta con poner el primer y el último separados por un caracter de dos puntos:

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
```

Pero para generar secuencias la mejor manera es utilizar la función `seq()`. Si le pasas tres valores numéricos le estarás especificando el primer y el último elemento de la secuencia, y la diferencia que debe haber entre cada dos elementos consecutivos:

```
> y <- seq (-1, 1, .2)
> y
 [1] -1.0 -0.8 -0.6 -0.4 -0.2  0.0  0.2  0.4  0.6  0.8  1.0
```

Para replicar un objeto varias veces, por ejemplo para generar una secuencia periódica, utiliza la función `rep()` pasándole el objeto y el número de veces que quieres replicarlo:

```
> z <- rep (seq (-1, 1, 1), 5)
> z
 [1] -1  0  1 -1  0  1 -1  0  1 -1  0  1 -1  0  1
```

Además de vectores numéricos, R permite operar con vectores *lógicos*. Los valores válidos para los elementos de los vectores lógicos son los de la *lógica triestada*: `TRUE`, `FALSE` y `NA` ("Not Available", no disponible). Los operadores lógicos para manejar estos valores son `&` para "and", `|` para "or" y `!` para "not".

Una forma de generar un vector de valores lógicos es efectuar una comparación entre un vector numérico y un valor numérico:

```
> z <- rep (seq (-1, 1, 1), 3)
> z
[1] -1  0  1 -1  0  1 -1  0  1
> z > 0
[1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE
```

Al contrario que en la mayoría de lenguajes de programación, en R los vectores se indexan comenzando por 1. Para acceder a un elemento de un vector utiliza la notación usual de los corchetes. También puedes acceder a un subconjunto del vector especificando como índice un vector con las posiciones de los elemenos:

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x[2:5]
[1] 2 3 4 5
> x[8:2]
[1] 8 7 6 5 4 3 2
```

También puedes acceder a un subconjunto de un vector utilizando un vector lógico:

```
> x[x > 4]
[1]  5  6  7  8  9 10
```

Incluso puedes indexar un vector no palabras, poniendo nombres a las posiciones dentro del vector mediante la función `names()`. Luego puedes acceder a los elementos mediante los nombres de las posiciones:

```
fruta = c (5, 10, 1, 20)
> fruta <- c (5, 10, 1, 20)
> names (fruta) <- c ("naranja", "plátano", "manzana", "ñame")
> fruta[c ("manzana", "naranja")]
manzana naranja
      1       5
> fruta
naranja plátano manzana    ñame
      5      10       1      20
```

16.5. Arrays y matrices

Un *array*[2] es una variable indexada multidimensional. Un vector es un array unidimensional y una matriz es un array bidimensional. R proporciona un buen manejo de arrays, sobretodo para matrices.

La dimensión de un array es un vector cuya longitud es la dimesión del array, y cada uno de sus elementos es la "longitud" de cada dimensión en el array. Por ejemplo, un array con dimesión $(3, 5, 100)$ sería como un cubo de 1500 elementos con 3 celdas de largo, 5 de ancho y 100 de alto.

En la mayoría de lenguajes los elementos de un array se almacenan "por filas", lo que significa que el índice que avanza más rápido es el último. Sin embargo, en R la ordenación se hace al estilo de FORTRAN, ordenando los elementos "por columnas", lo que significa que el índice que avanza más rápido es el primero.

Es importante tener esto en cuenta porque a la hora de crear una matriz hay que proporcionar los elementos agrupados por columnas, no por filas. Por ejemplo:

```
> a <- array (c (1:3,-3:-1), dim = c (3,2))
> a
     [,1] [,2]
[1,]    1   -3
[2,]    2   -2
[3,]    3   -1
```

Para acceder a los elementos de una matriz (o array) ponemos los subíndices entre corchetes y separados por comas. Si omitimos el subíndice de las filas (el primero) obtenemos la columna señalada por el segundo subíndice, y análogamente obtenemos las filas omitiendo el segundo sunídice.

```
> a[1,2]
[1] -3
> a[,2]
[1] -3 -2 -1
> a[1,]
[1]  1 -3
```

De la misma forma que podemos extraer elementos, filas o columnas de una matriz también podemos asignarles valores:

```
> a[,2] <- 3:5
> a[,2]
[1] 3 4 5
> a
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    3    5
```

Para multiplicar matrices (o arrays en general) utiliza el operador **%*%** que vimos antes:

```
> a
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    3    5
> b = array (1:2, 2:1, dim = c (2,2))
> b
     [,1] [,2]
[1,]    1    1
[2,]    2    2
```

[2] a veces traducido por "arreglo"

```
> a %*% b
     [,1] [,2]
[1,]    7    7
[2,]   10   10
[3,]   13   13
```

Vimos antes que si multiplicamos dos vectores `x` y `y` (de igual longitud) utilizando el operador `%*%` el resultado es el producto escalar de los vectores. En realidad la operación `x %*% y` es ambigua, porque podría significar $x'x$ o xx' (considerando x un vector columna). En estos casos de ambigûedad se considera implícitamente que la interpretación deseada es aquella de la que resulte la matriz más paqueña, por lo que se obtiene el producto escalar $x'x$.

Para calcular la matriz xx' puedes utilizar las funciones `cbind()` y `rbind()`, ya que éstas siempre devuelven matrices. Las funciones `cbind` y `rbind` toman una serie de vectores o números y los agrupan por columnas o filas en una matriz.

```
> x
[1] 1 2 3 4
> y
[1] 4 5 6 7
> x %*% y
     [,1]
[1,]   60
> cbind(x,y)
     x y
[1,] 1 4
[2,] 2 5
[3,] 3 6
[4,] 4 7
> rbind(x,y)
  [,1] [,2] [,3] [,4]
x    1    2    3    4
y    4    5    6    7
> cbind(x) %*% rbind(y)

      [,1] [,2] [,3] [,4]
  [1,]    4    5    6    7
  [2,]    8   10   12   14
  [3,]   12   15   18   21
  [4,]   16   20   24   28
```

16.6. Factores: clasificación de datos

Un *factor* es un vector que utilizamos para especificar una clasificación discreta (agrupamiento) de los componentes de otros vectores del mismo tamaño. Para entender lo que son los factores nada mejor que un ejemplo claro y sencillo. En el entorno R y ejecuta lo siguiente:

```
> edad <- c (13, 16, 15, 17, 17, 18, 16, 16, 15, 16)
> sexo <- c ("hombre", "hombre", "mujer", "hombre", "mujer", "mujer",
+            "hombre", "mujer", "hombre", "mujer")
> edad
 [1] 13 16 15 17 17 18 16 16 15 16
> sexo
[1] "hombre" "hombre" "mujer"  "hombre" "mujer"  "mujer"  "hombre"
[9] "mujer" "hombre" "mujer"
```

Ahora tienes una variable `edad` que contiene las edades de 10 personas, y una variable `sexo` que define el sexo de cada una de las 10 personas anteriores (en el mismo orden) y que toma únicamente los valores `"hombre"` y `"mujer"`. Vamos a

calcular la media y la varianza muestrales de la edad de estas personas clasificándolas según el sexo, i.e. la edad media de los hombres por un lado y la de las mujeres por otro.

Necesitamos separar (agrupar) los elementos del vector `edad` según los valores que toma el vector `sexo`. Para esto utilizamos un factor. El factor en realidad es como el vector original, pero tiene un atributo añadido llamado `levels` (niveles) que son los distintos valores que toman los elementos del vector original. En el caso del vector `sexo` los niveles son `"hombre"` y `"mujer"`. La función `levels` devuelve la lista de niveles que tenga el factor que le pases como parámetro.

```
> sexo.factor = factor (sexo)
> sexo.factor
[1] hombre hombre mujer  hombre mujer  mujer  hombre mujer  hombre
Levels:  hombre mujer
> levels (sexo.factor)
[1] "hombre" "mujer"
```

Ahora queremos aplicar unas funciones, en este caso `mean()` y `var()`, a un vector de datos de manera que los datos sean separados según un factor. La función apropiada para esta tarea es `tapply`, y se usa del siguiente modo:

```
> tapply (edad, sexo.factor, mean)
hombre  mujer
  15.4   16.4
> tapply (edad, sexo.factor, var)
hombre  mujer
   2.3    1.3
```

Así obtenemos que dentro de esta muestra de 10 personas, la edad media de los hombres es 15,4 y la de las mujeres 16,4. Volveremos sobre los factores más adelante, así que asegúrate de entender al menos este uso de los mismos. Para mayores detalles consulta la ayuda de R.

16.7. Listas

En un array todos los elementos deben ser del mismo tipo, i.e. no puede ser que un array contenga a la vez números y palabras. Sin embargo la mayoría de muestras contienen datos de diferentes tipos: números, palabras, valores lógicos, intervalos, etc.

Una *lista* en R es una colección ordenada de objetos de cualquier tipo. Es como un vector, pero con la diferencia de que sus elementos pueden ser de distintos tipos.

Los elementos de una lista están siempre numerados por un subíndice y puedes referirte a ellos como con un vector, pero usando corquetes dobles. Así si `L` es una lista `L[[1]]` es su primer elemento. Pero también, al igual que en los vectores, puedes indexar los elementos de una lista dándoles nombres. En caso de nombrar los elementos de una lista hay una forma más cómoda de referirse a ellos: en lugar de `L[["nombre"]]` puedes usar `L$nombre`.

```
> L = list (nombre = "Pepe", esposa = "Pepa", hijos = 3,
+     edades.hijos = c (34,6,12))
> L
$nombre
[1] "Pepe"

$esposa
[1] "Pepa"

$hijos
[1] 3

$edades.hijos
[1] 34  6 12
```

```
> L[["nombre"]]
[1] "Pepe"
> L$nombre
[1] "Pepe"
```

Para concatenar listas recuerda que la función `c()` sirve para ello.

16.8. Hojas de datos

Una "hoja de datos" (en inglés "data frame") es básicamente una lista de vectores, con algunas restricciones. Los vectores de palabras son convertidos en factores.

Piensa en una hoja de datos como su nombre sugiere: una hoja o tabla en la que tienes varios datos sobre varios individuos. Las columnas de la hoja de datos son los vectores que la format, y cada fila es una lista.

En el fichero `muestra.dat` tienes una hoja de datos preparada para ser leida con la función `read.table()`. La primera fila son los nombres de los vectores columna, y luego cada línea del fichero introduce una lista de valores, un valor para cada vector. Para entender esto con claridad lee la tabla del fichero `muestra.dat` y mantenla en una variable:

```
> hoja.de.datos = read.table ("muestra.dat")
> names (hoja.de.datos)
[1] "Sexo"     "Edad"     "Habitat"  "Ingresos" "Lectura"  "TV"
```

Esta tabla contiene los datos (ficticios) de 50 jóvenes: su sexo, su edad, su hábidat, sus ingresos familiares (en miles de pesetas mensuales), el número de libros leidos anualmente y sus horas de televisión diarias.

Normalmente para acceder al vector `Edad` de la hoja de datos utilizaríamos la expresión `hoja.de.datos$Edad`, pero hay una forma más cómoda. Consiste en "conectar" la hoja de datos para que puedas referirte a sus vectores directamente por su nombre:

```
> attach (hoja.de.datos)
> summary (Edad)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  12.00   15.00   16.00   15.64   17.00   20.00
```

Ten en cuenta que este atajo es sólo para *leer* los datos de la hoja, no para modificarlos. Si quieres modificar un vector de la hoja de datos tienes que usar la notación normal:

```
> hoja.de.datos$Edad <- Edad + 10
```

De hecho, esto modifica el vector en la hoja de datos, pero no verás los cambios en los atajos hasta que desconectes la hoja de datos y la vuelvas a conectar. Para desconectar una hoja de datos utiliza la función `detach()`.

Cuando quieras almacenar una hoja de datos en un fichero utiliza la función `write.table()`. Su uso básico es darle la hoja de datos y el nombre del fichero, pero admite varias opciones para personalizar la forma en la que escribirá los datos en el fichero. Para ver estos detalles consulta la ayuda sobre la función ejecutando `?write.table`.

16.8.1. Valores perdidos

En ocasiones te encontrarás con hojas de datos en las que alguna celda no tiene el dato. Esto puede suceder porque no haya sido posible averiguar el dato, o porque éste no tenga sentido. En estos casos se utiliza para esa celda el valor `NA`, que significa que el valor "no está disponible" (NA es abreviatura de "Not Available").

En otros casos sucede que tras efectuar una serie de operaciones sobre un conjunto de datos algunos de los resultados no tengan sentido matemático, como por ejemplo una división por cero (recuerda que la precisión de los procesadores es limitada). En caso de hacer una operación así R no se quejará en absoluto, sino que devolverá el valor `NaN` que significa que eso "no es un número" (NaN es abreviatura de "Not a Number").

16.9. Funciones

Como todo buen lenguaje de programación, R te permite definir funciones para tu uso propio. Para definir una función asignas al objeto (que será tu función) el valor devuelto por la función `function`. Esta función particular recibe los mismos parámetros que recibirá tu función, y a continuación una *agrupación* de comandos, encerrados entre llaves y separados con ; o saltos de línea. El valor devuelto por tu función será el valor de la última expresión de la agrupación.

Por ejemplo si quieres una función que calcule la curtosis de una muestra:

```
> curtosis <- function (x) {
+ n <- length (x)
+ c <- ( (n * (n - 1) * sum((x - mean(x))^4) ) /
+      ( (n - 1) * (n - 2) * (n - 3) * (var(x))^4 ) ) -
+      ( (3 * (n - 1)^2) / ( (n - 2) * (n - 3) ) )
+ c
+ }
> hoja.de.datos = read.table ("muestra.dat")
> attach (hoja.de.datos)
> curtosis (Edad)
[1] -2.921993
> curtosis (Lectura)
[1] -3.191575
> curtosis (TV)
[1] -3.192770
```

Normalmente las funciones no quedan bien escritas a la primera, por lo que tendrás que corregirlas una y otra vez. O tal vez quieras definir más funciones, modificar las que ya tengas escritas, etc. Todo esto puedes hacerlo más cómodamente si escribes las funciones en un fichero y lo importas con la función `source()` que vimos antes.

Así por ejemplo tienes escrita la función `curtosis` en el fichero `curtosis.R`. Si quieres modificarla y volver a usarla después de modificada no hay problema. Edita el fichero `curtosis.R` para modificar la función y luego vuelve a importarlo con la función `curtosis`. Recuerda que la función `source()` ejecuta los comandos que encuentra en el fichero que le digas, por lo que las funciones que tengas escritas en el fichero serán redefinidas y estarán lista para usar al instante.

Fichero `curtosis.R`

```
curtosis <- function (x) {
n <- length (x)
c <- ( (n * (n - 1) * sum((x - mean(x))^4) ) /
     ( (n - 1) * (n - 2) * (n - 3) * (var(x))^4 ) ) -
     ( (3 * (n - 1)^2) / ( (n - 2) * (n - 3) ) )
c
}
```

Ejemplo 16.2: Este fichero contiene la definición de una función, y cada vez que lo leas con la función `source()` será ejecutado. Por lo tanto las funciones que contiene son redefinidas, lo que te permite modificar las funciones y hacer efectivos los cambios sin tener que salir del entorno.

16.9.1. Control de flujo

R proporciona la sintaxis necesaria para construir funciones que mantengan en control del flujo del programa. Nos referimos a los condicionales y los bucles:

Ejecución condicional

La forma de contruir una sentencia condicional en R es `if (expresion_logica) sentencia_1 else sentencia_2`. Si la `expresion_logica` es cierta (su valor es `TRUE`) se ejecutará la `sentencia_1`, en otro caso se ejecutará la `sentencia_2`. `sentencia_1` y `sentencia_2` pueden ser también agrupaciones de comandos y/o expresiones.

```
> x = 1
> y = 2
> if (x > y) x else y
[1] 2
```

En las expresiones booleanas puedes emplear los operadores | (OR) & (AND) y ! (NOT) normales, o si prefieres ahorrar un poco de tiempo los operadores "cortocicuitados" || (OR) y &&. Estos operadores lógicos tienen la ventaja de que sólo evalúan la segunda expresión cuando es necesario.

Para las comparaciones numéricas se utilizan los mismos comparadores binarios que en el lenguaje C: <, >, <=, >=, != y ==. Ten cuidado de no utilizar el operador = para comparaciones.

Existe además una versión *vectorizada* de la construción condicional. Se trata de la función `ifelse ()`, que recibe tres vectores `condicion, a, b` y devuelve un vector del tamaño del más largo. En este vector el elemnto i-ésimo es `a[i]` si `condicion[i]` es cierto (su valor es `TRUE`) o bien `b[i]`) en caso contrario.

```
> condicion = c (TRUE, FALSE)
> a = c (1, 2)
> b = c (3, 4)
> ifelse (condicion, a, b)
[1] 1 4
```

Ejecución repetitiva

Para la ejecución repetitiva de comandos dispones de la construcción `for (variable in valores) comandos`, donde `variable` es una variable vacía que irá cambiando de valor en cada iteración tomando consecutivamente los valores del vector `valores`. Para cada uno de estos valores se ejecutará la secuencia `comandos`.

```
> for (i in 1:3) print (c (i^2, i^3, i^4, i^5))
[1] 1   1   1   1
[1] 4   8  16  32
[1] 9  27  81 243
```

Para ejecutar una secuencia *mientras* se cumpla una condición (podría no ejecutarse la primera vez) utiliza la construcción `while (condicion) comandos`. Así mientras el valor de la expresión `condicion` sea `TRUE` se seguirá ejecutando la secuencia `comandos`.

```
x = rnorm (1)
> while (x > -2) {
+   print (x)
+   x = rnorm (1)
+ }
[1] -0.2199842
[1] -0.1615499
[1] 2.580791
[1] -0.1202099
[1] -0.794566
[1] -1.413912
[1] 0.4829018
[1] 0.2780835
[1] -0.5475556
[1] 2.974901
[1] -0.1805834
```

La construcción `repeat comandos` se mantiene ejecutando la secuencia `comandos` hasta que se ejecute (dentro de la secuencia) el comando `break`. Esta es la única forma de interrumpir la ejecución de un `repeat`. En iteración del bluce la intrucción `next` hace que la se termine la iteración y se comienze con una nueva (no sale del bucle).

16.10. Distribuciones de probabilidad

R tiene un conjunto de funciones para evaluar las funciones de distribución, densidad y probabilidad de las distribudiones que están tabuladas. Para cada distribución la función que lleva su nombre devuelve el valor de su función de distribución, i.e. $P(X \leq x)$[3].

Las distribuciones para las que R proporciona estas funciones son, con sus nombres en R: beta (`beta`), binomial (`binom`), Cauchy (`cauchy`), χ^2 (`chisq`), exponencial (`exp`), F de Fisher (`f`), gamma (`gamma`), geométrica (`geom`), hipergeométrica (`hyper`), log-normal (`lnorm`), logística (`logis`), binomial negativa (`nbinom`), normal (`norm`), Poisson (`pois`), t de Student (`t`), uniforme (`unif`), Weibull (`weibull`) y Wilcoxon (`wilcox`).

Cada una de estas distribuciones tabuladas proporciona cuatro funciones, cuyos nombres se obtienen precediendo las letras `d`, `p`, `q` o `r` al nombre en R de la distriburión, y que son respectivamente las funciones de densidad, distribución, quantiles y simulación (generadoras aleatorias).

Cada una de estas funciones puede requierer argumentos obligatorios para especificar los parámetros de la distribución o bien permitir argumentos opcionales para modificar los parámetros por defecto. Por ejemplo, `rnorm(10)` genera un vector de 10 números aleatorios que siguen una distribución normal con media 0 y varianza 1, pero estos parámetros pueden modificarse con las opciones oportunas:

```
> rnorm(5)
[1]  0.5849966  2.6217292 -0.9060517  1.1373629  1.4008370
> rnorm(5, mean = 100, sd = 10)
[1] 109.8315 106.1667  94.7198 116.6163 109.8492
```

Por otra parte, para evaluar la función de densidad (o cualquier otra) de la distribución χ^2_n es necesario saber el número de *grados de libertad* n (en inglés *degrees of freedom*). Este parámetro es obligatorio:

```
> rchisq (5)
Error in rchisq(5) : Argument "df" is missing, with no default
> rchisq (5, 3)
[1] 3.019664 3.335430 6.537741 4.534492 1.843734
> rchisq (5, 30)
[1] 29.76932 41.46605 25.69897 25.35080 33.47488
```

16.11. Estadística descriptiva

R te proporciona todas las funciones que necesites para hacer un estudio estadístico, empezando por una descripción de la muestra a base de medidas de posisición centrales, no centrales, de dispersióin y de forma.

Como siempre, cada una de estas funciones tiene sus propios parámetros (además de la muestra), así que no dudes en consultar la ayuda de R sobre cada función que necesites. Puedes encontrar opciones realmente interesantes. Las funciones más comunes para estos casos son:

`mean(x)` Calcula la media aritmética de los valores de un vector numérico `x`.

`median(x)` Calcula la mediana de los valores de un vector numérico `x`.

`var(x)` Calcula la varianza de los valores `x`, que puede ser un vector o una matriz.

`cov(x,y)` Calcula la covarianza de `x` y `y`, que pueden ser dos vectores o dos matrices.

`cor(x,y)` Calcula la correlación entre de `x` y `y`, que pueden ser dos vectores o dos matrices.

`sd(x)` Calcula la desviación estándar de los valores `x`, que puede ser un vector o una matriz.

`range(x)` Devuelve un vector con los valores mínimo y máximo encontrados en `x`, que puede ser un vector o una matriz.

`fivenum(x)` Devuelve un vector con el mínimo, el máximo y los cuartiles del vector `x`.

[3]En algunas tablas encontrarás que los valores son para la probabilidad complementaria a esta, i.e. $P(X > x) = 1 - P(X \leq x)$

`summary(x)` Como `fivenum()` pero insertar la media (`mean()` entre la mediana y el tercer cuartil. Además define los nombres de los resultados.

```
> x = rnorm (100)
> mean (x)
[1] -0.06673138
> median (x)
[1] -0.01544592
> var (x)
[1] 1.081379
> sd (x)
[1] 1.039894
> range (x)
[1] -3.122787  2.667167
> fivenum (x)
[1] -3.12278656 -0.76220579 -0.01544592  0.58888639  2.66716657
> summary (x)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-3.12300 -0.75680 -0.01545 -0.06673  0.57860  2.66700
```

16.12. Inferencia estadística

Si no tienes conocimientos de inferencia estadística será mejor que te saltes este apartado. Vamos a ver que también para tareas de inferencia estadística R cuenta con multitud de funciones.

Veamos algunos ejercicios de inferencia estadística, utilizando como fuente de datos la tabla contenida en el fichero `muestra.dat` No te vamos a dar una referencia amplia ni profunda, pero debería ser suficiente para que aprendas cómo debes buscar las funcionalidades que necesites para tus problemas.

Tests de normalidad

Consideremos la variable `Lectura` y vamos si se puede decir que se distribuye según una distribución normal $N(\mu, \sigma)$. Para ello queremos utilizar los tests de normalidad de Kolmogorov-Smirnov y Shapiro-Wilk. Considera un nivel de significación del 99 %.

No sabemos qué funciones proporciona `R` para estos tests, así que ejecutamos `help.search ("test")` y buscamos los tests deseados. Encontramos entre éstos:

```
ks.test(ctest)          Kolmogorov-Smirnov Tests
shapiro.test(ctest)     Shapiro-Wilk Normality Test
```

Las funciones que necesitas no se encuentran en el paquete `base`, por lo que tendrás que cargar el paquete en el que se encuentran. Para ello utiliza la función `library()` como verás más adelante.

Para el test de Kolmogorov-Smirnov necesitas estimadores los parámetros de la distribución con la que quieras comparar la muestra. En esta caso has de estimar la media y la desviación típica, y una buena forma es utilizar los estimadores insesgados por analogía que son la media muestral y la cuasi-desviación típica muestral.

Utilizaremos en el ejemplo la desviación típica, para dejarte como ejercicio substituirla por la cuasi-desviación típica muestral definiendo tu propia función que la calcule.

```
> hoja.de.datos = read.table ("muestra.dat")
> attach (hoja.de.datos)
> library (ctest)
> shapiro.test (Lectura)

Shapiro-Wilk normality test
```

```
data:  Lectura
W = 0.9577, p-value = 0.0714

> ks.test (Lectura, "pnorm", mean = mean (Lectura), sd = sd (Lectura))

One-sample Kolmogorov-Smirnov test

data:  Lectura
D = 0.0907, p-value = 0.8053
alternative hypothesis: two.sided
```

Este este ejemplo, dado que los p-valores de los tests son $0{,}0714, 0{,}8053 > 0{,}01 = \alpha = 1 - 0{,}99$ se puede decir que el número de libros leidos anualmente se distribuye según una normal.

Contraste de hipótesis

Una vez comprobada la normalidad de una variable hay una serie de tests interesantes que puedes utilizar. Aprovechando esto vamos a plantear el siguiente un contraste de hipótesis para la media de la variable `Lectura`. La hipótesis nula es que la media es 15. Considera un nivel de confianza del 99 % ($1 - \alpha = 0{,}99$)

$$\left.\begin{array}{l} H_0{:}\mu = 15 \\ H_1{:}\mu \neq 15 \end{array}\right\}$$

Para este contraste tiramos del test T de Stundent para una muestra, que buscando como antes encontrarás que lo proporciona la función `t.test()`. Con una mirada a la ayuda de esta función verás que debes especificar la media con la que quieres constrastar (`mu = 15`) y el nivel de confianza (`conf.level = 0.99`).

```
> t.test (Lectura, mu = 15, conf.level = 0.99)

One Sample t-test

data:  Lectura
t = -1.8956, df = 49, p-value = 0.06392
alternative hypothesis: true mean is not equal to 15
99 percent confidence interval:
 10.89657 15.70343
sample estimates:
mean of x
     13.3
```

Dado el p-valor del contraste, $0{,}06392 > 0{,}01 = \alpha$, no se rechaza la hipótesis nula; se puede considerar que el número medio de libros leidos anualmente es 15, con un nivel de confiaza del 99 %.

Independencia de variables

Considera ahora la dos variables `Ingresos` y `Habitat`. Ambas son vectores de caracteres y por lo tanto al estar en una hoja de datos han sido convertidas a factores. Vamos a contrastar si estas variables son independientes, utilizando la tabla de contingencia y el test χ^2. Considera un nivel de confianza del 95 %.

De nuevo para averiguar qué funciones nos proporciona R buscamos en la ayuda: ejecutando `help.search ("contingency")` encontramos

```
ftable(base)            Flat Contingency Tables
```

Con la misma forma de búsqueda encontramos la función para el test χ^2 con `help.search ("test")`

```
chisq.test(ctest)       Pearson's Chi-squared Test for Count Data
```

Aplicamos el test χ^2 a la tabla de contingencia de las variables `Ingresos` y `Habitat`:

```
> t = ftable(Ingresos, Habitat)
> t
         Habitat Rural Urbano
Ingresos
<100                 2      0
100-200             13      9
200-300              5     10
300-400              1      3
400-500              0      4
>500                 0      2
> chisq.test(t)

Pearson's Chi-squared test

data:  t
X-squared = 10.6105, df = 5, p-value = 0.05967
> qchisq (0.95, 5)
[1] 11.07050
```

La región crítica para este test es $W = \chi^2 > \chi^2_{5,0,05}$, y dado que el estadístico χ^2 es $10{,}6105 < 11{,}07050 = \chi^2_{5,0,05}$ podemos considerar que las variables son independientes.

16.13. Gráficas

La función más frecuentemente usada para crear gráficas es `plot()`. Esta función representa uno o dos vectores numéricos como una nube de puntos, un factor mediante un diagrama de barras (`barplot`), un factor y un vector numérico con un diagrama de cajas (`boxplot`), y un par de factores como un histograma con cada barras dividida según el factor.

Para verlo más claro ejecuta la función `plot()` con las variables de la tabla del fichero `muestra.dat`. Prueba las siguientes combinaciones: `plot (Lectura)`, `plot (Lectura, TV)`, `plot (Ingresos)`, `plot (Ingresos, Habitat)`

Para representar vectores numéricos agrupando sus valores en intervalos utilizamos un histograma. La función `hist()` hace exactamente eso.

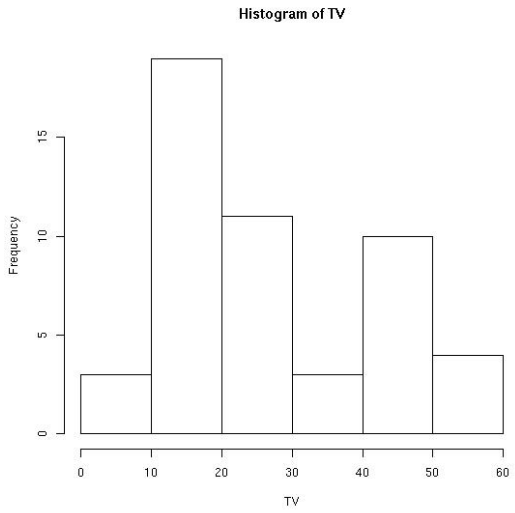
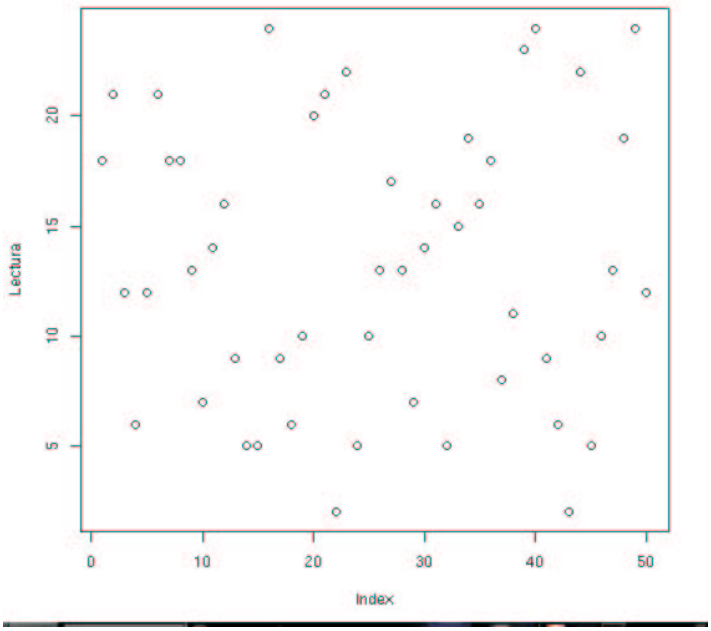
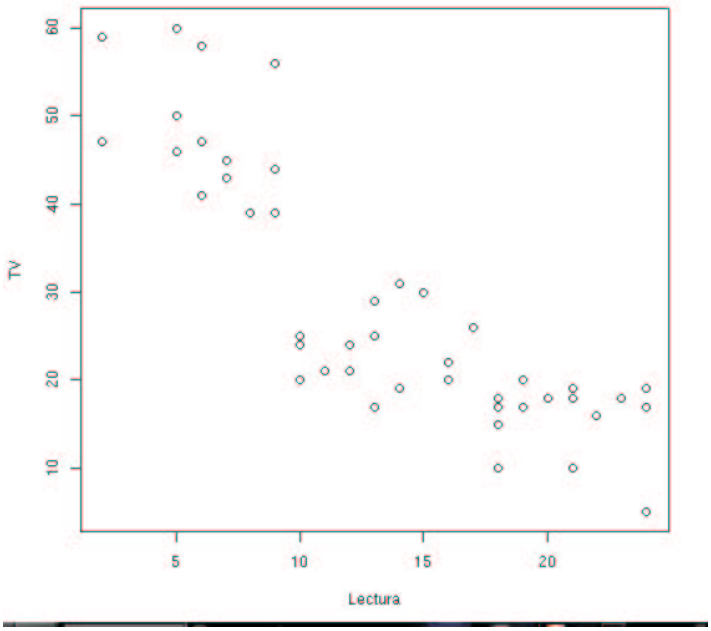


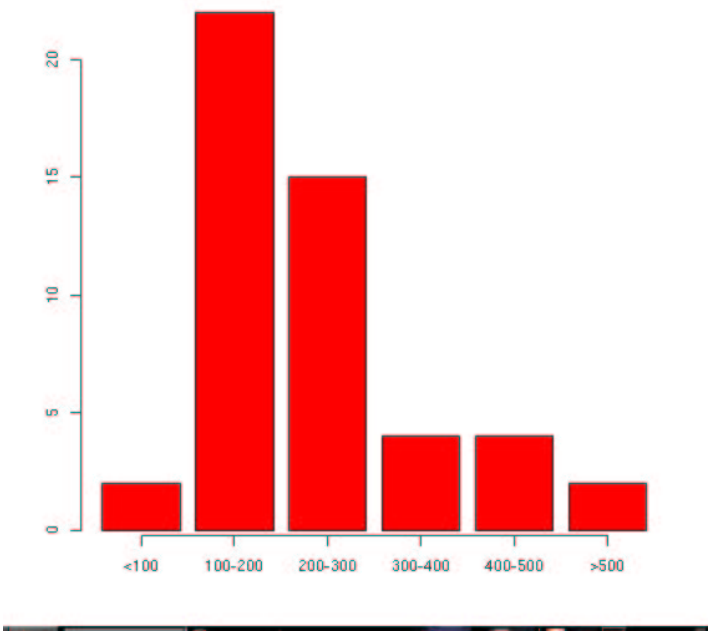
Figura 16.2: Histograma obtenido con `hist (TV)`



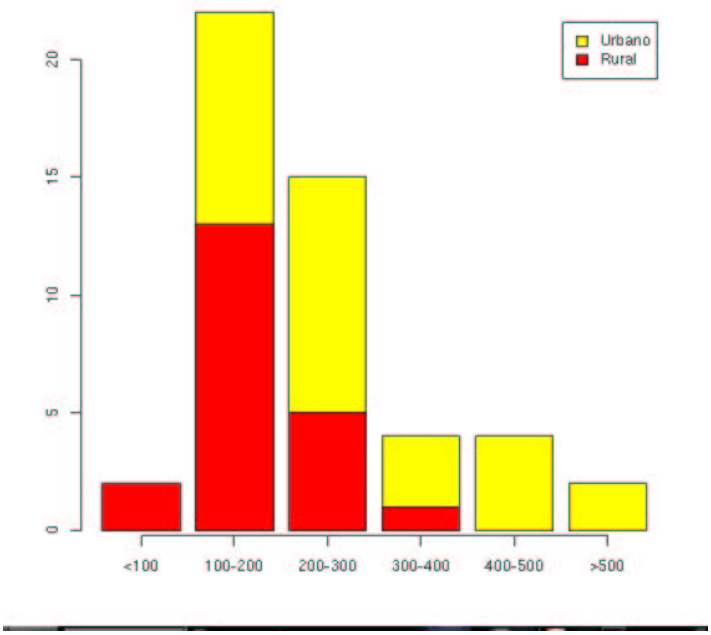
(a) `plot (Lectura)`



(b) `plot (Lectura, TV)`



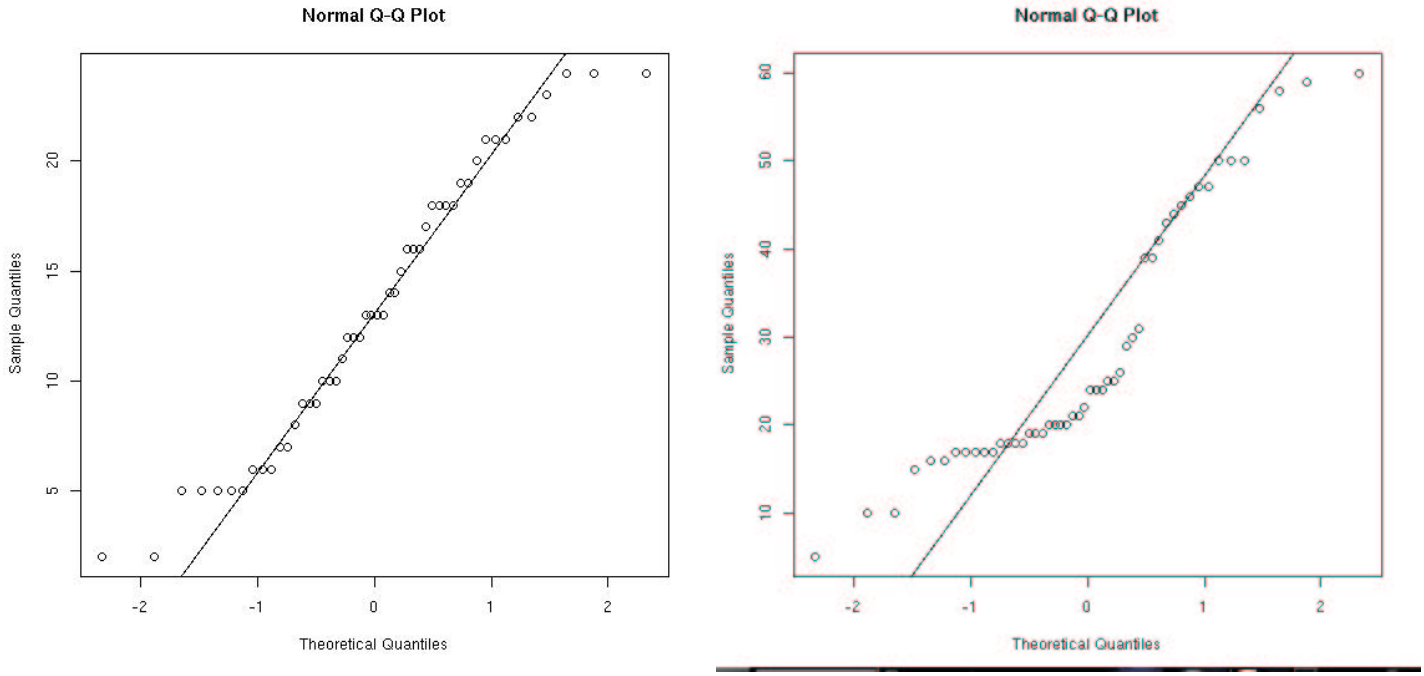
(c) `plot (Ingresos)`



(d) `plot (Ingresos, Habitat)`

Figura 16.3: Las distintas formas de la función `plot()`

Otras gráficas interesantes son las de comparación de distribuciones, basadas en los cuantiles. La función `qqnorm ()` toma un vector numérico y lo representa comparando sus cuantiles con los cuantiles teóricos (los que se esperarían en una muestra que siguiera una distribución normal). La función `qqline()` añade a la gráfica la recta que pasa por los cuartiles de los datos y la distribución.



(a) Gráfico Q-Q para `Lectura`

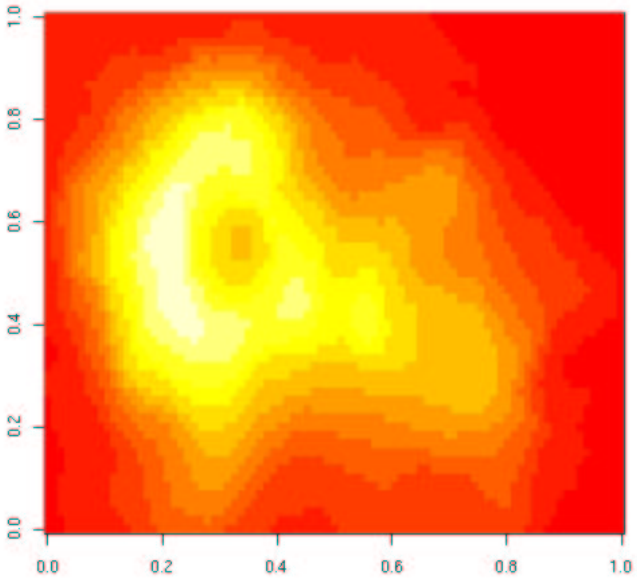
(b) Gráfico Q-Q para `TV`

Figura 16.4: Gráficos Q-Q para dos variables, una que se comporta como una normal y otra que no

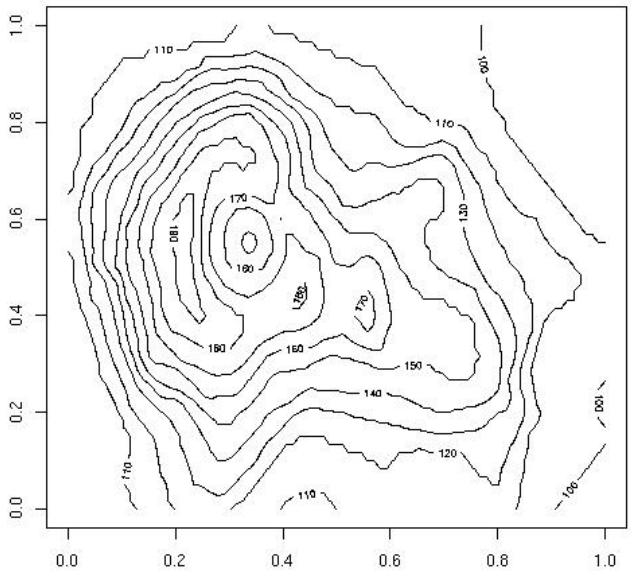
Para representar matrices puedes optar por ver sus valores como imágenes. La función `image()` lo hace a modo de una rejilla de rectángulos de diferentes colores para representar el valor de las celdas. Para verlo en forma de contorno usa `contour()`, y para verlo en perspectiva utiliza `persp()`.

Los gráficos de sectores te dan una impresión clara a primera vista de los porcentajes que tienen cada "clase" en una clasificación de datos. Por ejemplo para ver los porcentajes de personas según las edades representamos en un gráfico de sectores el número de individuos que tiene cada edad. Una forma de hacerlo (seguramente no es la mejor) sería:

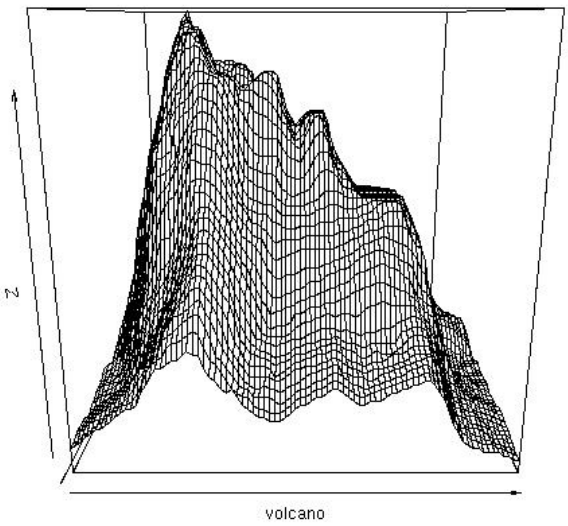
```
> pie (tapply (Edad, fedad, length))
```



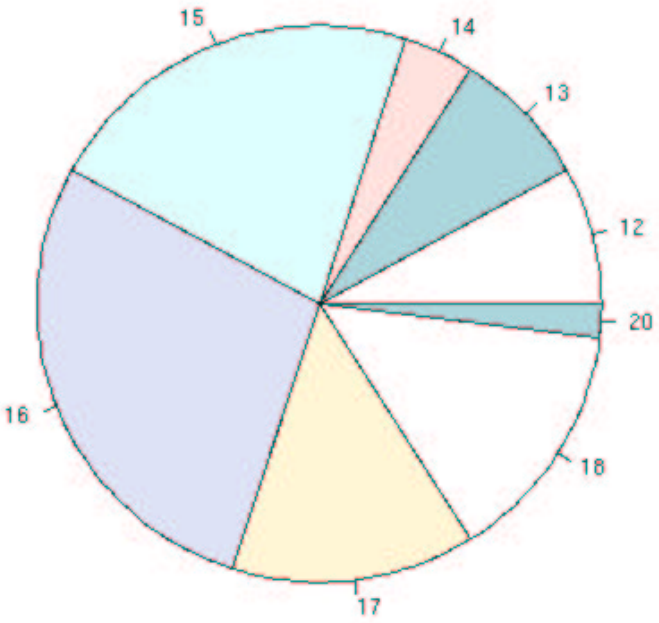
(a) `image(volcano)`



(b) `contour(volcano)`



(c) `persp(volcano)`



(d) `Gráfica de sectores para la Edad`

Figura 16.5: Gráficas para representar matrices. El objeto `volcano` lo obtenemos al ejecutar `data(volcano)`

TEMA 17

Yacas

En este tema aprenderás a utilizar el programa Yacas para relizar cálculos simbólicos. Este tipo de cálculos resulta de gran utilidad puesto que funciona de la misma forma que harías tú mismo a mano, operando con expresiones matemáticas sin calcular el valor numérico de cada expresión.

17.1. ¿Qué es Yacas?

Yacas es un Sistema de Álgebra Computacional (en inglés CAS, de Computer Algebra System) de propósito general fácil de usar. Un Sistema de Álgebra Computacional (SAC) es un programa que permite manipulaciones simbólicas sobre expresiones matemáticas, reduciendo el tiempo necesario para realizar cálculos embarazosos pero triviales. Esto se hace no mediante números, sino mediante símbolos, por lo que el resultado de una operación con expresiones matemáticas es una nueva expresión matemática.

Yacas está construido sobre su propio lenguaje de programación diseñado para este propósito, en el que se pueden implementar fácilmente nuevos algoritmos. Además, incorpora una documentación extensa sobre las funcionalidades que implementa y los métodos usados para implementarlas.

Entre las características que Yacas implementa encontramos precisión arbitraria, números racionales, vectores, números complejos, cálculos con matrices (incluyendo inversas, determinantes y resolución de sistemas lineales), derivación, series de Taylor, resolución numérica (método de Newton), y un montón más de algoritmos no matemáticos. Tiene también soporte básico para polinomios en una variable, integración de funciones y cálculo tensorial.

En la web del proyecto Yacas (`http://yacas.sourceforge.net` encontrarás el código fuente del programa y toda su documentación: desde un tutorial básico hasta una guía de programación en Yacas. Parte del texto de este tema son traducciones de trozos de la documentación oficial de Yacas que se distribuye junto con el programa bajo licencia GPL.

17.2. Empezando con Yacas

Yacas tiene un intérprete de comandos que nos permite ejecutar funciones para probar lo que queremos programar, para luego escribir un script (un fichero de comandos a modo de guión). Para entrar al intérprete de Yacas basta con ejecutar el comando `yacas` en una consola o terminal. Para salir del intérprete puedes pulsar `Control-C` o bien ejecutar la función `Exit();`. Pulsar `Control-C` también es útil para detener inmediatamente la ejecución del intérprete de Yacas (o de un script escrito para Yacas).

```
$ yacas
[editvi.ys] [unix.ys]
True;
Numeric mode: "Gmp"
To exit Yacas, enter  Exit(); or quit or Ctrl-c. Type ?? for help.
Or type ?function for help on a function.
Type 'restart' to restart Yacas.
To see example commands, keep typing Example();
In>
```

También existe una interfaz gráfica llamada Proteus. Si la tienes instalada deberías poder iniciarla ejecutando el comando `proteusworksheet`. Esta utilidad te proporciona un entorno más cómodo e integrado en el que puedes probar código en el intérprete e ir escribiendo un script en el editor que incluye.

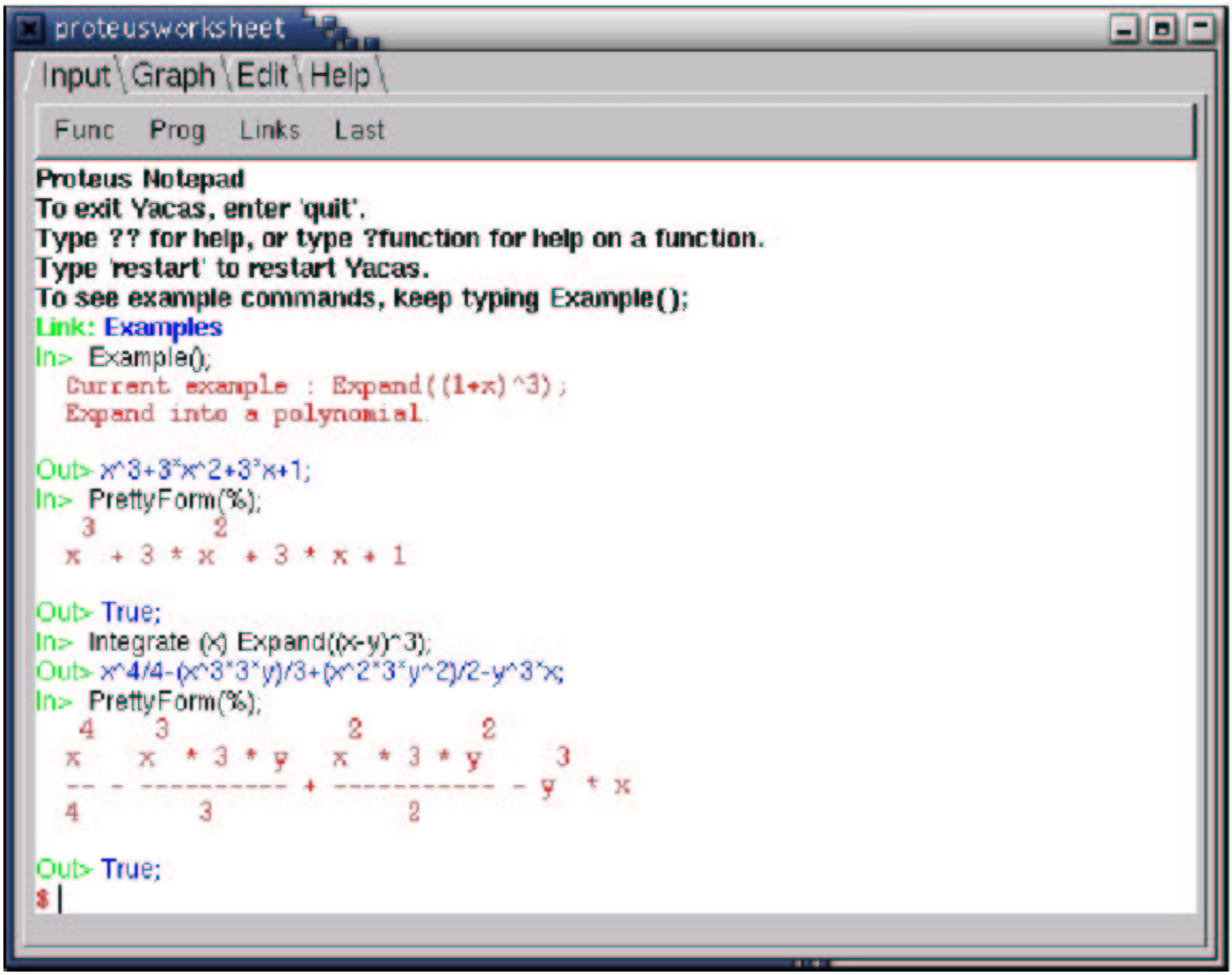


Figura 17.1: Proteus, interfaz gráfica para Yacas

La última línea (`In>`) es el prompt de entrada de Yacas, que nos indica que el intérprete está esperando nuestras órdenes. Todas las órdenes de Yacas deben terminar con un punto y coma (`;`), aunque el intérprete suele añadirlo si no lo ponemos. Sin embargo a la hora de programar en Yacas veremos que el punto y coma hay que escribirlo siempre, de modo que es mejor acostumbrarse a ponerlo para evitar posteriores dolores de cabeza.

Para ir abriendo el apetito sigue la sugerencia de Yacas: ejecuta la función `Example();` varias veces. Por supuesto no tienes que teclear el comando todas las veces, ya que Yacas cuenta con edición de línea. Esto incluye que todos los comandos que ejecutes en Yacas se almacenarán en un historial, y para repetirlos puedes recuperarlos pulsando la tecla de cursor hacia arriba. Este historial queda guardado en el fichero `~/.yacas_history`. Veamos la salida que produce Yacas tras varias ejecuciones de `Example();`.

```
In> Example();
Current example : 40!;

Simple factorial of a number.
```

```
Out> 815915283247897734345611269596115894272000000000;
In> Example();
Current example : D(x)Sin(x);

Taking the derivative of a function (the derivative of Sin(x) with
respect to x in this case).

Out> Cos(x);
In> Example();
Current example : Taylor(x,0,5)Sin(x);

Expanding a function into a taylor series.

Out> x-x^3/6+x^5/120;
In> Example();
Current example : Integrate(x,a,b)Sin(x);

Integrate a function.

Out> Cos(a)-Cos(b);
In> Example();
Current example : Solve(a+x*y==z,x);

Solve a function for a variable.

Out> (z-a)/y;
In> Example();
Current example : Limit(x,0)Sin(x)/x;

Take a limit.

Out> 1;
```

En cualquier momento puedes acceder a la documentación de una función ejecutando la función precedida por un signo de interrogación, p.ej. `?IsFreeOf`. Un detalle importante de Yacas es que, como muchos lenguajes del mundo de Un*x, es sensible a las mayúsculas; esto es, no es lo mismo `esto` que `Esto` ni que `eSTO`.

Estos ejemplos ilustran lo fácil que es obtener buenos resultados en cálculos simbólicos con Yacas. Sin embargo es probable que el resultado del siguiente ejemplo no te resulte fácil de interpretar:

```
In> Integrate(x) Expand((x-y)^3);
Out> x^4/4-(x^3*3*y)/3+(x^2*3*y^2)/2-y^3*x;
```

la forma en que Yacas recibe nuestra orden de integrar $\int (x-y)^3\,dx$ no es nada críptica, pero su respuesta no es precisamente fácil de leer. Esta forma de mostrar las expresiones matemáticas es la normal en Yacas, pero no la única. Podemos pedir a Yacas que muestre las expresiones de forma más "bonita" con la función `PrettyForm()`.

```
In> PrettyForm(%)

 4    3           2        2
x    x  * 3 * y   x  * 3 * y     3
-- - ---------- + ----------- - y  * x
4        3             2

Out> True;
```

El signo `%` es una referencia que apunta siempre al último valor devuelto por el intérprete, es decir lo que aparezca a la derecha del último `Out>`. Esta referencia sólo está disponible cuando usamos Yacas como intérprete interactivo, no se puede utilizar desde un script.

Otras formas de mostrar una expresión son las utilizadas en el lenguaje de programación C y en el lenguaje tipográfico TEX. Para mostrar una expresión de estas formas están las funciones `CForm()` y `TeXForm()` respectivamente. Fíjate que estas funciones no devuelven la expresión que muestran, por lo que no podemos utilizar la referencia `%` dos veces seguidas para mostrar una expresión (la segunda mostraría la expresión lógica `True`). Por eso escribimos `Integrate(x) Expand((x-y)^3)` en cada línea:

```
In> CForm (Integrate(x) Expand((x-y)^3))
Out> "pow(x, 4.) / 4. - ( pow(x, 3.) * 3. * y)  / 3. + ( pow(x, 2.) *
3. * pow(y, 2.))  / 2. - pow(y, 3.) * x";
In> TeXForm (Integrate(x) Expand((x-y)^3))
Out> "$\frac{x ^{4}}{4}  - \frac{x ^{3} 3 y}{3}  + \frac{x ^{2} 3 y ^{
2}}{2} - y ^{3} x$";
```

La función `CForm()` devuelve la expresión escrita en lenguaje C, de forma que podemos incluirla en el código de un programa en C. La función `TeXForm()` devuelve la expresión escrita en el lenguaje tipográfico TEX, de forma que podemos incluirla en un documento escrito en TEX o LATEX. Por ejemplo, este libro está escrito en LATEX, lo que me permite incluir la salida de `TeXForm()` en este mismo párrafo con sólo copiar y pegar. De esta forma puedo escribir:

$$\int (x-y)^3\,dx = \frac{x^4}{4} - \frac{x^3 3y}{3} + \frac{x^2 3y^2}{2} - y^3 x$$

En ocasiones nos interesa conocer el valor numérico de una expresión determinada, y posiblemente sólo nos interese un número determinado de decimales. Yacas es un lenguaje de precisión arbitraria, lo que significa que puedes pedirle toda la precisión que quieras, pero ten cuidado no le pidas demasiada precisión si tu máquina no es realmente potente.

La función `Precision()` establece el número de cifras decimales con las que se aproximarán los valores. Estas aproximaciones se realizan con la función `N()`, que además permite saltarse la precisión establecida por `Precision()` pasándosela como segundo parámetro. En el siguiente ejemplo se ilustra esto:

```
In> alpha := Sqrt (Pi ())
Out> Sqrt(3.14159265358979323846264338327950288419716 94);
In> Precision (2);
Out> True;
In> N (alpha)
Out> 1.77;
In> Precision (20);
Out> True;
In> N (alpha)
Out> 1.77245385090551602729;
In> N (alpha, 50)
Out> 1.77245385090551602729816748334114518279754945629866;
```

Normalmente no hay problema por pedirle a Yacas unos cientos de dígitos de precisión, pero siempre que vayas a trabajar con números o precisiones enormes recuerda que el tiempo necesario para los cálculos aumentará rápidamente.

17.3. Variables y funciones

En Yacas se entiende por variable un objeto al que se puede asignar un valor. Las variables pueden llamarse como quieras siempre que el nombre empiece por una letra y continúe con letras y números. Para asignar un valor a una variable se utiliza el operador de asignación `:=` en lugar de `=` (este último se utiliza como comparador). En cualquier momento puedes ver el valor de una variable tecleando su nombre como si fuera un comando de Yacas, y esto también es válido en un script. Asignaciones válidas son:

```
In> n := 10;
Out> 10;
In> m := n!;
Out> 3628800;
```

```
In> n
Out> 10;
In> m
Out> 3628800;
```

Si en algún momento quieres que una variable pierda su valor no tienes más que "limpiarla" con la función `Clear()`:

```
In> m
Out> 3628800;
In> Clear (m);
Out> True;
In> m
Out> m;
```

Las funciones son como las variables, con el añadido de que pueden depender de una o más variables. Además con Yacas podemos sobrecargar una función, esto es, definir dos funciones con el mismo nombre pero distinto número de variables de forma que se evalúe una u otra según el número de parámetros que reciba.

Como ejemplo definimos la función `Area` de dos formas: si recibe un parámetro devuelve el área del círculo con el radio igual a ese parámetro, pero si recibe dos parámetros devuelve el área de la elipse que tenga esos dos parámetros como radios.

```
In> Area (r) := Pi () * r^2;
Out> True;
In> Area (a, b) := Pi () * a * b;
Out> True;
In> Area (3);
Out> 28.2743338823;
In> Area (3, 5);
Out> 47.1238898035;
```

Al definir una función, Yacas no evalúa la parte de la derecha sino que se asigna como expresión simbólica. Esto puede no interesarte en casos como el que ilustra el siguiente ejemplo:

```
In> f(x) := x^5;
Out> True;
In> g(x) := D(x) f(x);
Out> True;
In> g(x)
Out> 5*x^4;
In> g(2)
Out> 5*x^4;
```

Sería deseable que `g(2)` devolviera el valor de la función `g(x)` cuando `x` vale 2, pero no lo hace porque `g(x)` almacena la expresión `5*x^4` sin ninguna referencia a la necesidad de evaluarla en un valor determinado. Sin embargo si indicamos a Yacas que evalúe la expresión antes de asignarla a la función (mediante `Eval()`) podremos pedirle que evalúe la función `g(x)` en valores determinados. Esto que parece un follón es en realidad simple, el siguiente ejemplo muestra el código que funciona como es de esperar:

```
In> g(x) := Eval (D(x) f(x));
Out> True;
In> g(x)
Out> 5*x^4;
In> g(2)
Out> 80;
```

17.4. Listas

Uno de los elementos más importantes del lenguaje de Yacas son las listas, que como era de esperar son grupos de objetos ordenados. Yacas representa las listas con sus elementos entre llaves y separados por comas. Los vectores son listas, y las matrices son listas de listas. De hecho, cualquier expresión matemática en Yacas puede ser transformada en una lista. Para acceder a los elementos de una lista puedes usar la notación habitual en los corchetes, y no sólo con números sino también con secuencias de ellos. Los elementos de la lista pueden ser cualquier tipo de objetos.

```
In> lista := {a, b, c, d, e, f};
Out> {a,b,c,d,e,f};
In> lista[2];
Out> b;
In> lista[2 .. 4];
Out> {b,c,d};
```

Fíjate que los espacios a ambos lados del operador `..` son necesarios para distinguir estos puntos de los que podrían formar parte de un número.

También puedes indexar una lista con palabras, no sólo con números. Esto te permite tener pequeñas bases de datos en forma de listas asociativas con parejas clave – valor.

```
In> boy := {};
Out> {};
In> boy["nombre"] := "Eric";
Out> True;
In> boy["apellido"] := "Cartman";
Out> True;
In> boy["edad"] := 7.34;
Out> True;
In> boy["educado"] := False;
Out> True;
In> boy
Out> {{"educado",False},{"edad",7.34},{"apellido","Cartman"},
{"nombre","Eric"}};
```

17.5. Álgebra Lineal

Los vectores de dimensión fija se representan mediante listas. La lista `{1,2,3}` es el vector $(1, 2, 3)$. Las matrices se representan como vectores de vectores. Dado que los vectores son realmente listas, sus elementos pueden asignarse igual que los de las listas:

```
In> l := ZeroVector (3);
Out> {0,0,0};
In> l;
Out> {0,0,0};
In> l[2] := 2;
Out> True;
In> l;
Out> {0,2,0};
```

Yacas puede multiplicar vectores, matrices y números del modo usual en álgebra lineal:

```
In> v := {1, 0, 0, 0}
Out> {1,0,0,0};
In> E4 := { {0, u1, 0,  0}, \
```

```
In>         {d0, 0, u2, 0}, \
In>         {0, d1, 0,  0}, \
In>         {0, 0,  d2, 0} }
Out> {{0,u1,0,0},{d0,0,u2,0},{0,d1,0,0},{0,0,d2,0}};
In> PrettyForm (%)

/                            \
| ( 0 )  ( u1 ) ( 0 )  ( 0 )  |
|                             |
| ( d0 ) ( 0 )  ( u2 ) ( 0 )  |
|                             |
| ( 0 )  ( d1 ) ( 0 )  ( 0 )  |
|                             |
| ( 0 )  ( 0 )  ( d2 ) ( 0 )  |
\                            /

Out> True;
In> CharacteristicEquation (E4, x)
Out> x^4-x*u2*d1*x-u1*d0*x^2;
In> Expand (%, x)
Out> x^4-(u2*d1+u1*d0)*x^2;
In> PrettyForm (%)

 4                          2
x  - ( u2 * d1 + u1 * d0 ) * x

Out> True;
In> v  +  E4 * v  +  E4 * E4 * v  +  E4 * E4 * E4 * v
Out> {u1*d0+1,d0+(d0*u1+u2*d1)*d0,d1*d0,d2*d1*d0};
In> PrettyForm (%)

/                                \
| u1 * d0 + 1                    |
|                                |
| d0 + ( d0 * u1 + u2 * d1 ) * d0 |
|                                |
| d1 * d0                        |
|                                |
| d2 * d1 * d0                   |
\                                /

Out> True;
```

La librería estándar de Yacas incluye también cálculo del determinante y la inversa de una matriz, autovectores y autovalores (en casos simples) y resolución de sistemas de ecuaciones lineales del tipo $Ax = b$ donde A es una matriz y x y b son vectores.

17.6. Control de flujo

El lenguaje de Yacas incluye algunas construcciones y funciones para el control de flujo. Para ello necesitas saber también que Yacas te permite agrupar bloques de instrucciones de forma que aparezcan como una sola instrucción. Para ello simplemente encierra las instrucciones del bloque entre corchetes (`[` y `]`), o bien utiliza la función `Prog()` pasándole las instrucciones separadas por `;`.

Para los bucles disponemos de las funciones `ForEach()` y `While()`. La función `ForEach (x, list) body` ejecuta el bloque de instrucciones `body` para cada elemento de la lista `list` asignando el valor de cada elemento a la variable `x` en cada interación. La función `While (predicate) body` repite el bloque `body` hasta que la condición `predicate` devuelva `False`.

Para los condicionales está la función `If (predicate, body1, body2)`, en la que se ejecuta el bloque `body1` si `predicate` devuelve `True` o ejecuta el bloque `body2` si `predicate` devuelve `False`, y en ambos casos devuelve el valor devuelto por el bloque ejecutado. El segundo bloque es opcional. Si llamas a `If (predicate, body1)` se ejecutará el bloque `body1` si `predicate` devuelve `True` y devolverá el valor que devuelva `body1`, o bien se devolverá `False` si así lo hace `predicate`.

Como ejemplo de control de flujo construimos una lista con los números enteros pares de 2 a 20 y calculamos el producto de los que no sean divisibles por 3. Luego definimos el factorial de un número de forma recursiva.

```
In> L := {};
Out> {};
In> i := 2;
Out> 2;
In> While (i <= 20) [ \
In>   L := Append (L, i); \
In>   i := i + 2; \
In> ];
Out> True;
In> L;
Out> {2,4,6,8,10,12,14,16,18,20};
In> answer := 1;
Out> 1;
In> ForEach (i, L) [ \
In>   If ( Mod (i, 3) != 0, \
In>     answer := answer * i \
In>   ); \
In> ];
Out> True;
In> answer;
Out> 2867200;
In> Factorial (x) := [ \
In>   If ( IsInteger (x) And x >= 0, \
In>     If (x = 0, 1, \
In>       x * Eval (Factorial (x-1)) ), \
In>   False ); \
In> ];
In> Factorial (5)
Out> 120;
In> Factorial (25)
Out> 15511210043330985984000000;
```

Una barra invertida \ al final de una línea indica al intérprete de Yacas que la línea no está terminada, sino que continúa después del salto de carro. En este ejemplo hemos utilizado esto descaradamente. Esto se puede utilizar en el intérprete para hacer que nuestros comandos sean más cómodos de leer, pero no es necesario al escribir un script.

17.7. Gráficas

Yacas incorpora la posibilidad de representar gráficas bidimensionales utilizando GNUplot, si bien esta capacidad depende de que tengas instalado también el programa GNUplot. Para saber si tu versión de Yacas tiene soporte para GNUplot fíjate en la primera línea que imprime el intérprete cuando arranca, si aparece `[gnuplot.ys]` es que tu versión de Yacas soporte para gráficas con GNUplot[1].

```
$ yacas
[editvi.ys] [gnuplot.ys] [unix.ys]
```

[1] Esto era cierto hasta la versión 1.0.50 de Yacas, pero en el momento de escribir esta nota la versión 1.0.51 de mi máquina no aparece `[gnuplot.ys]` pero puede hacer las gráficas con la función `Plot2D`

La función para representar gráficas es `Plot2D()` y recibe dos parámetros: la expresión (o lista de expresiones) para representar y el intervalo de la variable en la forma `valor_minimo : valor_maximo`.

En anteriores versiones de Yacas, la función para representar gráficas era `GnuPlot()` y recibía cuatro parámetros: el valor mínimo de la variable, el valor máximo de la variable, el número de puntos utilizados para la representación y la función para representar.

En los siguientes ejemplos calculamos el el polinomio de Taylor de orden 5 para la función $\sin(x)$ en el origen y la representamos (en rojo) junto con la función $\sin(x)$ (en verde).

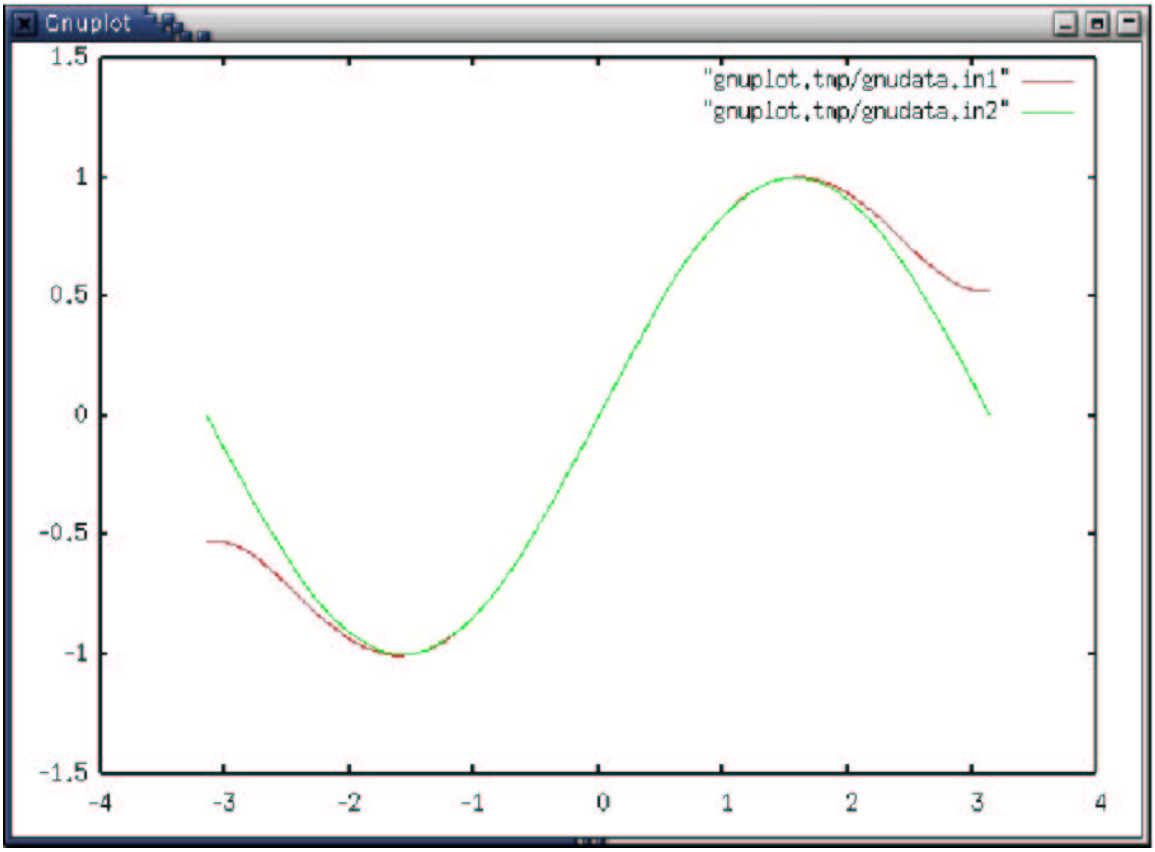


Figura 17.2: Representación gráfica con GNUplot desde Yacas

```
In> f(x) := Eval (Taylor (x, 0, 5) Sin(x) )
Out> True;
In> Plot2D ({f(x), Sin (x)}, -Pi:Pi)
Out> True;
In> GnuPlot (-Pi(), Pi(), 50, {f(x), Sin(x)})
GnuPlot: created file gnuplot.tmp/gnudata.in1
GnuPlot: created file gnuplot.tmp/gnudata.in2
Out> True;
```

17.8. Programando con Yacas

Si bien el intérprete de Yacas proporciona una interfaz cómoda para ejecutar cálculos, cuando éstos se complican resulta más práctico escribir un "script" y ejecutarlo con Yacas. Para hacer un programa con Yacas escribe en un fichero la secuencia de comandos que componen el programa, incluyendo definiciones de funciones. Los ficheros de script de Yacas suelen tener tener la extensión `.ys` (no es obligatorio).

Para ejecutar el script simplemente ejecuta Yacas dándole el nombre del fichero como parámetro. La opción `-c` del comando `yacas` hace que Yacas no muestre los prompts `In>` y `Out>` (útil para sesiones no interactivas, como el caso de

ejecutar un script). El siguiente ejemplo muestra como ejecutar un script en Yacas, en este caso el script es el fichero de ejemplo `lagrange.ys`

```
$ yacas -c lagrange.ys
```

Fichero `lagrange.ys`

```
f(x) := 1.0 / (1.0 + x^2);
xl := {-5, -2, 0, 3, 5};
yl := {};
n := Length (xl);
For ( i := 1 , i <= n , i++ ) [
  yl := Append (yl, Eval (f (xl[i])));
];
pol (x) := Eval (LagrangeInterpolant (xl, yl, x));
Plot2D ({f(x), pol(x)}, xl[1]:xl[n]);
```

Ejemplo 17.1: Script en Yacas que muestra un interpolante de Lagrage para una nube de cinco puntos tomados de la función $f(x) = \frac{1}{1+x^2}$. En este ejemplo el interpolante de Lagrange no es el adecuado, pero nuestra intención no es interpolar $f(x)$

Otra forma de ejecutar un script es desde el intérprete, mediante las funciones `Load()` y `Use()`. La función `Load("script.ys")` lee el fichero `script.ys` y evalúa todas las expresiones que encuentre en él. Siempre devuelve `True`. `Use()` hace lo mismo que `Load()` con la salvedad de que sólo lee el fichero si no ha sido leído antes por otra llamada `Load()` o `Use()`.

Un uso común de `Use()` es cargar un fichero con funciones con las que queremos trabajar desde el intérprete, de modo que no tenemos que teclear las funciones cada vez ni tampoco hay que ejecutar el script. La función `Load()` resulta útil para ejecutar un script desde el intérprete cuando queremos hacerlo varias veces modificando el script sin salir del intérprete.

17.9. Un ejemplo real

A continuación explicamos un ejemplo del uso de Yacas "en la vida real": calcular la sucesión de Sturm asociada a un polinomio real de una variable real. Esta sucesión se utiliza en el teorema de Sturm para calcular el número de raíces reales de un polinomio en un intervalo de la recta real. El algoritmo para calcular esta sucesión de polinomios no es complicado, pero por si no lo conoces aquí tienes una breve explicación:

El algoritmo de Sturm parte de un polinomio con coeficientes reales $p(x)$ y su derivada $p'(x)$. Tomando $f_0(x) = p(x)$ como primer polinomio y $f_1(x) = p'(x)$ como segundo polinomio, se realizan sucesivas divisiones euclídeas utilizando en cada división el divisor de la anterior como dividendo y el opuesto del resto de la anterior como divisor. Al cabo de un número finito de iteraciones, en las que $f_j(x)$ es el resto de la división $j-1$, se obtiene un resto nulo en la iteración $m+1$.

$$\begin{aligned} f_0(x) &= f_1(x)q_1(x) - f_2(x) \\ f_1(x) &= f_2(x)q_2(x) - f_3(x) \\ &\vdots \\ f_{m-2}(x) &= f_{m-1}(x)q_1(x) - f_m(x) \\ f_{m-1}(x) &= f_m(x)q_m(x) \end{aligned}$$

A partir de los polinomios f_j obtenidos en estas divisiones se construye la sucesión de Sturm asociada a $p(x)$, que viene dada por

$$\left\{\hat{f}_j(x) = \frac{f_j(x)}{f_m(x)}\right\}_{j=0}^{n}$$

El teorema de Sturm garantiza que el número de raíces del polinomio $p(x)$ en el intervalo $[a,b]$ es exactamente $V(a) - V(b)$, donde $V(x_0)$ es el número de variaciones de la sucesión de Sturm evaluada en el punto x_0, i.e. el número de cambios de signo en la sucesión $\left\{\hat{f}_0(x_0), \hat{f}_1(x_0), \ldots, \hat{f}_m(x_0)\right\}$.

Para implementar esto en Yacas definimos la función `Sturm()` que recibe el polinomio `P` junto con la variable en la que éste está expresado `x` y genera una lista `L` con el polinomio, su derivada y los sucesivos restos de las divisiones euclídeas. Para ahorrar esfuerzo computacional acotamos el número de términos de la lista con el grado del polinomio `P(x)`, ya que éste es el número máximo de divisiones necesarias para encontrar el máximo común divisor del polinomio y su derivada. Luego busca el último polinomio no nulo en `L`, que es el máximo común divisor del polinomio y su derivada, y construye la sucesión de Sturm en una nueva lista `S` que finalmente devuelve como valor de retorno de la función. La función `Simplify` simplifica cualquier expresión que reciba.

Fichero `sturm.ys`

```
Sturm (P, x) := [
    Local (L, S, m, i);

    L := List ();
    L := Append (L, P(x));
    L := Append (L, D(x) P(x));

    /* El número de términos en la sucesión de Sturm asociada a P(x)
     * es siempre no superior al grado del polinomio P(x) */
    For ( i := 3 , i <= Degree(P(x)) , i++ )
      L := Append (L, - Mod ( L[i-2] , L[i-1] ) );

    m := 0;
    While ( m < Length (L) And
      Not IsZero (L[m+1]) ) m++;

    S := List ();
    For (i := 1, i <= m, i++)
      S := Append (S, Simplify (Div ( L[i] , L[m]) ) );

    S;
];

P(x) := RandomPoly (x, 3, 1, 10);
PrettyForm (P(x));
S := Sturm (P, x);
PrettyForm (S);
```

Ejemplo 17.2: Script en Yacas que implementa el algoritmo de Sturm.

En el script escribimos esta función y añadimos los comandos necesarios para ejecutar un test. La función `RandomPoly()` genera un polinomio aleatorio, lo mostramos con `PrettyForm()`, calculamos su sucesión de Sturm y la mostramos con `PrettyForm()` de nuevo. El resultado de ejecutar el script es el siguiente:

```
     3         2
7 * x  + 10 * x  + 3 * x + 3


/                                        \
|     /           2                    \ |
| 9 * \ -96605 * x  + 73253 * x - 181452 / |
| ---------------------------------------- |
|                 2685619                  |
|                                          |
| 81 * ( -139 * x - 117 )                  |
| -----------------------                  |
|          19321                           |
|                                          |
```

```
| 1                                       |
\                                         /
```

Este ejemplo real ha sido extraido del programa presentado por unos alumnos como trabajo para asignatura "Álgebra Computacional" que se imparte en la Facultad de Matemáticas de la Universidad de La Laguna. En teoría debieron realizar el trabajo utilizando Maple V, pero propusieron al profesor la alternativa de programarlo en Yacas y éste aceptó con curiosidad. El programa aproxima las raíces reales de un polinomio utilizando el algoritmo de Sturm para aislar las raíces en intervalos y aproximándolas con el método de la bisección (raíces de multiplicidad impar) o con el método de cristalización simulada (raíces de multiplicidad par). El trabajo fue expuesto en clase y tuvo la calificación de sobresaliente, el código y las transparencias están disponibles en `http://www.fmat.ull.es/~miguev/algcomp/`

MÓDULO V

Programación

TEMA 18

FreePascal

El lenguaje de programación Pascal es sencillo y bastante didáctico, por lo que se suele enseñar en el primer año de algunas carreras técnicas como Matemáticas, Física o Informática. Normalmente, estos cursos o asignaturas de programación tratan de enseñar al estudiante los conceptos básicos de la programación de computadores sin entrar en demasiados detalles acerca del funcionamiento interno de los mismos. En este capítulo aprenderemos las herramientas básicas que se encuentran disponibles en GNU/Linux para programar en Pascal. El compilador que utilizaremos está siendo desarrollado por el proyecto **Free Pascal**, que proporciona un buen compilador de Pascal para múltiples plataformas, entre éstas GNU/Linux, MS-DOS, MS Windows, Amiga, MaC OS y otras.

Comenzaremos escribiendo un ejemplo muy sencillo de programa en Pascal, el típico "Hola Mundo". En cualquier editor escribimos el siguiente código y lo guardamos con el nombre `HolaMundo.pas`

Fichero `HolaMundo.pas`

```
{ Ejemplo 1 de Pascal para CILA }

Program HolaMundo;

Begin
  writeln ('Hola Mundo');
End.
```

Ejemplo 18.1: Ejemplo 1 de Pascal

Para compilar un programa escrito en Pascal con el compilador de FreePascal utilizamos el comando `ppc386` del siguiente modo:

```
$ ppc386 HolaMundo.pas
Free Pascal Compiler version 1.0.4 [2001/08/31] for i386
Copyright (c) 1993-2000 by Florian Klaempfl
Target OS: Linux for i386
Compiling HolaMundo.pas
Assembling holamundo
Linking holamundo
7 Lines compiled, 0.3 sec

$ ls
holamundo2    HolaMundo2.pas  holamundo2.o
```

Como podemos apreciar en los mensajes del compilador, él mismo se encarga de compilar, ensamblar y enlazar el programa para generar el fichero ejecutable `holamundo`. Para cambiar el nombre del ejecutable resultante se utiliza la opción `-onombredelejecutable` (sin dejar espacio entre la `o` y el nombre del ejecutable.

```
$ ppc386 -oHolaMundo HolaMundo.pas
Free Pascal Compiler version 1.0.4 [2001/08/31] for i386
Copyright (c) 1993-2000 by Florian Klaempfl
Target OS: Linux for i386
Compiling HolaMundo.pas
Assembling HolaMundo
Linking HolaMundo
7 Lines compiled, 0.3 sec

$ ls
HolaMundo    HolaMundo2.pas  holamundo2.o
```

Para ejecutar el programa resultante, hemos de recordar que debemos poner `./` delante del nombre del ejecutable:

```
$ HolaMundo
bash: HolaMundo: command not found
$ ./HolaMundo
Hola Mundo
```

Veamos ahora un ejemplo del uso de las "unidades" en Pascal. El concepto de unidades en Pascal es equivalente al de librerías en C. Se trata de ficheros binarios que obtenemos a partir de código fuente separado y luego enlazamos con el programa principal. Esto permite dividir el código de un programa en varios ficheros y evita tener que compilar todo el programa cada vez que se modifica una función. Con el uso de unidades basta con recompilar la unidad en la que se modifica el código fuente y volverla a enlazar con el programa. A diferencia del Borland Pascal, el compilador Free Pascal utiliza la extensión `ppu` (en lugar de `tpu`) para los ficheros binarios de unidades.

Tenemos para este ejemplo dos ficheros de código fuente en Pascal, `HolaMundo2.pas` y `saludos.pas`.

Fichero `HolaMundo2.pas`

```
{ Ejemplo 2 de Pascal para CILA }
{    Fichero: HolaMundo2.pas    }

Program HolaMundo2;
Uses
  Crt, Saludos ;
Var
  nombre : string ;
Begin
  TextColor(13) ;
  write ('>Cómo te llamas? ') ;
  TextColor(15) ;
  readln (nombre) ;
  TextColor(14) ;
  Saluda (nombre) ;
  TextColor(7) ;
End.
```

Ejemplo 18.2: Ejemplo 2 de Pascal

Fichero `saludos.pas`

```
{ Ejemplo 2 de Pascal para CILA }
{    Fichero: saludos.pas       }

Unit Saludos ;

Interface
```

```
Procedure Saluda ( mensaje : string ) ;

Implementation

Procedure Saluda ( mensaje : string ) ;
Begin
  writeln('Hola ', mensaje);
End;

Begin
End.
```

Ejemplo 18.3: Ejemplo 2 de Pascal

Debemos tener cuidado con un detalle: Los nombres de las unidades deben coincidir con el nombre del fichero en el que están escritas, i.e. la unidad `Saludos` no se puede escribir en un fichero llamado `Otronombre.pas`. Además, los ficheros en los que se implementan las unidades conviene que tengan el nombre completamente en **minúsculas**. De lo contrario, el compilador FreePascal no encontrará la unidad al compilar un programa que la use, pero podremos aún compilarla manualmente. El siguiente ejemplo ilustra la situación:

```
$ ls
HolaMundo2.pas  saludos.pas

$ mv saludos.pas otronombre.pas

$ ls
HolaMundo2.pas  otronombre.pas

$ ppc386 otronombre.pas
Free Pascal Compiler version 1.0.4 [2001/08/31] for i386
Copyright (c) 1993-2000 by Florian Klaempfl
Target OS: Linux for i386
Compiling otro.pas
otro.pas(4,6) Error: Illegal unit name: SALUDOS
otro.pas(10,1) Fatal: There were 1 errors compiling module, stopping

$ mv otronombre.pas Saludos.pas

$ rm *.o *.ppu

$ ls
HolaMundo2.pas  Saludos.pas

$ ppc386 HolaMundo2.pas
Free Pascal Compiler version 1.0.4 [2001/08/31] for i386
Copyright (c) 1993-2000 by Florian Klaempfl
Target OS: Linux for i386
Compiling HolaMundo2.pas
HolaMundo2.pas(6,16) Fatal: Can't find unit SALUDOS

$ ppc386 Saludos.pas
Free Pascal Compiler version 1.0.4 [2001/08/31] for i386
Copyright (c) 1993-2000 by Florian Klaempfl
Target OS: Linux for i386
Compiling Saludos.pas
Assembling saludos
18 Lines compiled, 0.0 sec
```

```
$ ppc386 HolaMundo2.pas
Free Pascal Compiler version 1.0.4 [2001/08/31] for i386
Copyright (c) 1993-2000 by Florian Klaempfl
Target OS: Linux for i386
Compiling HolaMundo2.pas
Assembling holamundo2
Linking holamundo2
17 Lines compiled, 0.0 sec

$ ls
holamundo2    HolaMundo2.pas  Saludos.pas
holamundo2.o  saludos.o       saludos.ppu

$ ./holamundo2
>Cómo te llamas? Pepe
Hola Pepe

$ mv Saludos.pas saludos.pas

$ rm *.o *.ppu

$ ls
HolaMundo2.pas  saludos.pas

$ ppc386 HolaMundo2.pas
Free Pascal Compiler version 1.0.4 [2001/08/31] for i386
Copyright (c) 1993-2000 by Florian Klaempfl
Target OS: Linux for i386
Compiling HolaMundo2.pas
Assembling holamundo2
Linking holamundo2
18 Lines compiled, 0.0 sec

$ ls
holamundo2    HolaMundo2.pas  Saludos.pas
holamundo2.o  saludos.o       saludos.ppu

$ ./holamundo2
>Cómo te llamas? Pepe
Hola Pepe
```

En FreePascal para Linux, al contrario que la versión windows de deste mismo compilador, hacer trazas a los programas es algo más complejo, y requiere un programa adicional: GNU Debugger (GDB). Para utilizar este programa con nuestros programas en Pascal, debemos añadir un parámetro adicional en la línea de comandos:

```
ppc386 -g programa.pas
```

Con esto, "incrustamos" el código fuente de nuestro programa en el interior del ejecutable, de forma que si ejecutamos gdb pasando como argumento el nombre del programa, podremos hacerle una traza.

Para saber más acerca de como hacer una traza a nuestros programas escritos en Pascal, puedes consultar la sección sobre GDB, en la página 313.

Otros argumentos interesantes del compilador de FreePascal:

Sin entrar demasiado en las interioridades de FreePascal, creo que estas opciones pueden ayudarte a la hora de compilar una práctica, o de estructurar mejor tus proyectos. Ahí van:

`-FUdirectorio`: Esta opción nos permite guardar las "units", una vez compiladas, en el directorio especificado. Por ejemplo:

```
ppc386 -FUunits unidad.pas
```

Para utilizar units que estén en otro directorio que no sea el actual, utilizamos el parámetro `-Fudirectorio` (nótese que en este caso la "u" es minúscula).

`-So`: Usar esta opción nos permitirá comprobar la compatibilidad de nuestro código con Turbo Pascal 7

`-Un`: Esta opción nos permite llamar los ficheros de unidad (units) con otro nombre distinto al que hemos especificado en el código tras la palabra reservada ünit"

Por último, algunos editores que "se llevan bien" con FreePascal:

`Anjuta` Tal vez, el mejor editor para programadores (o al menos, en opinión del autor de este tema). Soporta resaltado de sintaxis para varios lenguajes, compilar y ejecutar desde el mismo editor, etc.

`Emacs` A pesar de su aparente dificultad de manejo, el modo de Pascal está muy trabajado, permitiendo indentar automáticamente todo el código tan sólo pulsando la tecla tab en cada línea.

`Vim` El autor de este tema sólo ha usado de vim la funcionalidad de resaltado de sintaxis mientras programaba en Pascal. Esta se activa con la opción ":syntax on".

`Scite` Es un editor muy minimalista, y algo lioso de configurar, pero es un editor para programadores muy bueno, cómodo, y sobre todo, compacto (cabe en un disco de 1'44), además de estar disponible tanto para plataformas Win32 como GNU/Linux.

TEMA 19

GNU Fortran

19.1. ¿Fortran?

`Fortran` no es un lenguaje muy popular, ni tiene por qué serlo. Se trata de un lenguaje para un uso específico: cálculo numérico. Fortran no es un lenguaje de propósito general como C, Pascal, Python o Perl y por eso sólo se utiliza en entornos científicos como centros de cálculo.

El lenguaje de programación `Fortran` es importante en entornos científicos. Su nombre es un acrónimo de **For**mula **Tran**slator, ya que su mayor uso era traducir las fórmulas de cálculo matemático al lenguaje de las máquinas. Desde sus principios, el lenguaje Fortran ha tenido una sintaxis muy peculiar, adaptada al uso de tarjetas perforadas. En la actualidad, Fortran se utiliza en asignaturas de Cálculo Numérico en carreras técnicas como Matemáticas, Física y algunas ingenierías.

Utilizaremos aquí el compilador **GNU Fortran**, compatible con la mayoría del lenguaje básico de Fortran 77 y algunas extensiones propias de Fortran 90, suficiente para las prácticas de programación en Fortran 77. En este libro nos mantendremos dentro del estándar ANSI Fortran 77.

Para programación en Fortran 95 existe un proyecto de compilador en marcha en `http://g95.sf.net`, aunque aún se encuentra en estado larval.

19.2. Uso básico del compilador

El compilador de GNU Fortran es muy similar al del GNU C/C++, por lo que los detalles de ambos serán abarcados en el próximo tema. Veamos una vez más el típico ejemplo de "HolaMundo". En el editor que más te guste, escribe el siguiente código Fortran:

Fichero `HolaMundo.for`

```
* Programa básico de HolaMundo

      Program HolaMundo

      write (*,5) 'Hola Mundo'

 5    format (//,2x,a,/)

      stop
      end
```

Ejemplo 19.1: Programa básico de holamundo en Fortran 77, define un formato e imprime un mensaje usándolo. también incluye una línea de comentario.

El comando para invocar al compilador es `g77`, y su sintaxis es muy similar a la del compilador `gcc` (GNU C/C++). Estos compiladores producen la salida por defecto en un ejecutable con el nombre `a.out`, comportamiento que puedes modificar con la opción `-o nombreejecutable`.

```
$ ls
HolaMundo.for

$ g77 HolaMundo.for

$ ls
a.out  HolaMundo.for

$ g77 -o HolaMundo HolaMundo.for

$ ls
HolaMundo  HolaMundo.for

$ ./HolaMundo


  Hola Mundo
```

19.3. Dividir el código

Veamos ahora también como podemos dividir un programa en varios ficheros de código fuente. En el editor escribe los siguientes códigos fuente Fortran y guárdalos como `HolaMundo2.for` y `Saludos.for` respectivamente.

Fichero `HolaMundo2.for`

```
* Ejemplo 2 de Fortran para CILA
* Fichero: HolaMundo2.for

      Program HolaMundo2
      character(10) saludo

      saludo = 'Mundo'

      call saluda (saludo)

      stop
      end
```

Ejemplo 19.2: Segundo "Hola Mundo" en Fortran. Este fichero llama a una función `saluda()` que no está definida en él, sino en otro fichero.

Fichero `Saludos.for`

```
* Ejemplo 2 de Fortran para CILA
* Fichero: saludos.for

      Subroutine saluda(mensaje)
      character(10) mensaje

      print 5, 'Hola ', mensaje

 5    format (//,2x,a,a,/)

      return
      end
```

Ejemplo 19.3: Segundo "Hola Mundo" en Fortran. Este fichero implementa la función `saluda()` que es llamada desde el fichero `HolaMundo2.for`.

Para generar ahora el ejecutable utiliza el comando `g77` dándole ambos ficheros como argumentos. En general, para compilar un programa Fortran escrito en varios ficheros bastará con utilizar el comando de la forma `g77 -o ejecutable fichero1.for ... ficheroN.for`

```
$ ls
HolaMundo2.for  Saludos.for

$ g77 -o HolaMundo2 HolaMundo2.for Saludos.for

$ ls
HolaMundo2  HolaMundo2.for  Saludos.for

$ ./HolaMundo2


  Hola Mundo
```

Si lo prefieres puedes también compilar los ficheros de código fuente por separado para obtener los ficheros de código objeto, y luego enlazarlos al final. Esto resulta útil cuando tienes muchos ficheros de código fuente y sólo estás trabajando en uno de ellos, ya que puede significar un ahorro de tiempo no tener que compilar todos los ficheros cada vez que quieras compilar el programa. En este ejemplo los comandos serían los siguientes:

```
$ g77 -c HolaMundo2.for
$ g77 -c Saludos.for
$ g77 -o HolaMundo2 HolaMundo2.o Saludos.o
```

En los dos primeros comandos, la opción `-c` del compilador le indica que sólo debe generar el código objeto, sin intentar enlazarlo. En el último, el compilador está recibiendo los ficheros de código objeto (y la opción para modificar el nombre del ejecutable resultante) e interpreta que debe enlazarlos. Es importante notar que no podemos utilizar el comando `ld` para enlazar código en Fortran porque faltarían muchos símbolos (mayormente funciones) que `g77` se encarga de poner pero `ld` desconoce.

Si después de haber compilado el programa de esta forma modificas el fichero `Saludos.for` y quieres recompilar el programa, sólo has de recompilar el fichero (o los ficheros) que has modificado y volver a enlazar los ficheros de código objeto:

```
$ g77 -c Saludos.for
$ g77 -o HolaMundo2 HolaMundo2.o Saludos.o
```

En este ejemplo no se nota la ventaja, pero en una práctica de programación en la que estás aprovechando tres ficheros de código fuente de prácticas anteriores y escribiendo otros tres ficheros de código fuente nuevos, según la máquina en la que trabajes puede que no te apetezca tener que compilar los seis ficheros cada vez que modificas uno. Si a esto le sumas el uso de GNU Make (que estudiaremos más adelante) el proceso de compilación y recompilación resulta mucho más cómodo.

19.4. Mezclar Fortran con C

Con lo que has visto en este tema sabes más que suficiente para hacer prácticas de programación en Fortran 77, pero ahora vamos a rizar el rizo un poco. Si sabes algo (un poco) de programación en C no te resultará difícil entender lo siguiente, en caso contrario deja este apartado para cuando hayas terminado con los temas de GNU C/C++ y GNU Make.

Fortran es un lenguaje de cálculo numérico, e intentar programar algo que no sea cálculo numérico en Fortran puede ser un suplicio. Un ejemplo sencillo de esto es un menú, que permanezca preguntando opciones hasta que el usuario decida salir

del programa. Es posible programar un menú en Fortran, pero no resulta tan efectivo como en otros lenguajes como C o Pascal.

C es un lenguaje de propósito general, lo que significa que puedes programar en C casi cualquier cosa que te propongas. Sin embargo programar cálculo numérico en C puede no ser tan cómodo como hacerlo en Fortran. Entonces, unamos ambos lenguajes en un mismo programa, aprovechando lo mejor de cada uno.

Esta curiosa mezcla es posible con los compiladores de GNU, porque de hecho el compilador de GNU Fortran está basado en el de GNU C/C++. Sin embargo hay una serie de consideraciones que debes tener en cuenta, y es conveniente que sepas programar algo en C porque hay que mirarlo desde el punto de vista del lenguaje C.

- **En Fortran las funciones reciben siempre punteros**. Esto implica que si modificas el valor de un parámetro dentro de una función en Fortran, ese cambio seguirá siendo efectivo tras terminar la función.
- **En Fortran los vectores son también punteros**. Cuando declaras en Fortran un vector puedes especificar el rango de índices que será válido dentro del vector, por ejemplo `integer a(-2:8)` define un vector de enteros que puede ser indexado desde -2 hasta 8 ambos inclusive, de forma que `a(-2)` y `a(8)` son los extremos del vector. Esto se maneja internamente como un puntero a entero a partir del cual hay 11 posiciones reservadas. Es decir, su equivalente en C sería `int a[11]` y `a(-2)` sería `a[0]`. Incluso programando sólo en Fortran es importante entender esto.
- **Las funciónes de Fortran se renombran en el código objeto**. Esto es, si has definido una función en Fortran llamada `spline` para llamarla desde C debes hacerlo con el nombre `spline_` (añadiendo dos caracteres de subrayado). Además debes pasarle los parámetros como punteros.
- **Las funciones llamadas desde Fortran también se renombran**. Para llamar a la función `menu` desde el código Fortran hay que definirla en C con el nombre `menu_`.
- **Los ficheros de código fuente se compilan sin enlazar**. Para compilar un fichero de código fuente de Fortran utiliza el comando `g77 -c fichero.for` y para compilar un fichero de código fuente de C utiliza el comando `gcc -c fichero.for`.
- **Los ficheros de código objeto se enlazan con g77**. Cuando hayas compilado todos los fuentes utiliza el compilador g77 (no el gcc) para enlazarlo y producir el ejecutable.

Veamos un ejemplo sencillo de cómo usar esto. Los siguientes ficheros, dos de Fortran y uno de C, implementan un sencillo algoritmo de ordenación de datos (selección directa).

Fichero `ordena.for`

```
      program ordena

      ! Ejemplo de programación conjunta Fortran y C
      ! CILA -- Curso de Introducción a Linux para Alumnos

      parameter (n = 20)
      real a(1:n)
      integer i

      call menu (a, n)

      write (*,*) 'Al abandonar el menú el vector contiene:'
      write (*,10) ('a(', i, ') = ', a(i), i = 1, n)
 10   format (2x,a,i2,a,f10.3)

      stop
      end
```

Ejemplo 19.4: Declara las variables, llama al menú escrito en C y finalmente muestra el contenido del vector después de terminar el menú. Nótese que el vector `a` está indexado desde 1 hasta `n`.

Fichero `menu.c`

```
#include <stdio.h>

void mysort_ (float *a, int *m);

void menu_ (float *a, int *n)
{
    int i, m, op;
    char trash[51];
    m  =  0;
    op = -1;

    printf ("\n\tEjemplo de programación conjunta Fortran y C\n");
    printf ("\n\tCILA -- Curso de Introducción a Linux para Alumnos\n");
    while (op)
    {
        printf ("\nMenú:\n");
        printf (" 1. Vaciar el vector\n");
        printf (" 2. Añadir un elemento al vector\n");
        printf (" 3. Ordenar el vector\n");
        printf (" 4. Mostrar el vector\n");
        printf (" 0. Salir del menú\n");
        printf ("Elije una opción (0/1/2/3/4): ");
        while (scanf ("%d", &op) != 1)
        {
            scanf ("%50s", trash);
            printf ("La opción %s no es válida\n", trash);
            printf ("Elije una opción (0/1/2/3/4): ");
        } /* while */
        switch (op)
        {
            case 1:
                for (i = 0; i < m; i++)
                    a[i] = 0;
                m = 0;
                printf ("El vector ha sido vaciado\n");
                break;
            case 2:
                printf ("Introduce un número real: ");
                while (scanf ("%f", a+m) != 1)
                {
                    scanf ("%50s", trash);
                    printf ("%s no es un número real\n", trash);
                    printf ("Introduce un número real: ");
                } /* while */
                m++;
                break;
            case 3:
                mysort_ (a, &m);
                printf ("El vector ha sido ordenado\n");
                break;
            case 4:
                printf ("\n\tEl vector contiene %d números\n", m);
                for (i = 0; i < m; i++)
                    printf ("\ta[%d] = %f\n", i, a[i]);
                break;
            case 0:
                printf ("Menú terminado\n");
                break;
```

```
            default:
                printf ("La opción %d no es válida\n", op);
        } /* switch */
    } /* while */
} /* menu */
```

Ejemplo 19.5: Implementa el menú que permite operar con el vector y llamar a la función `mysort` programada en Fortran. Además protege al programa de entradas de datos erróneas o maliciosas. Nótese qe el vector `a` es de tamaño `*n`, pero sólo se utilizan las `m` primeras posiciones, indexadas desde 0 hasta `m - 1`.

Fichero `mysort.for`

```
      subroutine mysort (a, m)
      real a(1:m), aux
      integer m, i, j
      do 20 i = 1, m - 1
        do 10 j = i + 1, m
          if (a(j) .lt. a(i)) then
            aux  = a(j)
            a(j) = a(i)
            a(i) = aux
          endif
 10     continue
 20   continue
      return
      end
```

Ejemplo 19.6: Implementa en Fortran el método de ordenación por selección directa. Nótese que el tamaño del vector se recibe como parámetro y sólo se utilizan las `m` primeras posiciones, indexadas desde `1` hasta `m`.

Para compilar este programa hazlo fichero a fichero, generando primero los ficheros de código objeto utilizando g77 o gcc según esté cada fichero de código fuente escrito en Fortran o en C. Luego enlazalos todos con g77.

```
$ g77 -c ordena.for
$ gcc -c menu.c
$ g77 -c mysort.for
$ g77 -o ordena ordena.o menu.o mysort.o
```

Con la ayuda de GNU Make el proceso de recompilado se convierte en algo casi automático y bastante más cómodo. Para saber más sobre GNU Make ve a la página 307.

TEMA 20

GNU C/C++

En este capítulo tomaremos en consideración la programación en lenguaje C/C++ bajo el sistema operativo GNU/Linux. Todos los lenguajes de programación compilados pasan en menor o mayor medida por las mismas fases, por lo que mucho de lo que estudiaremos se puede aplicar a otros lenguajes y problemas.

20.1. Compilado y enlazado

Todos los leguajes de programación compilados, y el C no es una excepción, requieren de dos fases para generar el binario (o ejecutable) de un programa:

Compilar Traduce cada archivo fuente en C (`*.c`) del programa a lenguaje máquina, almacenando la traducción en los archivos de código objeto (`*.o`). Para ello usamos el GCC (GNU C Compiler) a través del comando `gcc`.

Enlazar Une todos los archivos de código objeto generados en la etapa anterior para fundirlos en el ejecutable de la aplicación. Para esta tarea se utiliza el comando `ld`.

Por fortuna para nosotros `gcc` llama por defecto a `ld`, ahorrándonos tener que realizar los dos pasos a mano. Como ejemplo tomemos el siguiente programa:

Fichero `holamundo.c`

```
/* holamundo.c .- Sencillo ejemplo de programa en C */

#include <stdio.h>

int
main (void)
{
  puts ("El que a buen árbol se arrima, buena sombra le cobija.\n");
  return 0;
}
```

Ejemplo 20.1: Ejemplo de un sencillo programa en C para compilar y enlazar. El programa muestra un mensaje por la salida estándar y termina.

Y ejecutemos el comando:

```
$ gcc holamundo.c
```

Si listásemos el contenido del directorio actual con el comando `ls` veríamos un nuevo archivo con el nombre de `a.out`. Ese archivo contiene nuestro programa y puede ser fácilmente ejecutado.

```
$ ./a.out
El que a buen árbol se arrima, buena sombra le cobija.
```

Es probable que pocos programadores consideren que `a.out` sea un nombre apropiado para el ejecutable de su aplicación. Con el fin de cambiar dicho nombre se puede utilizar la opción `-o nombre_ejecutable`.

```
$ gcc -o holamundo holamundo.c
$ ./holamundo
El que a buen árbol se arrima, buena sombra le cobija.
```

Como podemos apreciar, el ejecutable de nuestra aplicación ahora se denomina `holamundo`.

Una práctica muy habitual en programación es dividir nuestro código en varios archivos, cada uno especializado en un tarea determinada. Supongamos que disponemos del siguiente programa:

Fichero `holamain.c`

```
/* holamain.c .- Ejemplo de programa en varios archivos.
 *         Punto de entrada a la aplicación.
 */

#include "holafunc.h"

int
main (void)
{
  holafunc ("buena sombra le cobija");
  return 0;
}
```

Ejemplo 20.2: Definición de la función `main()`; punto de entrada al ejemplo de un sencillo programa en C en varios archivos. La función llama a `holafunc()`, definida en `holafunc.c`, para que muestre un mensaje por la salida estándar.

Fichero `holafunc.h`

```
/* holafunc.h .- Ejemplo de programa en varios archivos.
 *         Declaración de holafunc().
 */

int holafunc (const char *str);
```

Ejemplo 20.3: Declaración de la función `holafunc()` encargada de mostrar un mensaje por la salida estándar.

Fichero `holafunc.c`

```
/* holafunc.c .- Ejemplo de programa en varios archivos.
 *         Definición de holafunc().
 */

#include <stdio.h>
#include "holafunc.h"

int
holafunc (const char *str)
{
  return printf ("El que a buen árbol se arrima, %s.\n\n", str);
}
```

Ejemplo 20.4: Definición de la función `holafunc()` encargada de mostrar un mensaje por la salida estándar.

Para generar nuestro programa sólo debemos listar los archivos que lo forman en la línea de comandos del `gcc`.

```
$ gcc -o holamundo holamain.c holafunc.c
$ ./holamundo
El que a buen árbol se arrima, buena sombra le cobija.
```

20.2. Enlazado con bibliotecas externas

Como hemos comentado anteriormente, `gcc` compila cada unos de los archivos de código fuente pasados en la línea de comandos. El resultado es un archivo de código objeto por cada archivo de código fuente. Dichos archivos de código objeto son enlazados por `ld` para generar el ejecutable de nuestra aplicación. Cuando `gcc` llama a `ld` no solo se están enlazando a nuestro programa el código objeto de `holamain.c` y el de `holafunc.c`. El compilador sabe que para que nuestro programa funcione es necesario que esté enlazado a una serie de bibliotecas estándar del sistema, así que las incluye automáticamente.

Una de esas bibliotecas es la Biblioteca GNU Estándar de C (GLIBC, GNU C Library), comunmente conocida como `libc`. Podemos acceder a la documentación de dicha biblioteca a través del comando:

```
$ info libc
```

Funciones como `fopen()`, `malloc()`, `printf()` y en general todas las del C estándar y muchas más se definen en la `libc`. Evidentemente en nuestro sistema existen bibliotecas para toda clase de tareas cuya documentación se puede obtener recurriendo a la ayuda del sistema. Sin embargo, la mayor parte de esas bibliotecas no se enlazan automáticamente, así que nos queda la duda de cómo resolver este problema. Por ejemplo vamos a modificar `holafunc.c`:

Fichero `holafuncm.c`

```
/* holafuncm.c .- Ejemplo de programa que require la biblioteca de
 *        funciones matemáticas. Definición de holafunc().
 */

#include <math.h>
#include <stdio.h>
#include "holafunc.h"

int
holafunc (const char *str)
{
  float a, b;

  printf ("El que a buen árbol se arrima, %s,\n", str);

  a = 0.5;
  b = cos (a);
  return printf ("y el coseno de %f es %f. \n\n", a, b);
}
```

Ejemplo 20.5: Definición de la función `holafunc()` encargada de mostrar un mensaje por la salida estándar y de calcular el coseno de 0.5 radianes.

Si compilamos `holamundo` veremos el siguiente mensaje de error:

```
$ gcc -o holamundo holamain.c holafuncm.c
/tmp/ccKSzM6q.o: In function ‘holafunc’:
/tmp/ccKSzM6q.o(.text+0x30): undefined reference to ‘cos’
collect2: ld returned 1 exit status
```

Ese mensaje nos indica que la función `cos()` llamada desde `holafunc()` no está definida puesto que se encuentra en una biblioteca que no está siendo enlazada al programa. Para resolver el problema se emplea la opción `-lnombre_biblioteca`, con la que se indica la biblioteca adicional que debe ser enlazada al programa.

```
$ gcc -lm -o holamundo holamain.c holafuncm.c
$ ./holamundo
El que a buen árbol se arrima, buena sombra lo cobija,
y el coseno de 0.500000 es 0.877583.
```

Al especificar `-lm` se enlaza la biblioteca `libm` (como vemos no hace falta poner el prefijo `lib` cuando se indica el nombre de la biblioteca) que contiene la definición de `cos()`. Es importante destacar que el enlazador sólo busca bibliotecas en una serie de directorios estándar de nuestro sistema. Por ejemplo, `libm` se encuentra en `/usr/lib/` que es un directorio estándar. Si deseamos enlazar bibliotecas situadas en otros directorios, como por ejemplo el directorio actual, debemos usar la opción `-Lruta_biblioteca`. Por ejemplo, el siguiente comando enlaza a nuestro programa una biblioteca de nombre `libpropia` que se encuentra en el directorio actual:

```
$ gcc -L. -lm -lpropia -o holamundo holamain.c holafuncm.c
```

20.3. Separar compilación y enlazado

En caso de que prefiramos separar la etapa de compilación y enlazado, se utiliza la opción `-c` con el compilador `gcc`. De esa manera le informamos de que sólo queremos que genere el código objeto. Probemos a generar el código objeto para cada archivo:

```
$ gcc -c holamain.c holafuncm.c
```

Ahora tenemos dos nuevos archivos denominados `holamain.o` y `holafuncm.o` que se corresponden con el código objeto de `holamain.c` y `holafuncm.c` respectivamente.

Para enlazar, sólo debemos especificar los archivos de código objeto en la línea de comandos del `gcc`.

```
$ gcc -lm -o holamundo holamain.o holafuncm.o
```

La parte positiva de esto es que ahora podemos enlazar en nuestra aplicación código objeto generado por otros lenguajes, como `Fortran` (véase 19.4) o `Pascal`. Además, nos permite compilar sólo aquellos archivos de código fuente que han cambiado desde la última compilación. Esto no parece importante cuando trabajamos en proyectos pequeños con apenas un par de archivos. Pero es fundamental a la hora de desarrollar programas de mayor envergadura.

También es posible utilizar directamente `ld`, en lugar de ejecutar `gcc` y que éste llame al primero. El problema es que debemos indicarle a mano al `ld` que enlace a nuestro programa las bibliotecas estándar del sistema, puesto que él no lo hace automáticamente. Ya que `gcc` hace dicho trabajo por nosotros, evitaremos entrar en más detalles.

20.4. Bibliotecas de enlace dinámico

Un uso interesante del `ld` es para generar nuestras propias bibliotecas de enlace dinámico. Ahora que disponemos de un archivo `holafuncm.o` ejecutemos lo siguiente:

```
$ ld -shared -lm -o libholafunc.so holafuncm.o
```

Aunque también es posible, y recomendable, invocar al `gcc` para que realice el mismo trabajo.

```
$ gcc -shared -lm -o libholafunc.so holafuncm.o
```

En ambos casos, si listamos el contenido del directorio observaremos un nuevo archivo denominado `libholafunc.so` que es nuestra biblioteca de enlace dinámico. Dicha biblioteca puede ser utilizada por cualquier aplicación del sistema, si la enlazamos como hemos aprendido.

```
$ gcc -L. -lholafunc -o holamundo holamain.o
```

A diferencia de ejemplos anteriores, nuestro programa no funcionará si no disponemos de `libholafunc.so`. Sin embargo, cualquier programa futuro podrá utilizar las funciones de nuestra biblioteca.

El inconveniente de crear bibliotecas de enlace dinámico es que hay que instalarlas en algún directorio donde el enlazador dinámico (`ld.so`) sepa que hay bibliotecas (p.ej. `/usr/lib`), pero la labor de instalar o desinstalar bibliotecas en el sistema, así como la de configurar `ld.so` para que las busque es del `root`. Por ello, si ejecutáramos ahora nuestro programa éste no funcionaría, puesto que sería incapaz de encontrar `libholafunc.so`. Para que esto no sea así debemos definir la variable de entorno $LD_LIBRARY_PATH con la ruta de nuestra biblioteca.

```
$ export LD_LIBRARY_PATH=./
$ ./holamundo
El que a buen árbol se arrima, buena sombra le cobija,
y el coseno de 0.500000 es 0.877583.
```

La forma en la que hemos generado nuestra biblioteca de enlace dinámico no suele dar problemas en sistemas GNU/Linux bajo plataforma Intel x86. Sin embargo, no es el procedimiento recomendado. Lo ideal, con el fin de garantizar la compatibilidad entre plataformas, es que compilemos el código fuente de nuestra biblioteca de enlace dinámico con la opción `-fPIC`. Dicha opción fuerza al compilador a generar código independiente de la posición. De esa manera la biblioteca puede ser cargada en cualquier punto del espacio de direcciones de la memoria de las aplicaciones que la utilizan. Por tanto, el procedimiento estándar para generar nuestro programa queda de la siguiente manera:

```
$ gcc -c holamain.c
$ gcc -fPIC -c holafuncm.c
$ gcc -shared -lm -o libholafunc.so holafuncm.o
$ gcc -L. -lholafunc -o holamundo holamain.o
```

O, en caso de no querer conservar los archivos de código objeto, podemos hacerlo de la siguiente manera:

```
$ gcc -fPIC -shared -lm -o libholafunc.so holafuncm.c
$ gcc -L. -lholafunc -o holamundo holamain.c
```

20.5. ¿Y que pasa con el C++?

A la hora de compilar código en C++ (habitualmente con extensiones `*.C`, `*.cc`, `*.cpp`, `*.c++`, `*.cp`, `*.cxx`) se utiliza el comando `g++`. Básicamente se encarga de ejecutar el `gcc` habilitando el C++ como lenguaje por defecto, y añadiendo las bibliotecas estándar del C++ en la fase de enlazado. Por tanto, todo lo explicado para `gcc` es aplicable para el `g++`. Y si no, veamos el siguiente código de ejemplo:

Fichero `holamain.cc`

```
// holamain.cc .- Ejemplo de programa en C++.
//              Punto de entrada a la aplicación.

#include "saludo.h"

int
main (void)
{
  saludo hola ("El que a buen árbol se arrima, buena sombra le cobija.\n\n");

  hola.print ();
  return 0;
}
```

Ejemplo 20.6: Definición de la función `main()`; punto de entrada al ejemplo de un sencillo programa en C++. La función crea un objeto de clase `saludo`, definida en `saludo.cc`, para que muestre un mensaje por la salida estándar.

Fichero `saludo.h`

```
// saludo.h .- Ejemplo de programa en C++.
//             Declaración de la clase saludo.

class saludo
{
public:
  saludo (const char *str);
   ~saludo ();

  void print (void);

private:
  const char *msg;
};
```

Ejemplo 20.7: Declaración de la clase `saludo` encargada de mostrar un mensaje por la salida estándar.

Fichero `saludo.cc`

```
// saludo.cc .- Ejemplo de programa en C++.
//             Definición de los miembros de la clase saludo.

#include <iostream.h>
#include "saludo.h"

saludo::saludo (const char *str):
msg (str)
{
}

saludo::~saludo ()
{
}

void
saludo::print (void)
{
  cout << msg;
}
```

Ejemplo 20.8: Definición de la clase `saludo` encargada de mostrar un mensaje por la salida estándar.

El cual se genera de forma semejante al caso en el que trabajamos en lenguaje C.

```
$ g++ -o holamundo holamain.cc saludo.cc
$ ./holamundo
El que a buen árbol se arrima, buena sombra le cobija.
```

Con todo esto se puede decir que ya estamos hechos unos *C/C++ Linux Programmers*. Por lo que sólo queda navegar un poco, escoger el *proyecto de software libre* que más nos guste, o disguste, y ponernos a colaborar.

TEMA 21

GNU Make

A la hora de compilar un programa resulta evidente que utilizar la línea de comandos es molesto, incluso cuando se dispone de unos pocos archivos de código fuente. Por ello es conveniente utilizar el programa GNU Make para ayudarnos en dicho proceso.

GNU Make sabe que tareas debe realizar a través de una serie de reglas descritas en un archivo de texto de nombre `Makefile`, habitualmente situado en el directorio de nuestro código fuente. El comando `make`, al ser invocado desde dicho directorio, ejecuta las reglas del archivo `Makefile`, recompilando sólo las partes que han sido modificadas desde la última compilación, y enlazando los archivos de código objeto para generar el ejecutable.

GNU Make puede usarse con cualquier lenguaje de programación cuyas tareas puedan lanzarse mediante la línea de comandos. En realidad, puede usarse para manejar cualquier conjunto de archivos dependientes entre sí, en los cuales algunos deban actualizarse mientras que otros no.

21.1. Makefile sencillo

Tomaremos como ejemplo uno de los programas del tema de GNU C/C++ (véase tema 20). Concretamente, utilizaremos el programa compilado y ejecutado en la página 304. Un ejemplo de `Makefile`, para compilar y enlazar dicho programa, puede ser el siguiente:

Fichero `makefile1.mak`

```
# Makefile .- Ejemplo sencillo de Makefile.

holamundo: holamain.o holafuncm.o
        gcc -lm -o holamundo holamain.o holafuncm.o

holamain.o: holamain.c holafunc.h
        gcc -c holamain.c

holafuncm.o: holafuncm.c holafunc.h
        gcc -c holafuncm.c

clean:
        rm holamundo \
           holamain.o holafuncm.o
```

Ejemplo 21.1: Makefile sencillo para compilar y enlazar el programa ejecutado en la página 304.

Las líneas que comienzan con # son comentarios, no se ejecutan. El resto son reglas que definen como generar el programa `holamundo`. Cada regla empieza con un nombre seguido de `:`. A continuación de cada regla pueden haber una o más líneas, iniciadas con tabulador, que especifican que comandos deben ser ejecutados para cumplir con dichas reglas.

Si guardamos `makefile1.mak` con el nombre `Makefile` en el directorio donde está el código fuente del programa, podemos generar y probar nuestro programa de la siguiente manera:

```
$ make
$ ./holamundo
El que a buen árbol se arrima, buena sombra lo cobija,
y el coseno de 0.500000 es 0.877583.
```

También es posible utilizar la opción `-f nombre_makefile` para especificar un nombre de archivo diferente a `Makefile`. Por ejemplo:

```
$ make -f makefile1.mak
$ ./holamundo
El que a buen árbol se arrima, buena sombra lo cobija,
y el coseno de 0.500000 es 0.877583.
```

Como se puede apreciar, `make` simplifica notablemente el proceso de compilación y enlazado de nuestra aplicación.

21.2. Reglas en los Makefile

Las reglas descritas en los `Makefile` siempre tienen el siguiente formato:

```
destino: requisito ...
        comando
        ...
```

Destino Es el nombre de un archivo a crear. Normalmente un ejecutable o un archivo de código objeto (`*.o`). También puede ser el nombre de una tarea a realizar. Por ejemplo, es habitual utilizar *clean* como el *destino* de una regla que ejecuta los comandos necesarios para eliminar los archivos de código objeto y otros archivos de uso intermedio.

Requisito Es el nombre de un archivo del cual depende el *destino* a crear. Un destino suele depender de varios archivos requisito. Cuando el destino define el nombre de una tarea, no se definen requisitos.

Comando Define una acción a realizar para generar un *destino*. Puesto que la creación de un destino puede requerir varios comandos, estos deben ser indicados en línea sucesivas, cada una de las cuales debe comenzar con un tabulador (no sirven los espacios). Los comandos son pasados al intérprete de comandos, el cual se encarga de interpretarlos y ejecutarlos.

En el ejemplo anterior vemos un elemento adicional que no hemos comentado, \ es el caracter de continuación de línea. Colocado al final de una línea, une dos líneas consecutivas como si fueran una sola. Se utiliza para facilitar la lectura.

Cuando ejecutamos `make` es necesario indicar el *destino* que queremos que sea generado. De lo contrario, se generará el *destino* por defecto, que es el definido en la primera regla del `Makefile`. Por ejemplo, si utilizamos el `Makefile` anterior y ejecutamos `make` de la siguiente manera:

```
$ make clean
rm holamundo \
   holamain.o holafuncm.o
```

Tanto el ejecutable como los archivos de código objeto de nuestra aplicación serán borrados. Sin embargo, si lo hacemos así:

```
$ make
gcc -c holamain.c
gcc -c holafuncm.c
gcc -lm -o holamundo holamain.o holafuncm.o
```

El comando `make` toma `holamundo` como destino por defecto y ejecuta los comandos adecuados para generarlo conforme a las reglas descritas por el `Makefile`. Evidentemente, se consigue el mismo efecto si ejecutamos `make holamundo`.

Una de las funcionalidades más importantes de `make` es que evita generar los *destinos* ya existentes, o cuyos archivos de *requerimiento* no hayan cambiado. Para ello `make` estudia las dependencias entre *destinos*. Por ejemplo, si ejecutamos `make` con nuestro sencillo `Makefile` de pruebas, el programa detectará, a partir de las reglas, una dependencia entre el ejecutable `holamundo` y los archivos de código objeto `holamain.o` y `holafuncm.o`. Y que estos a su vez dependen de `holamain.c` y `holafunc.h`, y `holafuncm.c` y `holafunc.h` respectivamente. El programa `make` es capaz de detectar si alguno de los archivos de código fuente (`holamain.c`, `holafuncm.c` u `holafunc.h`) fue modificado después de la última ejecución. En caso de ser así, `make` invoca los *comandos* de la regla o reglas adecuadas para garantizar que `holamain.o` y `holafuncm.o` existen y están actualizados con respecto a los archivos de código fuente de los que dependen. Hecho esto, pasa a verificar si el ejecutable `holamundo` existe y está actualizado con respecto a los archivos de código objeto de los que depende. De no ser así, se ejecutan los *comandos* definidos en su regla con el fin de actualizar el ejecutable. Por lo tanto, GNU Make garantiza que tengamos nuestro binario actualizado empleando el mínimo número de pasos posible.

21.3. Macros o variables

Las macros nos permiten definir constantes que utilizamos frecuentemente en nuestro archivo. La definición de una macro tiene la siguiente forma:

```
MACRO1 = cadena de texto
MACRO2 = otra cadena de texto
```

Las macros pueden ser referenciadas indicando el nombre entre paréntesis (`()`) o llaves (`{}`) precedidas por el signo del dolar (`$`). Por ejemplo, las macros anteriores serían referenciadas como:

```
$(MACRO1)
${MACRO2}
```

Algunas definiciones de macros válidas podrían ser:

```
LIBS = -lm
OBJECTS = holamain.o holafuncm.o
INCLUDES = holafunc.h
DEBUG_FLAG =            # poner -g para depurar
```

Los nombres de las macros pueden usar cualquier combinación de letras, números y subrayados. Además, por convenio, suelen definirse en mayúsculas. Los valores pueden ser nulos como en la definición anterior de `DEBUG_FLAG`, que nos muestra como un comentario puede ir a continuación de una definición. La propia definición de una macro puede contener referencias a otras macros definidas anterior o posteriormente.

Una características interesante de GNU Make es la posibilidad de definir macros desde la línea de comandos. Por ejemplo, podemos definir un valor para `DEBUG_FLAG`, que de otra forma tendría un valor nulo.

```
 $ make DEBUG_FLAG=-g
```

Las definiciones que incluyan espacios deben ir encerradas entre comillas dobles o simples, para que el intérprete de comandos se las entregue a `make` como un único argumento. Por ejemplo:

```
$ make "LIBS= -lm -lX11"
```

Las variables de entorno están disponibles, durante la ejecución de `make`, como cualquier otra macro definida en el `Makefile`.

Variable	Contenido
`@`	El nombre del archivo de *destino* de la regla
`<`	El nombre del primer *requisito*
`?`	Los nombres de todos los *requisitos* que son más nuevos que el *destino*, con espacios entre ellos
`^`	Los nombres de todos los *requisitos*

Tabla 21.1: Macros internas especiales de GNU Make

21.3.1. Macros especiales

GNU Make define algunas macros internas para simplificar nuestro trabajo. Podemos invocar `make` con la opción `-p` para obtener una lista de todas las macros, reglas y destinos que intervienen en la ejecución de `make`.

Además, hay unas pocas macros especiales que se definen en cada regla. Las más útiles a la hora de crear nuestro `Makefile` están descritas en el Cuadro 21.1.

Sabiendo todo esto podemos hacer uso de las macros para simplificar nuestro `Makefile` de ejemplo:

Fichero `makefile2.mak`

```
# Makefile .- Ejemplo de Makefile con macros.

LINK = $(CC)
CFLAGS += -g -Wall
LDFLAGS += -lm


holamundo: holamain.o holafuncm.o
        $(LINK) $(LDFLAGS) -o $@ $^

holamain.o: holamain.c holafunc.h
        $(CC) $(CFLAGS) -c $<

holafuncm.o: holafuncm.c holafunc.h
        $(CC) $(CFLAGS) -c $<

clean:
        rm holamundo \
           holamain.o holafuncm.o
```

Ejemplo 21.2: Makefile para compilar y enlazar el programa ejecutado en la página 304. Se han utilizado las macros de `make` para simplificar y flexibilizar las reglas descritas en el archivo.

Las primeras líneas se utilizan para definir las macros que serán utilizadas en el resto de nuestro programa. Por ejemplo, definimos en `LINK` el enlazador a utilizar, que será el definido internamente por `make` en `CC`; habitualmente el `gcc`. Mientras que las variables `CFLAGS` y `LDFLAGS` especifican las opciones para el compilador y el enlazador respectivamente. En nuestro caso, indicamos con `-lm` que queremos enlazar la biblioteca libm, con `-g` que deseamos incluir el código de depuración en el ejecutable de nuestro programa, y con `-Wall` que el compilador debe avisarnos a la más mínima sospecha de un posible error en el programa.

Al utilizar el operador `+=` durante la asignación, indicamos que los nuevos valores deben añadirse a los que ya tuvieran dichas macros previamente. Es decir, indicamos algo similar a:

```
CFLAGS = $(CFLAGS) -g -c
LDFLAGS = $(LDFLAGS) -lm
```

Solo que expresarlo de esa manera no está permitido. Los valores previos de las variables pueden haber sido definidos internamente por `make`, especificados desde la línea de comandos durante la invocación del mismo, o a través de la definición de variables de entorno del mismo nombre desde el intérprete de comandos.

21.4. Reglas implícitas

Ciertas formas de generar un *destino* son utilizadas frecuentemente. Por ejemplo, la forma de obtener una archivo de código objeto desde un archivo de código fuente usando un compilador de C como `gcc`.

Las reglas implícitas evitan tener que especificar de forma detallada este tipo de dependencias cuando tienen que ser usadas. Por ejemplo, habitualmente el nombre los archivos de código fuente en C termina en `.c`, mientras que el nombre de los archivos de código objeto termina en `.o`. Se puede crear una regla implícita que indique a `make` como compilar programas en C cuando detecta dichos sufijos en los nombres de los archivos.

Un procedimiento para definir reglas implícitas es con la ayuda de reglas de sufijo. Por ejemplo, la siguiente regla describe como generar un archivos de código objeto (`*.o`) a partir de un archivo de código fuente en C (`*.c`):

```
.c.o:
        $(CC) $(CFLAGS) -c $<
```

Sin embargo, en la actualidad se tiende a utilizar reglas de patrones por ser mucho más flexibles.

```
%.o: %.c
        $(CC) $(CFLAGS) -c $<
```

Como podemos apreciar, en las reglas de patrones el caracter`%` hace de comodín.

Con todo esto, nuevamente podemos simplificar nuestro `Makefile`:

Fichero `makefile3.mak`

```
# Makefile .- Ejemplo de Makefile con reglas implícitas.

LINK = $(CC)
CFLAGS += -g -Wall
LDFLAGS += -lm

INCLUDES = holafunc.h
OBJECTS = holamain.o holafuncm.o


holamundo: $(OBJECTS)
        $(LINK) $(LDFLAGS) -o $@ $^

%.o: %.c $(INCLUDES)
        $(CC) $(CFLAGS) -c $<

clean:
        rm holamundo \
           holamain.o holafuncm.o
```

Ejemplo 21.3: Makefile para compilar y enlazar el programa ejecutado en la página 304. Se han utilizado reglas implícitas para simplificar el archivo y hacerlo más general.

Para ayudarnos a la hora de crear nuestro `Makefile`, GNU Make incluye una serie de reglas implícitas definidas internamente. Estas definiciones pueden consultarse en la documentación del programa, ejecutando `info make`, o con ayuda de la opción `-p` del mismo. En cualquier caso, las reglas más comunes, como la de compilar archivos de código fuente en C, están ya definidas, por lo que nuestro `Makefile` puede simplificarse aún más.

Fichero `makefile4.mak`

```
# Makefile .- Ejemplo de Makefile con reglas implícitas internas.

LINK = $(CC)
CFLAGS += -g -Wall
```

```
LDFLAGS += -lm

INCLUDES = holafunc.h
OBJECTS = holamain.o holafuncm.o


holamundo: $(OBJECTS)
        $(LINK) $(LDFLAGS) -o $@ $^

clean:
        rm holamundo \
           holamain.o holafuncm.o
```

Ejemplo 21.4: Makefile para compilar y enlazar el programa ejecutado en la página 304. Se han utilizado reglas implícitas internas para simplificar el archivo.

GNU Make es un programa muy potente que no permite extender nuestro `Makefile` con nuevos *destinos* que se encarguen de tareas como: generar la documentación de ayuda, que generar bibliotecas de enlace dinámico con las funciones de uso más frecuente, que generar otros ejecutables que formen parte de la aplicación, e incluso empaquetar nuestro programa y lo dejarlo listo para su instalación en cualquier sistema.

TEMA 22

Depuradores

22.1. Depurando la aplicación

Se suele decir que el 10 % del tiempo de desarrollo de un programa se dedica a la codificación y el 90 % restante a la depuración. Al margen de que sea cierto o no la verdad es que es de vital importancia disponer de las herramientas adecuadas para corregir los errores del software en un tiempo razonable.

La primera recomendación es dejar que `lclint` analice nuestro código para que busque y notifique cualquier inconsistencia. Es importante destacar que dicho programa es mucho más potente detectando posibles errores que el analizador sintáctico del `gcc` debido a que el compilador asume que ya hemos pasado nuestro código por un programa como `lclint`. Esa suposición permite al `gcc` realizar ciertas optimizaciones que mejoran su velocidad de compilación.

```
$ lclint main.c holafunc.c
```

Si disponemos de nuestro programa perfectamente compilado y observamos que presenta algún error del que no sabemos determinar su origen, significa que ha llegado la hora utilizar el depurador. GNU/Linux dispone del GNU Debugger bajo el comando `gdb`. Para usarlo sólo debemos ejecutarlo especificando el nombre del programa en la línea de comandos; y haber compilado nuestro programa con la opción `-g`.

```
$ gdb holamundo
GNU gdb 2002-04-01-cvs
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-linux"...
(gdb)
```

En ese momento tendremos al `gdb` esperando alguno de los comandos de depuración. A continuación disponemos de una lista de los comandos más básicos.

`break` Sitúa un punto de ruptura en la línea o función indicada como argumento.

`continue` Continúa la ejecución de un programa que está siendo depurado y se encuentra detenido en un punto de ruptura.

`display exp` Muestra el valor de la expresión `exp` cada vez que el programa se detiene.

`help` Lista las clases de comandos disponibles. Si el comando va seguido por un nombre de clase se listan los comandos de dicha clase. Si va seguido por un nombre de comando se muestra una ayuda completa del comando indicado.

`list` Lista una línea o función especificada. Si el comando va seguido del nombre de una función, el comando muestra dicha función. Si va seguido de un número de línea, muestra esa línea. En programas de varios archivos se puede utilizar `nombre_archivo:nombre_funcion` o `nombre_archivo:numero_línea` para listar los contenidos de un archivo particular. Si no se especifican argumentos se lista desde la última línea mostrada; y si se especifican dos números de línea separados por una coma, se muestran las líneas comprendidas en el intervalo.

`next` Ejecución paso a paso pero ignorando las llamadas a funciones.

`print exp` Muestra el valor de la expresión `exp` en el punto actual.

`quit` Salir del `gdb`.

`run` Inicia la ejecución del programa. Los argumentos del comando son los argumentos que se le pasan al programa.

`step` Ejecución paso a paso incluso de las funciones llamadas por el programa.

`undisplay exp` Deja de mostrar el valor de la expresión `exp` cada vez que el programa se detiene.

El número de comandos es mucho más grande pero basta con los anteriores para agilizar enormemente nuestro trabajo. Una alternativa a todo esto es usar el `ddd` que se puede considerar como una interfaz gráfica para el `gdb`. Funciona sobre las X y su uso es semejante a de los depuradores existentes en otras plataformas.

22.2. Data Display Debugger: DDD

Como ya hemos comentado anteriormente los programadores suelen pasar mucho tiempo utilizando un depuraror para buscar fallos que no son faciles de encontrar simplemente mirando el código, esto suele ocurrir muy habitualmente cuando se trabaja con memoria dinámica, pues la asignacion de ésta no se hace en tiempo de compilación, sino que se hace en tiempo de ejecución y es más complicado seguirle la pista cuando estamos escribiendo código, además normalmente los analizadores sintácticos de los compiladores no detectan muchos de los errores que cometemos por ser "legales" en determinadas circunstancias.

Hemos presentado el `gdb` que es muy util y potente, de hecho es uno de los pocos depuradores que encontramos para los sistemas linux. El `gdb` es feo, de eso tampoco hay duda y no queremos que las cosas sean tan poco amigables. Así aparecen muchos otros programas que, utilizando ellos el `gdb` por nosotros, consiguen presentarnos una cara más amistosa de este fantástico depurador. Ahora verás uno de los más laureados, el Data Display Debugger.

Este es el aspecto que presenta el `ddd` cuando lo arrancamos, vamos a ir por pasos mostrando lo que normalmente un programador como nosotros vamos a necesitar de un depurador.

22.2.1. Mostrar el contenido de las variables

Para mostrar el contenido de las variables lo que tendremos que hacer es, como se dice coloquialmente, colocar un *watch* a la variable. En `ddd` esto es muy sencillo de hacer y hay varias maneras, una de ellas puede ser escribir el nombre de la variable en el cuadro de texto que tenemos justo por debajo de la barra de menú y pulsar el botón de watch que está al mismo nivel un poco más a la derecha:

Otra forma de hacerlo quizá más rápida poner el cursor del ratón sobre el nombre de la variable en el código fuente, pulsar el boton derecho y escoger la opcion display.

22.2.2. Colocar puntos de ruptura (Breakpoints)

Bueno, ya tenemos un watch de la variable, pero... ¿que pasa cuando pulsamos sobre el botón de run? ¡No vemos nada! Bueno, eso es una circunstancia pasajera, vamos a colocar un *punto de ruptura* o punto de ruptura.

Los puntos de ruptura son lugares donde el depurador hará que nuestro programa pare la ejecucuón para, de esta manera, analizar detenidamente el estado de nuestro programa. La manera de colocar un punto de ruptura en el `ddd` es simplemente desplazarse hasta la línea donde deseamos colocar el punto de ruptura y hacer un doble click con el boton izquierdo del ratón.

Vemos que se nos coloca un símbolo de stop donde hemos hecho el doble click, lo que indica que hay un punto de ruptura. Si pulsamos sobre el punto de ruptura con el boton derecho vemos que se nos da la opcion de desactivarlo, quitarlo y una muy

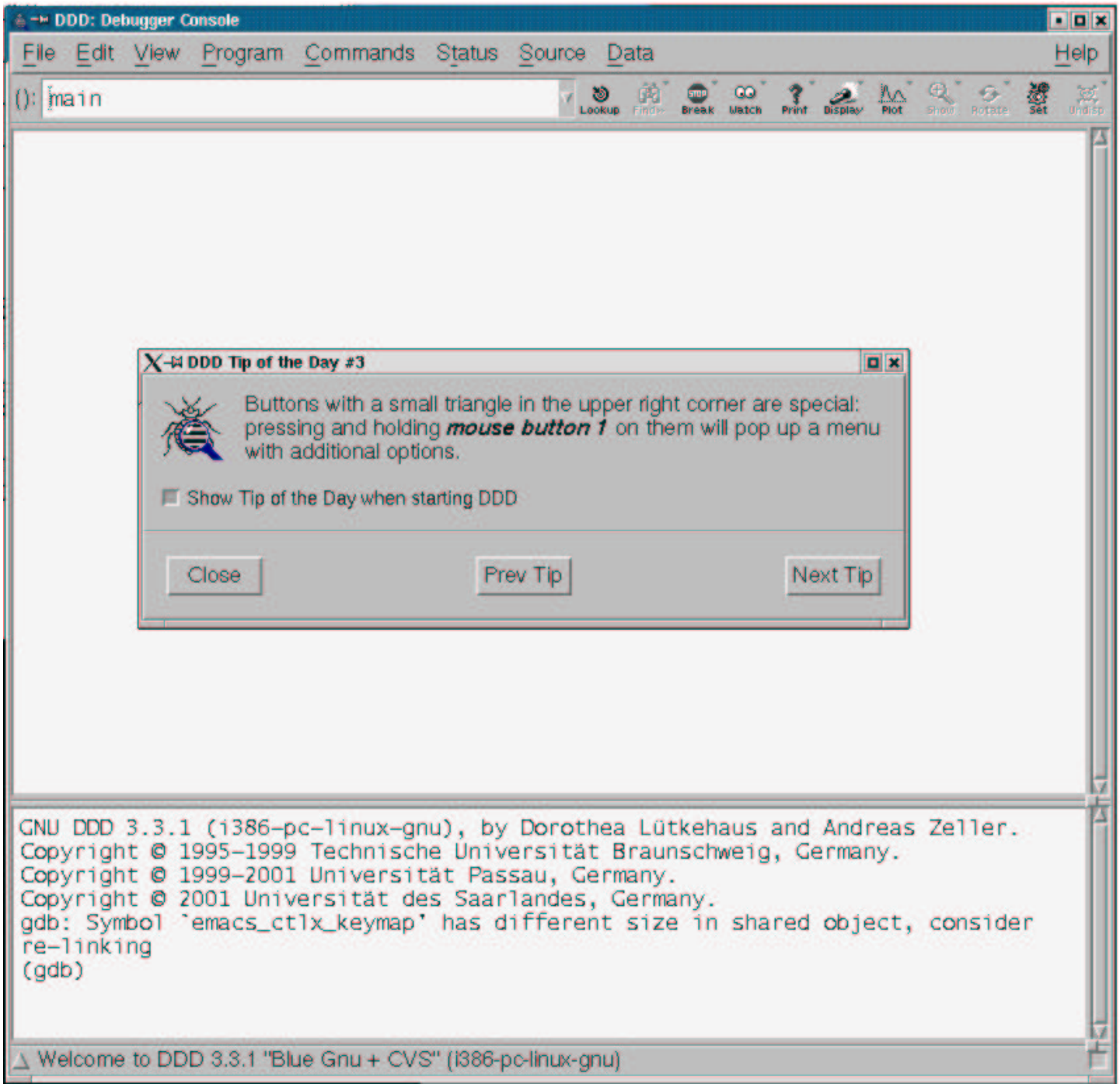


Figura 22.1: Pantalla de inicio

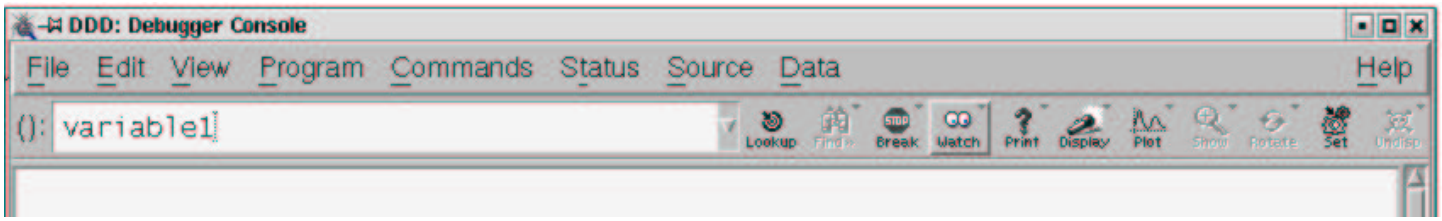


Figura 22.2: Barra de herramientas

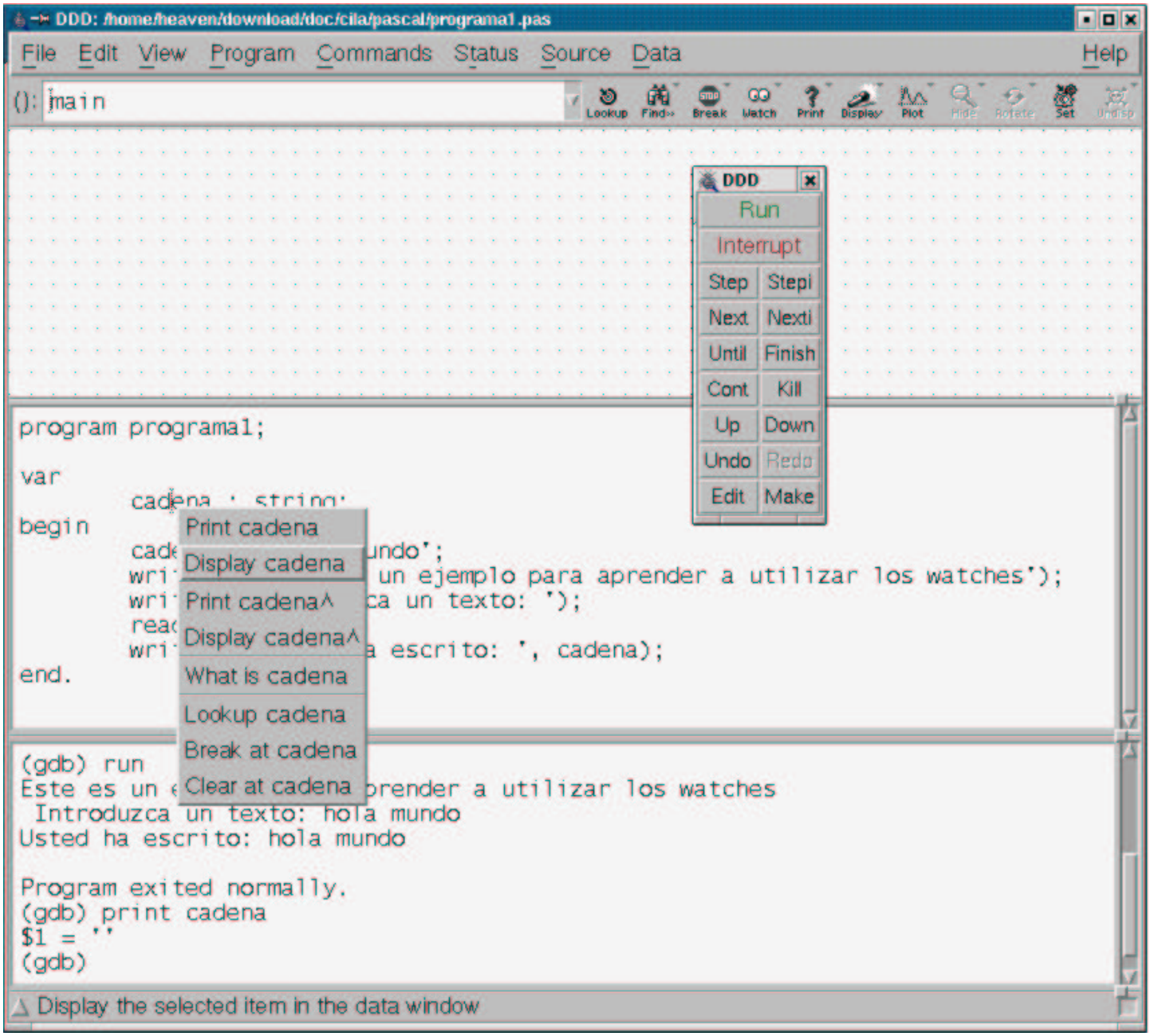


Figura 22.3: Watches

extraña de propiedades. La opcion de propiedades hace que nos aparezca un cuadro de dialogo en el cual podemos introducir una expresión, ésta expresión se utiliza para hacer que nuestro punto de ruptura sea inteligente y no pare la ejecución siempre que el programa pase por allí, sino que solo se parará cuando la expresión se cumpla, por ejemplo, si ponemos el punto de ruptura en un bucle con 10.000 iteraciones y nosotros queremos que se pare cuando la i (variable que cuenta las iteraciones) llegue a 5000 pues lo que hemos de poner en propiedades del punto de ruptura es `i = 5000` y de esa manera nos saltaremos las 5000 primeras iteraciones que no nos interesan.

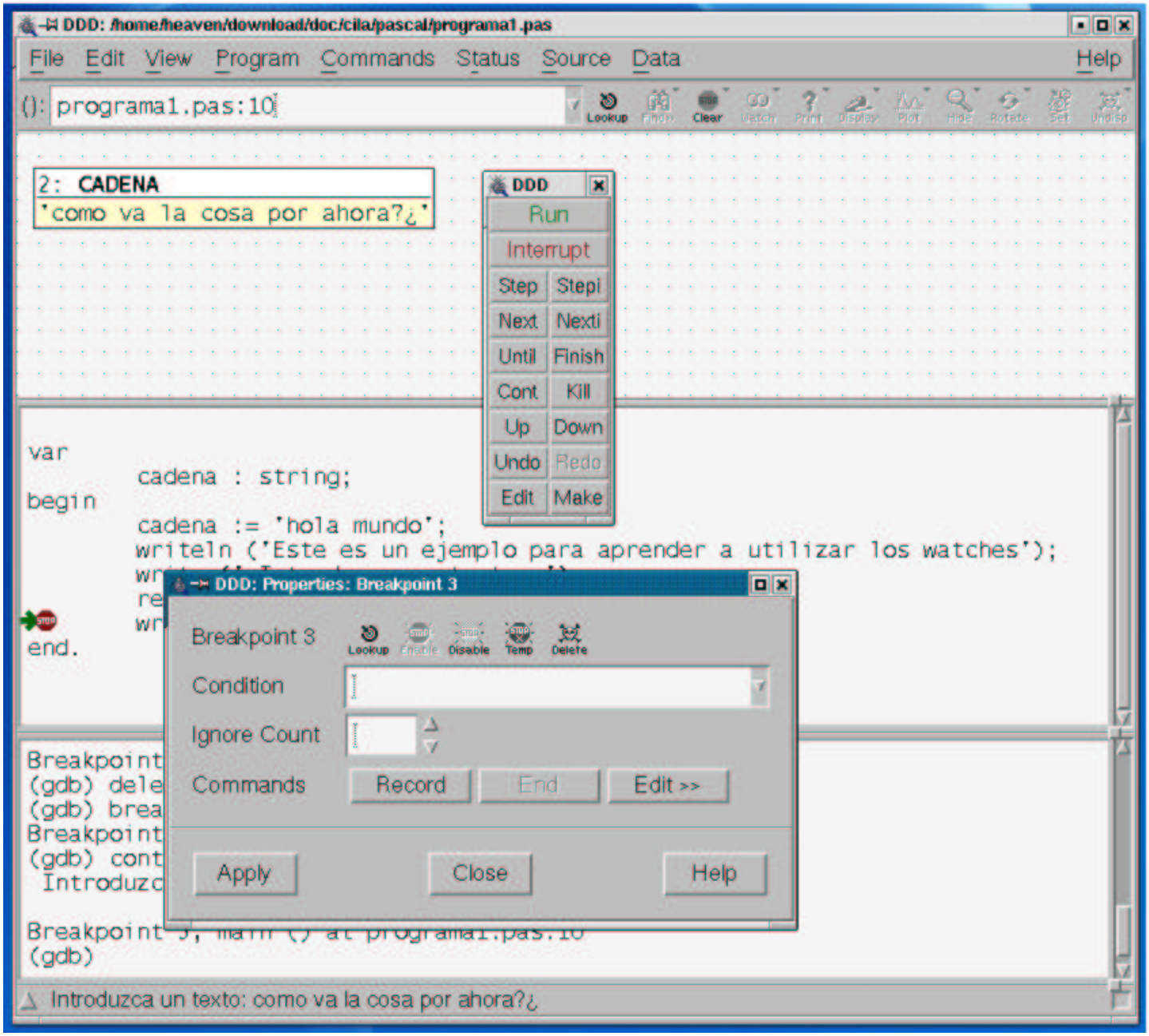


Figura 22.4: Breakpoints, o puntos de ruptura

22.2.3. Ejecución paso a paso

Ahora que ya tienes un watch y un punto de ruptura, en este momento solo podemos ir saltanto de punto de ruptura en punto de ruptura, y esto no nos es suficiente así que vamos a intentar controlar más el flujo de ejecución de nuestros programas. Para esta tarea tenemos varias herramientas comunes en la mayoría de depuradores, en el caso del dd las encontramos en una ventana que tenemos flotando sobre el depurador, con lo cual no es complicado utilizarlas, siempre las tenemos a mano y son las siguientes:

Step: (paso) esta herramienta nos sirve, como su propio nombre indica, para ejecutar nuestro programa paso a paso. Normalmente lo que hacemos es colocar un punto de ruptura al inicio de la funcion o procedimiento que queremos analizar y luego ir pulsando `step` para analizar sentencia a sentencia que es lo que hace nuesto código.

Next: (siguiente) muy parecida a step, solo que pulsando `next` si en el código hay una llamada a una funcion o procedimiento no entramos en ella, sino que ejecuta ésta y nos colocamos en la siguiente línea del programa.

Until: (hasta) esta herramienta nos sirve para ejecutar el programa hasta la posición actual del cursor, es muy parecida a un punto de ruptura temporal.

Cont: (continuar) saltamos hasta el siguiente punto de ruptura o hasta el final del programa si es que no hay ningún punto de ruptura más en el flujo de ejecución.

A parte de estas herramientas tenemos algunas más que se suelen utilizar menos, por ejemplo aquellas que ejecutan exactamente una instrucción (de código máquina).

22.2.4. Visión de punteros

Probablemente estabas deseando que existiera una cosa como la que vas a ver. No te asombres de la potencia de este depurador, es uno de los pocos que lo hacen y de ahí viene su buena fama. La capacidad de la que hablamos es la de *ver* los punteros.

Seguramente has hecho más de un programa con memoria dinámica que no acababa de funcionar bien porque se te perdían punteros y no sabias muy bien por donde andaban o a dónde estaban apuntando, pues bien, esto ya no es excusa para entregar una práctica a medio hacer, el `ddd` viene en nuestra ayuda y tenemos la *herramienta definitiva* para la depuración con punteros.

Cuando haces un watch de una estructura (record) salen todos los campos en un recuadro y los punteros también salen como otro tipo de dato cualquiera. Cuando tienes `$0` en la información de un puntero quiere decir que el puntero es `Nil` (nulo), y cuando tenemos un número precedido del símbolo `$` quiere decir que el puntero está apuntando a esa zona de memoria, por lo tanto, podremos mostrar el contenido de esa zona de memoria. Si decidimos mostrarla aparecerá otro cuadro, como el que inicialmente teníamos, con la información de la zona de memoria apuntada por el puntero que hemos decidido expandir y una flecha desde el recuadro original hasta el nuevo, marcada con el nombre del puntero expandido.

De esta manera es muy sencillo seguir los punteros y ver a que posiciones están apuntando, ver si estamos asignando cosas a punteros que son nulos así como que pinta va teniendo la estructura que estamos construyendo en memoria, lo cual ayuda mucho a la hora de moverse por las estructuras que fabricamos.

Un ejemplo de lo que se puede hacer es mostrar un árbol para, por ejemplo, comprobar que todos los punteros que salen de nuestros nodos hoja están a `Nil` y de esa manera asegurarnos de que nuestro programa no va a fracasar.

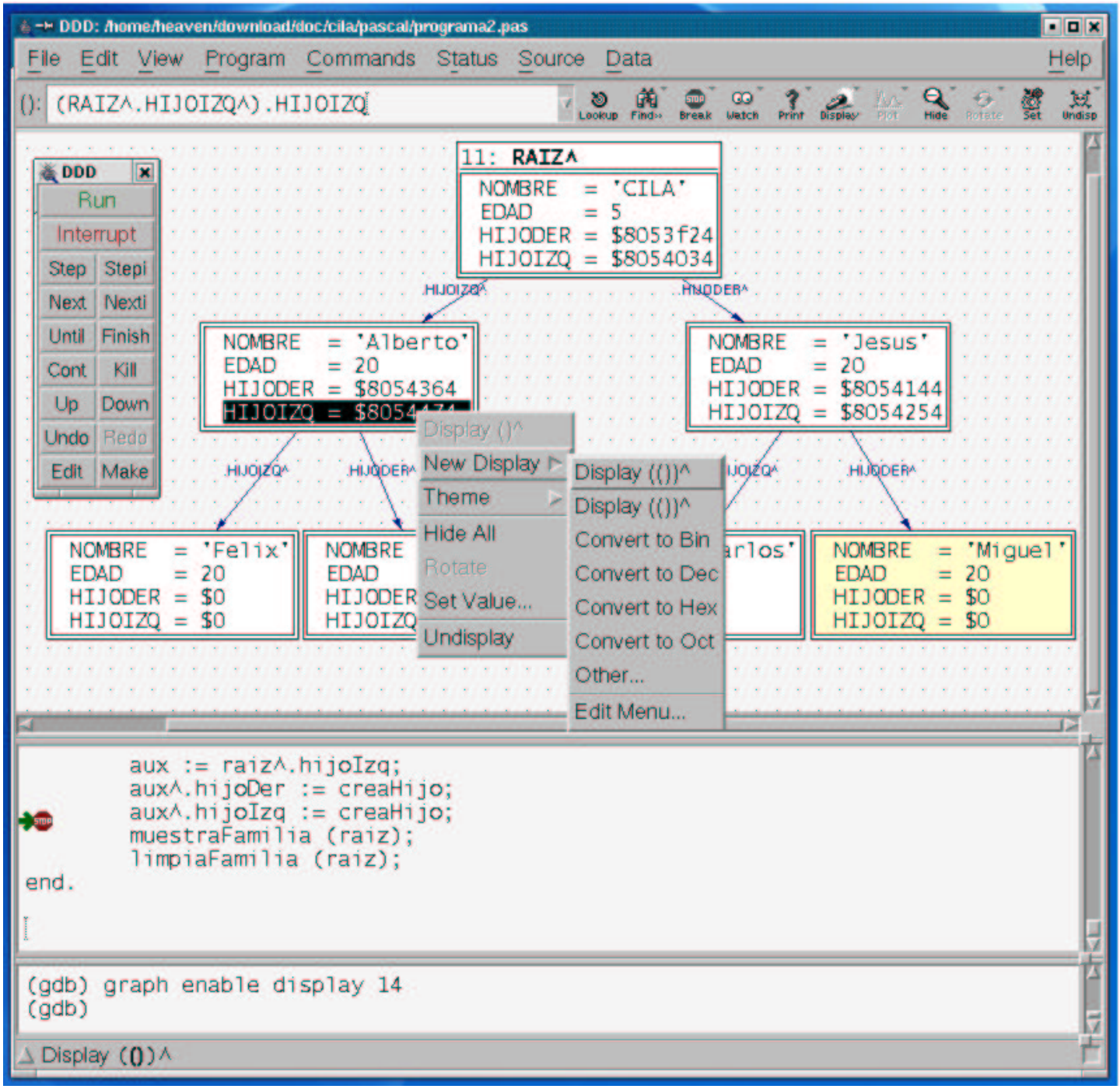


Figura 22.5: Estructura de un árbol binario

TEMA 23

GNU gprof

Puede sernos de mucha utilidad ejecutar nuestro software a través de un *profiler*, como puede ser `gprof`, pues que éste nos proporcionará suficiente información para detectar cuellos de botella en nuestro programa y así poder corregirlos.

23.1. El profiler gprof

Como ya se ha dicho, a menudo deseamos mejorar el rendimiento de nuestro programa, pero en determinadas ocasiones resulta difícil conocer dónde están localizados los cuellos de botella. Bien podría ser una función que consuma una gran cantidad de tiempo de ejecución, como alguna otra que, aunque muy rápida, se ejecuta demasiadas veces. Detectar esto nos permitirá optimizar el código de nuestra aplicación, sustituyendo por macros o funciones `inline` aquellas más llamadas (para eliminar el retardo de resolución de la llamada) y mejorando el código en sí.

Para compilar nuestra aplicación de forma que gprof pueda posteriormente extraer toda esta información, simplemente añadimos el parámetro `-pg` al compilador GCC. Una vez hecho esto, procedemos a ejecutar el programa en sí, y tras salir correctamente de él (al finalizar `main()`, o tras un `exit()`), observaremos que se ha generado un nuevo fichero en el directorio de trabajo llamado `gmon.out`. Sólo nos queda pasar este archivo a través de gprof, con el siguiente formato:

```
gprof <opciones> <ejecutable> <gmon.out>
```

En nuestro sempiterno ejemplo de programa, ejecutaríamos la orden `gprof miprograma gmon.out`.

Hay una opción que merece la pena destacar, que nos muestra el código fuente de nuestro programa junto con un conteo de ejecuciones por línea. Esta opción se activa con el parámetro `-A`.

Una vez más, ampliaremos nuestros conocimientos acerca de esta herramienta usando el comando `info gprof`, o bien con la lectura del manual de GNU.

TEMA 24

Java

Java es un lenguaje de alto nivel y de propósito general. Al principio fue desarrollado en los laboratorios de *Sun Microsystems* para servir en aparatos de electrónica de consumo, como videoteléfonos, set boxes (descodificadores), o aparatos similares, pues pretendía hacerse un lenguaje lo más independiente posible de la plataforma en la que fueran a ser ejecutados los programas hechos con él.

A diferencia de otros muchos lenguajes compilados, el compilador de java no genera ficheros ejecutables. Genera unos ficheros con extensión `.class` llamados *bytecodes*. Estos ficheros posteriormente podrán ser ejecutados mediante la *Máquina Virtual Java* (JVM en inglés), que es la encargada de ejecutar los programas de java.

Esto, que a primera vista puede parecer tedioso e inútil, es una gran ventaja para la portabilidad del código, pues cada plataforma tiene su propia Máquina Virtual Java: Apple Mac OS tiene una, Linux tiene otra, Windows otra, Amiga OS también, etc. Esto implica que si desarrollamos un programa en un iMac con Linux, el amigo que tenga un potente servidor SUN podrá ejecutar nuestra aplicación hecha en Java, al igual que nosotros podremos utilizar el complicado programa de cálculo de estructuras que otra persona ha desarrollado en su PC con Windows en el trabajo.

Realmente, Java no es tan bonito como lo estamos pintando. Uno de sus principales puntos débiles es que no es demasiado rápido, y es bastante caprichoso en cuestiones de recursos de hardware. Además, existen dos máquinas virtuales Java, la de Microsoft y la de SUN Microsystems, que presentan algunas incompatibilidades.

Pero su principal ventaja es la impresionante portabilidad, así como su utilidad en campos como Internet (donde principalmente se usa en la actualidad), en servidores web.

Los programas java podemos clasificarlos básicamente en tres categorías: *applets*, *servlets* y aplicaciones *stand alone*.

- Los *applets* son pequeñas aplicaciones creadas con el propósito de ser incluidas en páginas web. Cuando el cliente, desde su navegador web con java incorporado pide esa página web, el applet se descarga a su ordenador donde comienza su ejecución. El navegador tiene la maquina virtual java incorporada. Los applets son el tipo de aplicaciones java que más restricciones de seguridad presentan. No pueden acceder al sistema de archivos local fuera del directorio en que se ejecutan, no pueden abrir ventanas adicionales sin que éstas aparezcan señalizadas con el indicativo: "Warning: applet window", ni hacer muchas cosas que podrían ser perjudiciales para nuestro ordenador.
- Las aplicaciones que a nivel de seguridad permiten más libertades que los applets.
- Los *servlets* son una especie de applets que se ejecutan sólo en el servidor web cuando uno pide una página, y que generan dinámicamente páginas a partir de fuentes como bases de datos, terceros programas que recopilan información, etcétera.

A nivel visual, java tiene dos grupos de controles, widgets, o como prefiramos llamarlos (no son otra cosa que los campos de texto, formularios, botones, etc.):

- El Java AWT, (Advanced Window Toolkit), obsoleto y mantenido sólo por compatibilidad en versiones actuales de java.
- El conjunto de widgets **Java Swing**, que presenta unos controles mucho más estéticos y es mucho más flexible.

Ambos conjuntos de widgets proporcionan una presentación uniforme independiente de la plataforma en que sean ejecutados. Además, otro de los principales atractivos de java es que el compilador no nos cuesta nada. Se encuentra disponible para bajarlo de java.sun.com para cualquiera de las plataformas más comunes en el mercado.

Veamos cómo se generan programas básicos en java mediante un ejemplo. En un editor escribimos el siguiente código y lo guardamos con el nombre de fichero `Prueba.java`.

Fichero `HolaMundo.java`

```
// Ejemplo 1 de Java para CILA
// Fichero: HolaMundo.java

public class HolaMundo {
    public static void main(String[] argv){
        System.out.println("Hola Mundo");
    }
}
```

Ejemplo 24.1: Ejemplo mínimo de Java

Es de vital importancia que el nombre del fichero coincida con lo que escribimos después de "`public class`". Asimismo, Java es un lenguaje muy exigente en materia de mayúsculas y minúsculas, como de de espacios y tabuladores. Para compilar este programa utilizamos el siguiente comando:

```
$ ls
HolaMundo.java

$ javac HolaMundo.java

$ ls
HolaMundo.java  HolaMundo.class
```

Obtendremos un fichero con extensión `.class` que es el *bytecode*, el fichero que ejecutaremos con el siguiente comando:

```
$ java HolaMundo
Hola Mundo
```

Lo que no tiene sentido es poner "`java Prueba.class`", pues no funcionaría, devolviendo el siguiente error:

```
$ java HolaMundo.class
Can't find class HolaMundo.class

$ java HolaMunco.class
Exception in thread "main" java.lang.NoClassDefFoundError: HolaMundo/java
```

Veamos ahora un ejemplo de *applet*. Tecleamos el siguiente código y lo guardamos como `HolaMundo2.java`.

Fichero `HolaMundo2.java`

```
// Ejemplo 2 de Java para CILA
// Fichero: HolaMundo2.java

import java.applet.Applet;
import java.awt.*;

public class HolaMundo2 extends Applet {
    public Button botonUno = new Button ("Hola Mundo");
    public void init() {
    add ( botonUno );
    }
}
```

Ejemplo 24.2: Applet mínimo de Java

Compilamos el applet del mismo modo que hicimos con el ejemplo anterior:

```
$ javac HolaMundo.java
```

Con esto generamos el fichero `HolaMundo2.class` que contiene el applet. Ahora necesitamos una página web que cargue el applet. Creamos el fichero `HolaMundo2.html` con el siguiente código:

Fichero `HolaMundo2.html`

```
<html>
<head>
<title> Ejemplo 2 de Java para CILA </title>
</head>
<body>
<center>
<applet code="HolaMundo2.class" width="300" height="300" ></applet>
</center>
</body>
</html>
```

Ejemplo 24.3: Página HTML para incluir el applet de Java

Para probar el applet utilizaremos el método que utilizaría cualquier visitante, cargarlo desde la página web que hemos creado a tal efecto. Esto lo hacemos con cualquier navegador que soporte Java, entre los que recomendamos Netscape.

```
$ netscape HolaMundo2.html
```

Veremos como el applet se ejecuta dentro de la página web. Otra forma de ejecutar un applet de Java, por ejemplo cuando estamos programando y sólo deseamos probarlo pero no queremos ejecutar netscape, es el programa appletviewer que proporciona el JDK.

```
$ appletviewer HolaMundo2.html
```

Podemos apreciar la diferencia entre un *applet*, cuya ejecución está controlada por el navegador y se limita a la página web desde el que es cargado, y una aplicación intependiente que se ejecuta directamente en la consola del sistema sin más intermediario que la Máquina Virual Java.

TEMA 25

Expresiones regulares

Las expresiones regulares se comprenden mejor desde su utilidad que desde su definición, así que vamos a introducirlas con el ejemplo que utilizan en el libro [**?**].

Imagina que necesitas una herramienta para buscar palabras repetidas en documentos, cosas como "esto esto", un problema bastante frecuente en documentos que son editados una y otra vez (como este libro).

Tendrías que escribir una solución que: detectara palabras repetidas de una línea a la siguiente, cuando una palabra está al final de una línea y también al principio de la siguiente línea; que no tuviera en cuenta si las letras están en minúsculas o mayúsculas y encontrar palabras repetidas en medio de código HTML, LaTeX, LyX u otros lenguajes de marcado usados en documentación.

Esto es un problema real, y el autor del libro [**?**] escribió esta solución y se vió sorprendido por la cantidad de palabras repetidas que encontró en el propio libro. Este problema puede resolverse de muchas maneras y en muchos lenguajes de programación, pero las expresiones regulares son sin duda la herramienta que hacen este tipo de problemas más fáciles y rápidos de resolver.

25.1. Definición de expresión regular

Una *expresión regular* es un literal (cadena de caracteres) que define una serie de reglas que debe cumplir una expresión para considerar que *encaja* con la expresión regular. Esto se consigue utilizando caracteres con significados especiales para *describir* un patrón con el que deben encajar las expresiones.

Los caracteres con significados especiales se denominan *metacaracteres*, y aunque la mayoría son comunes entre los distintos estilos de expresiones regulares hay diferencias y conviene estudiarlos desde la herramienta en la que usemos las expresiones regulares. Así, nos encontramos con expresiones regulares de UNIX, de VI, de PERL, de Emacs, etc. y no tienen la misma sintaxis exactamente.

Dado que esto no es un curso de programación no veremos las expresiones regulares al estilo de lenguajes de programación sino al estilo de los editores de texto.

25.2. Expresiones regulares en VI & VIM

Una de las funcionalidades más queridas de VI es la búsqueda y sustitución de texto. Para empezar pensemos en simples palabras, por ejemplo en substituir la palabra "coche" por la palabra "carro". El comando para la substitución es `:s` y tiene el siguiente esquema:

```
:rango/busca/sustituye/[gc]
```

Lo que pongas en `rango` determina la región del fichero que será afectada por el comando. Puedes poner dos números de línea separados por coma para afectar a todas las líneas entre esas dos (ambas inclusive), o bien substituir el número de la segunda línea por un `$` para extender la región afectada hasta el final del fichero. Si en lugar de líneas poner `%` afectará a todo el fichero.

Las opciones `g` y `c` del final son opcionales. La opción `g` hace que se substituyan todas las ocurrencias en cada línea, ya que si no la especificas sólo se substituye la primera ocurrencia de cada línea. La opción `c` hace que `VI` pida confirmación antes de substituir cada ocurrencia.

La expresión `busca` es el patrón que VIM buscará. Puede ser una palabra, o más en general una *expresión regular*. La expresión `substituye` es el patrón que VIM utilizará para substituir en las ocurrencias, y también puede ser una *expresión regular*.

En VI las expresiones regulares utilizan los siguientes metacaracteres: `. ^ $ [ ] \ * - /`

El punto (`.`) encaja con cualquier caracter, `$` con el final de la línea, `^` con el principio de la línea cuando es el primero en la expresión regular, mientras que en otro caso encaja con cualquier caracter que **no** sea el que le suceda. Los caracteres `-`, `[` y `]` permiten definir rangos, `*` extiende el significado de la expresión que le preceda a ninguna o varias ocurrencias de la misma. La barra `/` se utiliza para separar los patrones de búsqueda. Para utilizar alguno de estos caracteres de forma que encajen consigo mismos (por ejemplo para buscar un punto en el texto) hay que *escaparlos* anteponiéndoles una barra invertida (\).

Otros caracteres adquieren un significado especial cuando son precedidos de la barra invertida. Así `\|` es el operador "o", es decir la expresión `casa\|coche` encaja con `casa` y con `coche`. Los símbolos de desigualdad permiten delimitar palabras en un patrón, usando `\<` que encaja con el principio de palabra y `\>` con el final. Así podremos sustituir la palabra `vi` por `vim` sin que cambien palabras del texto como `vivir`.

```
:s/\<vi\>/vim
```

Podemos utilizar el carácter `^` para indicar que no exista ningun carácter antes de la expresión regular que queremos sustituir en esa línea, y el carácter `$` para indica que no exista ningun carácter despuéas de la expresión regular que queremos sustituir en usa línea. Vamos a ver unos ejemplos:

```
:s/^vi/vim/g
```

En este ejemplo, se cambia la palabra `vi` por `vim` si no existe ningun carácter delante de la palabra `vi`.

```
:s/vi$/vim
```

En el anterior ejemplo, se sustituye la palabra `vi` por `vim` que se encuentre al final de una línea.

```
:s/^vi$/vim
```

Los metacaracteres nos permiten una gran flexibilidad a la hora de trabajar con expresiones regulares.

`.`	Cualquier caracter
`\s`	Espacios en blanco
`\d`	Digitos
`\h`	Cualquier letra
`\w`	Letras y digitos
`\l`	Letras minúsculas
`\u`	Letras mayúsculas

Tabla 25.1: Metacaracteres en las expresiones regulares de VI

Vamos a poner un ejemplo con este tipo de comandos. Si queremos buscar en el texto fechas con el formato DD/MM/AAAA, lo que haríamos sería:

```
/\d\d\/\d\d\/\d\d\d\d
```

O por ejemplo si buscamos palabras que empiezen con mayúsculas de un modo sencillo podemos hacer:

```
/\<\u
```

Ahora vamos a introducir unos elementos nuevos para hacer reemplazos y busquedas más "profesionales". Uno de ellos son los *cuantificadores*. Si necesitamos buscar en un documento grandes grupos de números de 9 cifras por ejemplo nos resultaría muy laborioso escribir `/\d\d\d\d\d\d\d\d\d`. Los cuantificadores son diversas expresiones que nos permiten definir el número de veces que aparecerá el tipo de caracter que queremos buscar. Los más importantes son `\+`, relaciona el caracter que aparece una o más veces en una palabra, `\=`, relaciona el caracter que aparece 0, 1 o más veces en una palabra. Con `\{n,m}` relaciona el caracter si se encuentra entre n y m veces. La expresión `\{n}` relaciona el caracter si se encutra n veces. Así en el ejemplo anterior podríamos usar:

```
/\d\{9}
```

y nos mostraría las expresiones que contengan 9 digitos.

Podemos combinar los cuantificadores con los rangos de caracteres para conseguir mejores resultados. Los rangos de expresiones regulares son definidos dentro de `[]`. Por ejemplo, `/[a-h]` buscará en el texto todos las letras entre la a y la h, ambas inclusive, `/[A-H]` haría lo mismo en mayúsculas. También existe un valor negativo en los rangos, es el `^`. El comando `/[^j-z]` mostrará todas las letras minúsculas anteriores a la j.

Para buscar mayúsculas o minúsculas hay algo mejor que los rangos `[a-z]` y `[A-Z]`, que son las *clases*. Las clases se refieren mediante palabras encerradas entre `[[:` y `:]]`, por lo que para buscar una expresión mediante clases el patrón de búsqueda deberá contener `[[:clase:]]` Las clases que nos interesan son:

`[[:lower:]]`	cualquier letra minúscula (incluida ñ)
`[[:upper:]]`	cualquier letra mayúscula (incluida Ñ)
`[[:alpha:]]`	cualquier letra
`[[:digit:]]`	cualquier dígito

Tabla 25.2: Metacaracteres en las expresiones regulares de VI

Tambien podemos cambiar el orden de las frases de todo un texto o de zonas específicas, utilizando los *agrupadores*. Se trata de crear grupos de expresiones que luego para sustituir utilizaremos expresiones que nos permitan reordenarlos. Para definir un grupo se utiliza `\(\)`, el texto que se encuentre entre esos el paréntesis "escapados" será la expresion que buscará. Por ejemplo, estamos usando un archivo con varios nombres de ciudades, pero para poder utilizarlo necesitamos que todos esten al principio de la línea, sin ningun espacio, podemos utilizar el siguiente comando:

```
:%s/^\(\s\+\)\(\w\+\)/\2/g
```

El anterior comando busca en todo el documento (`%`) que contenga cualquier número espacios al principio de la línea (`\(\s\+\)`) y que detrás contenga algún texto(`\(\w\+\)`) y lo sustituye por la exresión(`\2`), es decir, solo mantiene el grupo `\2`, el que contiene la palabra.

Apéndice A

Recursos en internet

La mayoría de los recursos que podamos necesitar para usar Linux se encuentran distribuidos por Internet, y una enorme cantidad están disponibles en español.

- `http://www.debian.org` – El Proyecto Debian es una asociación de personas que han hecho causa común para crear un sistema operativo (SO) libre. Este sistema operativo que hemos creado se llama Debian GNU/Linux, o simplemente Debian para acortar.
- `http://www.redhat.es` – La primera distribución Linux en reorientarse hacia los usuarios finales.
- `http://www.linux-mandrake.com/es/` – Linux-Mandrake es un amigable Sistema Operativo Linux. Es muy fácil de usar, tanto en el hogar u oficina como en servidores. Está disponible en forma gratuita en varios idiomas alrededor del mundo.
- `http://www.ututo.org` – Un GNU/Linux Simple.
- `http://www.linux-es.com` – Existen muchos lugares en Internet dedicados a LINUX, pero la mayoría de ellos están en inglés. Estas páginas pretenden ser un punto de partida para aquellos que necesitan encontrar información sobre este sistema y en la medida de lo posible se ha intentado que la mayoría de los enlaces y contenidos sean en castellano.
- `http://www.gnu.org/home.es.html` – El Projecto GNU comenzó en 1984 para desarrollar un sistema operativo tipo Unix completo, el cual es software libre: El sistema GNU. Variantes del sistema GNU, utilizando Linux como kernel, son ampliamente usadas, y aunque frecuentemente llamadas "Linux", dichas variantes deberían referirse más exactamente como sistemas GNU/Linux.
- `http://es.tldp.org` – Proyecto LuCAS - La mayor biblioteca en español dedicada a GNU/LiNUX de todo el planeta
- `http://www.insflug.org` – En el INSFLUG se coordina la traducción "oficial" de documentos breves, como los COMOs y PUFs o Preguntas de Uso Frecuente, las FAQs en inglés. Esperamos que la información que encuentre aquí le sea de utilidad.
- `http://www.gnu.org/software/emacs/emacs.html` – Es un editor de pantalla y ambiente para cómputo de tiempo real, extensible y personalizable. Ofrece Lisp (finalmente integrado al editor) para escribir extensiones y proporciona una interfaz al sistema de ventanas X.
- `http://www.vim.org` – VIM es una versión mejorada del editor VI, uno de los editores de texto estándar en los sistemas UNIX. VIM añade muchas de las características que se esperan en un editor: Deshacer ilimitado, coloreado de sintaxis, GUI, y mucho más.
- `http://www.chiark.greenend.org.uk/ sgtatham/putty/` – Implementación libre de un cliente TELNET/SSH para sistemas operativos Microsoft® Windows®. Escrito y mantenido por Simon Tatham.
- `http://pinsa.escomposlinux.org/sromero/linux/` – S.O.S. Linux.
- `http://www.geocities.com/Athens/Temple/2269/` – Tutorial de C/C++
- `http://www.cervantex.org` – Información LaTeX en español

- `http://www.lyx.org` – Página del proyecto LyX
- `http://www.octave.org` – Página del proyecto Octave
- `http://www.gnuplot.org` – Página del proyecto Gnuplot
- `http://www.r-project.org` – Página del proyecto R
- `http://yacas.sourceforge.net` – Página del proyecto Yacas
- `http://www.lysator.liu.se/~alla/dia/` – Página del proyecto DIA
- `http://www.gimp.org` – Página del proyecto GIMP
- `http://www.qcad.org` – Página del proyecto QCad
- `http://www.linuxfocus.org/Castellano/January2002/article132.shtml` – Tutorial de QCad en Castellano.

Apéndice B

Licencia de Documentación Libre GNU

Versión 1.1, Marzo de 2000

Ésta es la GNU Free Document License (GFDL), versión 1.1 (de marzo de 2000), que cubre manuales y documentación para el software de la Free Software Foundation, con posibilidades en otros campos. La traducción [1] no tiene ningún valor legal, ni ha sido comprobada de acuerdo a la legislación de ningún país en particular. Vea el original en `http://www.gnu.org/copyleft/fdl.html`

Los autores de esta traducción son:

- Igor Támara (`ikks@bigfoot.com`)
- Pablo Reyes (`reyes_pablo@hotmail.com`)
- Revisión: Vladimir Támara P. (`vtamara@gnu.org`)

Copyright © 2000 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Se permite la copia y distribución de copias literales de este documento de licencia, pero no se permiten cambios.

Preámbulo

El propósito de esta licencia es permitir que un manual, libro de texto, u otro documento escrito sea "libre" en el sentido de libertad: asegurar a todo el mundo la libertad efectiva de copiarlo y redistribuirlo, con o sin modificaciones, de manera comercial o no. En segundo término, esta licencia preserva para el autor o para quien publica una manera de obtener reconocimiento por su trabajo, al tiempo que no se consideran responsables de las modificaciones realizadas por terceros.

Esta licencia es una especie de "copyleft" que significa que los trabajos derivados del documento deben a su vez ser libres en el mismo sentido. Esto complementa la Licencia Pública General GNU, que es una licencia de copyleft diseñada para el software libre.

Hemos diseñado esta Licencia para usarla en manuales de software libre, ya que el software libre necesita documentación libre: Un programa libre debe venir con los manuales que ofrezcan la mismas libertades que da el software. Pero esta licencia no se limita a manuales de software; puede ser usada para cualquier trabajo textual, sin tener en cuenta su temática o si se publica como libro impreso. Recomendamos esta licencia principalmente para trabajos cuyo fin sea instructivo o de referencia.

[1] N. del T. Derechos Reservados en el sentido de GNU `http://www.gnu.org/copyleft/copyleft.es.html`

B.1. Aplicabilidad y definiciones

Esta Licencia se aplica a cualquier manual u otro documento que contenga una nota del propietario de los derechos que indique que puede ser distribuido bajo los términos de la Licencia. El "Documento", en adelante, se refiere a cualquiera de dichos manuales o trabajos. Cualquier miembro del público es un licenciatario, y será denominado como "Usted".

Una "Versión Modificada" del Documento significa cualquier trabajo que contenga el Documento o una porción del mismo, ya sea una copia literal o con modificaciones y/o traducciones a otro idioma.

Una "Sección Secundaria" es un apéndice titulado o una sección preliminar al prólogo del Documento que tiene que ver exclusivamente con la relación de quien publica, o los autores del Documento, o el tema general del Documento(o asuntos relacionados) y cuyo contenido no entra directamente en este tema general. (Por ejemplo, si el Documento es en parte un texto de matemáticas, una Sección Secundaria puede no explicar matemáticas.) La relación puede ser un asunto de conexión histórica, o de posición legal, comercial, filosófica, ética o política con el tema o la materia del texto.

Las "Secciones Invariantes" son ciertas Secciones Secundarias cuyos títulos son denominados como Secciones Invariantes, en la nota que indica que el documento es liberado bajo esta licencia.

Los "Textos de Cubierta" son ciertos pasajes cortos de texto que se listan, como Textos de Portada o Textos de Contra Portada, en la nota que indica que el documento está liberado bajo esta Licencia.

Una copia "Transparente" del Documento, significa una copia para lectura en máquina, representada en un formato cuya especificación está disponible al público general, cuyos contenidos pueden ser vistos y editados directamente con editores de texto genéricos o (para imágenes compuestas por pixeles) de programas genéricos de dibujo o (para dibujos) algún editor gráfico ampliamente disponible, y que sea adecuado para exportar a formateadores de texto o para traducción automática a una variedad de formatos adecuados para ingresar a formateadores de texto. Una copia hecha en un formato de un archivo que no sea Transparente, cuyo formato ha sido diseñado para impedir o dificultar subsecuentes modificaciones posteriores por parte de los lectores no es Transparente. Una copia que no es "Transparente" es llamada "Opaca".

Como ejemplos de formatos adecuados para copias Transparentes están el ASCII plano sin formato, formato de Texinfo, formato de LaTeX, SGML o XML usando un DTD disponible ampliamente, y HTML simple que sigue los estándares, diseñado para modificaciones humanas. Los formatos Opacos incluyen PostScript, PDF, formatos propietarios que pueden ser leídos y editados unicamente en procesadores de palabras propietarios, SGML o XML para los cuáles los DTD y/o herramientas de procesamiento no están disponibles generalmente, y el HTML generado por máquinas producto de algún procesador de palabras sólo para propósitos de salida.

La "Portada" en un libro impreso significa, la propia portada junto con las páginas siguientes necesarias para mantener la legibilidad del material, que esta Licencia requiere que aparezca en la portada. Para trabajos en formatos que no tienen Portada como tal, "Portada" significa el texto junto a la aparición más prominente del título del trabajo, precediendo el comienzo del cuerpo del trabajo.

B.2. Copia literal

Puede copiar y distribuir el Documento en cualquier medio, sea en forma comercial o no, siempre y cuando esta Licencia, las notas de derecho de autor, y la nota de licencia que indica que esta Licencia se aplica al Documento se reproduzca en todas las copias, y que usted no añada ninguna otra condición a las expuestas en en esta Licencia. No puede usar medidas técnicas para obstruir o controlar la lectura o copia posterior de las copias que usted haga o distribuya. Sin embargo, usted puede aceptar compensación a cambio de las copias. Si distribuye un número suficientemente grande de copias también deberá seguir las condiciones de la sección 3.

También puede prestar copias, bajo las mismas condiciones establecidas anteriormente, y puede exhibir copias publicamente.

B.3. Copiado en cantidades

Si publica copias impresas del Documento que sobrepasen las 100, y la nota de Licencia del Documento exige Textos de Cubierta, debe incluir las copias con cubiertas que lleven en forma clara y legible, todos esos textos de Cubierta: Textos Frontales en la cubierta frontal, y Textos Posteriores de Cubierta en la Cubierta Posterior. Ambas cubiertas deben identificarlo a Usted clara y legiblemente como quien publica tales copias. La Cubierta Frontal debe mostrar el título completo con todas las palabras igualmente prominentes y visibles. Además puede añadir otro material en la cubierta. Las copias con cambios

limitados en las cubiertas, siempre que preserven el título del Documento y satisfagan estas condiciones, puede considerarse como copia literal.

Si los textos requeridos para la cubierta son muy voluminosos para que ajusten legiblemente, debe colocar los primeros (tantos como sea razonable colocar) en la cubierta real, y continuar el resto en páginas adyacentes.

Si publica o distribuye copias Opacas del Documento cuya cantidad exceda las cien, debe incluir una copia Transparente que pueda ser leída por una máquina con cada copia Opaca, o entregar en o con cada copia Opaca una dirección en red de computador públicamente-accesible conteniendo una copia completa Transparente del Documento, sin material adicional, a la cual el público en general de la red pueda acceder a bajar anónimamente sin cargo usando protocolos de standard público. Si usted hace uso de la última opción, deberá tomar medidas necesarias, cuando comience la distribución de las copias Opacas en cantidad, para asegurar que esta copia Transparente permanecerá accesible en el sitio por lo menos un año después de su última distribución de copias Opacas (directamente o a través de sus agentes o distribuidores) de esa edición al público.

Se solicita, aunque no es requisito, que contacte con los autores del Documento antes de redistribuir cualquier número de copias, para permitirle la oportunidad de que le suministren una versión del Documento.

B.4. Modificaciones

Puede copiar y distribuir una Versión Modificada del Documento bajo las condiciones de las seccions 2 y 3 anteriores, siempre que Usted libere la Versión Modificada bajo esta misma Licencia, con la Versión Modificada haciendo el rol del Documento, por lo tanto licenciando la distribución y modificación de la Versión Modificada a quien quiera que posea una copia de éste. Además, debe hacer lo siguiente en la Versión Modificada:

- Uso en la Portada (y en las cubiertas, si hay alguna) de un título distinto al del Documento, y de versiones anteriores (que deberían, si hay alguna, estar listados en la sección de Historia del Documento). Puede usar el mismo título que el de las versiones anteriores al original siempre que quién publicó la primera versión lo permita.
- Listar en la Portada, como autores, una o más personas, o entidades responsables por la autoría o las modificaciones en la Versión Modificada, junto con por lo menos cinco de los autores principales del Documento (Todos sus autores principales, si son inferiores a cinco).
- Estado en la Portada del nombre de quien publica la Versión Modificada, como quien publica.
- Preservar todas las notas de derechos de autor del Documento.
- Añadir una nota de derecho de autor apropiada a sus modificaciones adyacentes a las otras notas de derecho de autor.
- Incluir, inmediatamente después de la nota de derecho de autor, una nota de licencia dando el permiso público para usar la Versión Modificada bajo los términos de esta Licencia, de la forma mostrada en la Adición (LEGAL) abajo.
- Preservar en esa nota de licencia el listado completo de Secciones Invariantes y en los Textos de las Cubiertas que sean requeridos como se especifique en la nota de Licencia del Documento
- Incluir una copia sin modificación de esta Licencia.
- Preservar la sección llamada “Historia”, y su título, y añadir a esta una sección estableciendo al menos el título, el año,los nuevos autores, y quién publicó la Versión Modificada como reza en la Portada. Si no hay una sección titulada “Historia” en el Documento, crear una estableciendo el título, el año, los autores y quién publicó el Documento como reza en la Portada, añadiendo además un artículo describiendo la Versión Modificada como se estableció en el punto anterior.
- Preservar la localización en red, si hay, dada en la Documentación para acceder públicamente a una copia Transparente del Documento, tanto como las otras direcciones de red dadas en el Documento para versiones anteriores en las cuáles estuviese basado. Éstas pueden ubicarse en la sección “Historia”. Se puede omitir la ubicación en red para un trabajo que sea publicado por lo menos 4 años antes que el mismo Documento, o si quien publica originalmente la versión da permiso explícitamente.
- En cualquier sección titulada “Agradecimientos” o “Dedicatorias”, preservar el título de la sección, y preservar en la sección toda la sustancia y el tono de los agradecimientos y/o dedicatorias de cada contribuyente que estén incluidas.

- Preservar todas las Secciones Invariantes del Documento, sin alterar su texto ni sus títulos. Los números de sección o el equivalente no son considerados parte de los títulos de la sección. M. Borrar cualquier sección titulada "Aprobaciones". Tales secciones no pueden estar incluidas en las Versiones Modificadas.
- Borrar cualquier sección titulada "Aprobaciones". Tales secciones no pueden estar incluidas en las Versiones Modificadas.
- No retitular ninguna sección existente como "Aprobaciones" o conflictuar con título con alguna Sección Invariante.

Si la Versión Modificada incluye secciones, apéndices nuevos o preliminares al prólogo que califican como Secciones Secundarias y contienen material no copiado del Documento, puede opcionalmente designar algunas o todas esas secciones como invariantes. Para hacerlo, añada sus títulos a la lista de Secciones Invariantes en la nota de licencia de la Versión Modificada. Tales títulos deben ser distintos de cualquier otro título de sección.

Puede añadir una sección titulada "Aprobaciones", siempre que contenga únicamente aprobaciones de su Versión Modificada por varias fuentes. Por ejemplo, observaciones de peritos o que el texto ha sido aprobado por una organización como un standard.

Puede añadir un pasaje de hasta cinco palabras como un Texto de Cubierta Frontal, y un pasaje de hasta 25 palabras como un texto de Cubierta Posterior, al final de la lista de Textos de Cubierta en la Versión Modificada. Solamente un pasaje de Texto de Cubierta Frontal y un Texto de Cubierta Posterior puede ser añadido por (o a manera de arreglos hechos por) una entidad. Si el Documento ya incluye un texto de cubierta para la misma cubierta, previamente añadido por usted o por arreglo hecho por la misma entidad, a nombre de la cual está actuando, no puede añadir otra, pero puede reemplazar la anterior, con permiso explícito de quien publicó anteriormente tal cubierta.

El(los) autor(es) y quien(es) publica(n) el Documento no da(n) con esta Licencia permiso para usar sus nombres para publicidad o para asegurar o implicar aprobación de cualquier Versión Modificada.

B.5. Combinando documentos

Puede combinar el Documento con otros documentos liberados bajo esta Licencia, bajo los términos definidos en la sección 4 anterior para versiones modificadas, siempre que incluya en la combinación todas las Secciones Invariantes de todos los documentos originales, sin modificar, y listadas todas como Secciones Invariantes del trabajo combinado en su nota de licencia.

El trabajo combinado necesita contener solamente una copia de esta Licencia, y múltiples Secciones Invariantes Idénticas que pueden ser reemplazadas por una sola copia. Si hay múltiples Secciones Invariantes con el mismo nombre pero con contenidos diferentes, haga el título de cada una de estas secciones único añadiéndole al final de éste, en paréntesis, el nombre del autor o de quien publicó originalmente esa sección, si es conocido, o si no, un número único. Haga el mismo ajuste a los títulos de sección en la lista de Secciones Invariantes en la nota de licencia del trabajo combinado.

En la combinación, debe combinar cualquier sección titulada "Historia" de los varios documentos originales, formando una sección titulada "Historia"; de la misma forma combine cualquier seción titulada "Agradecimientos", y cualquier sección titulada "Dedicatorias". Debe borrar todas las secciones tituladas "Aprobaciones".

B.6. Colecciones de documentos

Puede hacer una colección consistente del Documento y otros documentos liberados bajo esta Licencia, y reemplazar las copias individuales de esta Licencia en los varios documentos con una sola copia que esté incluida en la colección, siempre que siga las reglas de esta Licencia para una copia literal de cada uno de los documentos en cualquiera de todos los aspectos.

Puede extraer un solo documento de una de tales colecciones, y distribuirlo individualmente bajo esta Licencia, siempre que inserte una copia de esta Licencia en el documento extraído, y siga esta Licencia en todos los otros aspectos concernientes a la copia literal de tal documento.

B.7. Agregación con trabajos independientes

Una recopilación del Documento o de sus derivados con otros documentos o trabajos separados o independientes, en cualquier tipo de distribución o medio de almacenamiento, no como un todo, cuenta como una Versión Modificada del

Documento, teniendo en cuenta que ninguna compilación de derechos de autor sea clamada por la recopilación. A tal recopilación se le llama "agregado", y esta Licencia no se aplica a los otros trabajos auto-contenidos y por lo tanto compilados con el Documento, o a cuenta de haber sido compilados, si no son ellos los mismos trabajos derivados del Documento.

Si el requerimiento de la sección 3 del Texto de la Cubierta es aplicable a estas copias del Documento, entonces si el Documento es menor que un cuarto del agregado entero, Los Textos de la Cubierta del Documento pueden ser colocados en cubiertas que enmarquen solamente el Documento entre el agregado. De otra forma deben aparecer en cubiertas enmarcando todo el agregado.

B.8. Traducción

Se considera a la Traducción como una clase de modificación. Así que puede distribuir traducciones del Documento bajo los términos de la sección 4. Reemplazar las Secciones Invariantes con traducciones requiere permiso especial de los dueños de derecho de autor, pero puede incluir traducciones de algunas o todas las Secciones Invariantes adicionalmente a las versiones originales de las Secciones Invariantes. Puede incluir una traducción de esta Licencia siempre que incluya también la versión Inglesa de esta Licencia. En caso de un desacuerdo entre la traducción y la versión original en Inglés de esta Licencia, la versión original en Inglés prevalecerá.

B.9. Terminación

No se puede copiar, modificar, sublicenciar, o distribuir el Documento excepto por lo permitido expresamente bajo esta Licencia. Cualquier otro intento de copia, modificación, sublicenciamiento o distribución del Documento es nulo, y sus derechos serán automáticamente retirados de esa licencia. De todas maneras, los terceros que hayan recibido copias o derechos de su parte bajo esta Licencia no tendrán por terminadas sus licencias siempre que tales personas o entidades se encuentren en total conformidad con la licencia original.

B.10. Futuras revisiones de esta licencia

La Free Software Foundation puede publicar nuevas versiones o revisadas de la Licencia de Documentación Libre GNU cada cierto tiempo. Tales versiones serán similares en espíritu a la presente versión, pero pueden diferir en detalles para solucionar problemas o intereses. Vea `http://www.gnu.org/copyleft/`

Cada versión de la Licencia tiene un número de versión que la distingue. Si el Documento especifica que una versión numerada particularmente de esta licencia o "cualquier versión posterior" se aplica a ésta, tiene la opción de seguir los términos y condiciones de la versión especificada o cualquiera posterior que haya sido publicada(no como un borrador) por la Free Software Foundation. Si el Documento no especifica un número de versión de esta Licencia, puede escoger cualquier versión que haya sido publicada (no como un borrador) por la Free Software Foundation.

B.11. Addendum

Para usar esta licencia en un documento que usted haya escrito, incluya una copia de la Licencia en el documento y ponga el siguiente derecho de autor y nota de licencia justo después del título de la página:

> ©Año Su Nombre.
>
> Permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre GNU, Versión 1.1 o cualquier otra versión posterior publicada por la Free Software Foundation; con las Secciones Invariantes siendo LISTE SUS TÍTULOS, siendo LISTE el texto de la Cubierta Frontal, y siendo LISTELO el texto de la Cubierta Posterior.
>
> Se incluye una copia de la licencia en la sección titulada "Licencia de Documentación Libre GNU".

Si no tiene Secciones Invariantes, escriba "Sin Secciones Invariantes" en vez de decir cuáles son invariantes. Si no tiene Texto de Cubierta Frontal, escriba "Sin Texto de Cubierta Frontal" en vez de"siendo LÍSTE el texto de la Cubierta Frontal"; así como para la Cubierta Posterior.

Si su documento contiene ejemplos de código de programa no triviales, recomendamos liberar estos ejemplos en paralelo bajo su elección de licencia de software libre, tal como la Licencia de Público General GNU, para permitir su uso en software libre.

Índice alfabético

Bibliografía

[1] D. P. Carlisle. Packages in the "graphics" bundle. Se incluye en el conjunto "graphics" como `grfguide.tex`, disponible en el mismo sitio de donde se ha tomado la distribución de LaTeX.

[2] Jeffrey E. F. Friedl. *Mastering Regular Expresions.* OŔeilly & Associates, Inc.

[3] Donald E. Knuth. *The TeXbook, Tomo A de Computers and Typesetting.* Addison-Wesley Publishing Company, 1984. ISBN: 0-201-13448-9.

[4] Leslie Lamport. LaTeX: *A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, segunda edición edition, 1994. ISBN: 0-201-52983-1.

[5] local. *Cada instalación de LaTeX debería proporcionar la llamada* Guía Local de LaTeX, *que explica las cosas que son particulares del sistema local. Debería residir en un fichero llamado* `local.tex`. *Por desgracia, en algunos sitios no se halla dicha guía. En este caso, pídale ayuda a un experto de LaTeX.*

[6] Frank Mittelbach Michel Goossens and Alexander Samarin. The LaTeX Companion. Addison-Wesley, Reading, Massachusetts, 1994. ISBN: 0-201-54199-8.