Francisco Solsona y Elisa Viso
(coordinadores)

Manual de supervivencia en Linux

Mauricio Aldazosa José Galaviz Iván Hernández Canek Peláez
Karla Ramírez Fernanda Sánchez Puig Francisco Solsona
Manuel Sugawara Arturo Vázquez Elisa Viso

Facultad de Ciencias, UNAM
2007

las prensas de ciencias

Esta obra aparece gracias al apoyo
del proyecto PAPIME PE-100205

Manual de supervivencia en Linux
1ª edición, 2007

Diseño de portada: Laura Uribe

Facultad de Ciencias
Circuito exterior. Ciudad Universitaria.
México 04510
cse@fciencias.unam.mx

ISBN: 978-970-32-5040-0

Impreso y hecho en México

Índice general

Índice de figuras

Índice de tablas

Prefacio

Este libro presenta las aplicaciones y el enfoque que un estudiante de computación o un futuro programador debe dominar para sacar el mayor provecho de su enseñanza profesional en programación y, en general, en las ciencias de la computación. Hemos vertido aquí la experiencia de media docena de años impartiendo cursos propedéuticos de *supervivencia* para estudiantes de la licenciatura en ciencias de la computación de la Facultad de Ciencias de la Universidad Nacional Autónoma de México.

Convenciones tipográficas

Tabla 1 Convenciones tipográficas

Entidad	Descripción
Enfatizado	Utilizamos este tipo de letra (*font*) para enfatizar o resaltar palabras o frases.
itálicas	Las utilizamos para modismos, palabras o frases en inglés y palabras utilizadas en el medio y cuya traducción o aceptación en el idioma no es clara.
`comando`	Los comandos y opciones se muestran así. Cuando encuentres líneas tales como: `% comando`, éstas representan comandos que puedes teclear y ejecutar en una terminal. Por convención utilizaremos `%` para significar que debe ejecutarse como un usuario no-priviligiado y `$` cuando debe ejecutarse como súper-usuario o root.
`enunciados`	Utilizaremos este tipo de letra para escribir código que debe ir en un archivo.
	Este símbolo es el logo del ambiente Gnome y sirve para señalar qué aplicaciones para este ambiente son equivalentes a la revisada en la sección donde aparece.

Continúa en la siguiente página

Tabla 1 Convenciones tipográficas *Continúa de la página anterior*

Entidad	Descripción
	Este símbolo es el logo del ambiente KDE y sirve para señalar aplicaciones alternativas o similares a la revisada en la sección donde aparece.
	Este símbolo es el logo del editor Emacs y sirve para señalar paquetes alternativos o similares al revisado en la sección donde aparece.

A quién está dirigido

Este texto está dirigido a estudiantes y profesionales de la computación o futuros programadores que estén interesados en incursionar en el mundo del software libre, conocer el sistema operativo Linux y que requieran dominar rápidamente herramientas orientadas al desarrollo y productividad en este ambiente.

Es un libro práctico, por lo que es importante resaltar que todo lo que se ve en los distintos capítulos puede repetirse en un sistema de cómputo con sistema operativo Linux. Asimismo, los ejercicios en algunas de las secciones son de utilidad para reafirmar el conocimiento y se recomienda realizarlos en el orden presentado.

Distribución y ambientes

Algunas de estas notas muestran comandos

A lo largo del libro encontrarás muchos ejemplos que funcionarán bien en cualquier distribución de Linux. Sin embargo, y a pesar de nuestros mejores esfuerzos, hemos incluido aplicaciones que no son parte esencial de las distribuciones de Linux más utilizadas. Cada vez que esto sucede te mostramos, en una nota al margen, el comando que debes ejecutar como súper-usuario para instalar dicha aplicación.

Para lograr esto, por supuesto, nos hemos visto en la necesidad de utilizar una distribución y optamos por Ubuntu y como administrador de paquetes utilizamos apt. Por ejemplo, verás en notas al margen algo parecido a esto:

```
% smart install kate
```

lo que constituye (como verás en el Capítulo 1) una línea de comandos que, al ser ejecutada, se conecta a la red, localiza la aplicación kate, la descarga a tu computadora y la instala.

En el apéndice A se presenta una relación de algunas de las distribuciones más importantes en el mercado al momento de la edición de este libro. En dicho anexo encontrarás,

entre otros datos interesantes de cada distribución, la manera preferida de administrar aplicaciones.

En el apéndice B se presentan las opciones que se tienen para instalar paquetes o funciones de tu distribución favorita. Podrás ver ahí cómo compilar para instalar algunos de los servicios que se presentan en este libro.

Finalmente, en el apéndice C colocamos los archivos principales para tu archivo .emacs que fuimos construyendo a lo largo de este material, así como los archivos necesarios para correo e internet.

Introducción a Unix 1

1.1. Historia

La primera versión de UNIX, llamada *Unics*, fue escrita en 1969 por *Ken Thompson*. Corría en una computadora PDP-7 de Digital. Más adelante, los laboratorios Bell, el MIT y General Electric se involucraron en la creación del sistema Multics, grande en tamaño, y el primer sistema de tiempo compartido que incluía muchas ideas innovadoras acerca de sistemas operativos. Thompson y algunos de sus colegas admiraban las capacidades de Multics; sin embargo, pensaban que era demasiado complicado. Su idea era demostrar que era posible construir un sistema operativo que proporcionara un ambiente de desarrollo cómodo de una forma más simple. Y afortunadamente, tuvieron un éxito admirable en el desarrollo de esta idea; a pesar de esto, UNIX es irónicamente mucho más complicado de lo que Multics nunca llegó a ser.

En 1970 Thompson, junto con *Dennis Ritchie*, portó UNIX a la PDP-11/20. Ritchie diseñó y escribió el primer compilador de *C* para proveer un lenguaje que pudiera ser usado para escribir una versión portátil del sistema. En 1973, Ritchie y Thompson reescribieron el kernel de UNIX, el corazón del sistema operativo, en C.

Inicialmente se otorgaron licencias gratuitas para utilizar UNIX a universidades con propósitos meramente educativos (en 1974). Esta versión fue la quinta edición (las ediciones hacen referencia a las ediciones del manual de referencia de UNIX). La sexta edición fue liberada en 1975; sin embargo, fue hasta la séptima edición, liberada por los laboratorios Bell en 1979, donde se logró la meta deseada: *portabilidad*; esta edición fue la que sirvió como punto de partida para la generación de este nuevo y maravilloso mundo: el mundo UNIX. Ésta es considerada la edición clásica de UNIX. Las dos vertientes más

fuertes creadas a partir de esta edición de UNIX son: System V (no confundirlo con la quinta edición) y el sistema BSD (Berkeley Software Distribution).

A pesar de que ambos sistemas surgieron de la misma influencia y comparten muchas características, los dos sistemas se desarrollaron con personalidades muy distintas. System V es más conservador y sólido; mientras que BSD es más innovador y experimental. System V era un sistema comercial, mientras que BSD no.

La historia es mucho más detallada y aún hay mucho que contar acerca del desarrollo de cada una de estas vertientes, de las cuales surgieron todavía muchas vertientes más de UNIX.

En el caso de la licenciatura en Ciencias de la Computación en la Facultad de Ciencias, como en muchas otras universidades de prestigio, el servicio al que acceden los estudiantes y la mayoría de los profesores es utilizando una implementación de UNIX, Linux, con computadoras conectadas en una red, por lo que todo lo que sucede entre los estudiantes y la computadora será en el contexto de esta red.

1.1.1. Sistemas UNIX libres

Los sistemas compatibles UNIX libres[1]para la arquitectura i386, notablemente Linux y FreeBSD, han hecho de UNIX un sistema al alcance de cualquiera. Estos sistemas proveen excelente rendimiento y vienen con un conjunto de características y herramientas estándar de UNIX. Son tan robustos que incluso muchas empresas han basado su desarrollo en alguno de estos sistemas.

Dado que estos sistemas no contienen software propietario pueden ser usados, copiados y distribuidos por cualquiera (incluso gratis). Las copias de estos sistemas se pueden conseguir en CD-ROM o bien por medio de Internet.

Linux. Linux es una versión de UNIX libre, originada por Linus Torvalds en la Universidad de Helsinki en Finlandia. Fue inspirado por Minix – escrito por Andrew Tanenbaum – y sigue más o menos las convenciones de System V. Su desarrollo ha sido llevado a cabo por programadores (*hackers, wizards*) de todo el mundo, coordinados por Linus a través de Internet. Una gran cantidad de código incluida en el "sistema operativo Linux"[2]es GNU y la versión de X es XFree86.

Hay varias distribuciones de Linux y algunas de las más importantes son:

- Ubuntu: http://www.ubuntu.com/ (y Kubuntu, la distribución que utilizamos en este trabajo: http://www.kubuntu.org/)
- Fedora: http://fedora.redhat.com/

[1]En este contexto *libre*, es una traducción literal de *free* que no significa *gratis*, (como en cervezas gratis), sino significa *libre* (como en libertad de expresión).

[2]Un *sistema operativo* es un conjunto de programas que controla y administra los recursos de la computadora, como son el disco, la memoria, los dispositivos de entrada y salida.

- Debian: http://www.debian.org/
- openSUSE: http://www.opensuse.org/
- FreeSpire: http://freespire.org/

Linux se distribuye bajo la licencia GPL (*G*eneral *P*ublic *L*icence), que es conocida como *CopyLeft* y que representa gran parte de la idiosincrasia del mundo UNIX libre.

FreeBSD. FreeBSD es derivado del sistema BSD (de la versión 4.4 para ser exactos); fue creado, igual que Linux, para la arquitectura i386. Su idea fundamental de desarrollo es tener un ambiente estable de desarrollo para esa arquitectura. Está soportado por un grupo internacional de voluntarios. La gran mayoría de los programas que conforman los sistemas FreeBSD, a diferencia de Linux, están gobernados por licencias estilo Berkeley, las cuales permiten redistribuciones siempre y cuando el código incluya una nota reconociendo el derecho de autor de *Regents of the University of California and the FreeBSD project*. Algunas otras partes de FreeBSD son GNU y están cubiertas por la GPL.

La pregunta más común entre usuarios de UNIX en el mundo es: ¿cuál es mejor: Linux o FreeBSD? La respuesta no es clara. Ambos tienen cosas buenas y tal vez la mejor forma de saberlo es probar ambos.

1.1.2. El proyecto GNU

El proyecto GNU buscaba desarrollar un sistema compatible con UNIX, totalmente libre[3]. Es operado por la FSF (*F*ree *S*oftware *F*oundation), organización fundada por *Richard Stallman*, quien ha escrito gran parte del código de GNU. GNU es una anagrama de *"GNU's not UNIX"*. Algunas herramientas invaluables para el trabajo diario de un estudiante de Ciencias de la Computación (al menos en esta Universidad) que han sido desarrolladas dentro del proyecto GNU son: un editor de texto (Emacs), el compilador de C (gcc), un depurador (gdb), y versiones de prácticamente todas las herramientas estándar de UNIX.

Todo el software distribuido por GNU se libera bajo la licencia GPL. De acuerdo con los términos de esta licencia, cualquiera puede modificar el software y redistribuir su propia versión. La restricción principal es que te obliga a redistribuirlo incluyendo el código fuente, aún en el caso de que tú hayas hecho las modificaciones.

UNIX es un sistema operativo multi-usuario y multi-tarea, es decir, permite a muchos usuarios conectarse al sistema y cada uno de estos usuarios puede ejecutar varios programas a la vez.

[3] A varios años de haber sido creada, lo han logrado. Su nombre es *Hurd* y promete muchas cosas interesantes en el mundo de los sistemas operativos. Sin embargo, Linux, (que sigue mucha de la filosofía de GNU, pero no es el proyecto GNU), le lleva mucha ventaja. Sin duda alguna ésta es una época interesante para estar metido en el software libre: nos esperan cosas buenas.

1.2. Sistemas de tiempo compartido

UNIX es un sistema de tiempo compartido. Esto significa que las computadoras que trabajan con UNIX pueden aceptar más de un usuario al mismo tiempo. La computadora principal (la parte del hardware que ejecuta realmente la mayoría del trabajo) recibe el nombre de computadora anfitrión. Una de las tareas del sistema operativo es asegurar que los recursos de la computadora anfitrión se compartan entre todos los usuarios.

Para trabajar con UNIX normalmente usarás una terminal. Una terminal UNIX básica consta de una pantalla, un teclado y algunas cosas más. No obstante, como se verá más adelante algunas terminales son más complejas que otras.

Cada vez que oprimes una tecla, la terminal envía una señal a la computadora anfitrión. Como respuesta, esta última envía su propia señal de regreso a la terminal, indicándole que despliegue el carácter correspondiente en la pantalla. UNIX fue configurado de tal forma que cuando la computadora anfitrión efectúa el eco, puedes ver que lo que teclea se recibe bien y que la conexión entre la computadora anfitrión y la terminal está intacta.

1.2.1. Los sistemas multiusuario

Muchas computadoras anfitrión están conectadas directamente a las terminales. No obstante, hay muchos centros de cómputo que ofrecen una variedad de computadoras anfitrión. En tales casos, la terminal podría estar conectada a una computadora especial, denominada servidor.

Casi todas las computadoras anfitrión cuentan con un teclado y una pantalla, que son parte de la computadora. Para UNIX, teclado y pantalla son sólo otra terminal. Sin embargo, dichas partes reciben el nombre especial de *consola*.

Un sistema UNIX clásico puede usar una computadora anfitrión que se encuentre en la oficina del administrador del sistema. Dicha computadora podría estar conectada a una sala llena de terminales o computadoras al otro extremo del corredor o en otro piso e incluso en otro edificio.

Básicamente, todos los sistemas UNIX aceptan varios usuarios a la vez. No obstante, algunas computadoras UNIX sólo pueden ser usadas por una persona simultáneamenente y son mejor conocidas como *estaciones de trabajo*[4].

[4]Esto no significa que la estación de trabajo no tenga capacidades multiusuario o multitarea; la estación de trabajo es una máquina con capacidades limitadas tanto en espacio en disco como en velocidad de procesamiento, razón por la cual es conveniente que sólo una persona la utilice.

1.3. Inicio de sesión

Dentro de los sistemas UNIX, todos los usuarios del sistema, sea éste uniusuario o multiusuario, se identifican con un nombre de usuario, también conocido como su *login* o *logname*, que será provisto por el administrador del sistema. La clave de usuario generalmente está relacionada con el nombre y apellido del usuario. Dos usos comunes es la de dar al usuario la primera letra de su nombre junto con su apellido, o las tres iniciales de su nombre y dos apellidos. Por ejemplo, supongamos que el nombre completo de un usuario es *Juan Pérez* y su nombre de usuario podría ser *juan* o *jp*. En este libro, utilizaremos *juan* como tu nombre de usuario.

Para usar un sistema UNIX, en primer lugar debes iniciar una sesión con un nombre de usuario; el sistema pedirá este nombre por medio del programa login; deberás teclear la clave que te fue asignada, respetando minúsculas y mayúsculas[5]. Al terminar con los carácteres que componen tu clave, deberás oprimir la tecla Enter, para que el sistema reciba la clave y la procese. Supongamos que la clave de acceso es *juanPerez123*. La pantalla de la computadora se ve como en la figura 1.1, esperando a que le des la contraseña correcta.

Figura 1.1 Acceso al sistema.

```
Red Hat Linux release 6.2 (Zoot)
Kernel 2.2.14-5.0 on an i686
login: juan
Password:
```

En este punto del proceso de iniciar una sesión, deberás teclear la contraseña de acceso que tienes registrada en el sistema. Nota que, por seguridad, la contraseña no se muestra en la pantalla, por lo cual se deberá teclear con cuidado, también respetando mayúsculas y minúsculas. Si la clave de acceso corresponde con la registrada en el sistema, podrás iniciar una sesión.

Si Juan ingresa correctamente su contraseña, la pantalla de la computadora lucirá como en la figura 1.2, donde *estNum* es el nombre de la computadora en la que se está iniciando la sesión. La forma precisa de cómo aparece la combinación del nombre del usuario y el nombre de la máquina puede variar, pero usualmente aparecen en donde el sistema espera

[5] UNIX es sensible al uso de mayúsculas y minúsculas.

que le des instrucciones una vez iniciada la sesión.

Figura 1.2 Inicio de una sesión en un sistema UNIX.

```
Fedora Core release 2 (Tettnang)
Kernel 2.6.5-1.385 on an i686
login: juan
Password:
Last login: Tue Nov 28 11:20:54 from bar.fciencias.unam.mx
[juan@estNum jd]%
```

Es común equivocarse al teclear la contraseña. Si esto sucede, el sistema responderá con un mensaje de contraseña inválida y procederá a pedirte nuevamente el inicio de sesión, desde la clave de usuario:

Figura 1.3 Equivocación al teclear la contraseña.

```
Incorrect password.Try again
login:
```

Puedes corregir la contraseña mientras no hayas oprimido la tecla Enter, si oprimes simultáneamente las teclas Control–u[6], lo que borra todos los carácteres tecleados hasta ese momento y puedes volver a empezar a teclearla. Esto es conveniente por dos razones: la primera de ellas, para ahorrarte el tener que volver a teclear la clave de usuario y la segunda porque el sistema te dará un número máximo de oportunidades (generalmente entre tres y cinco) para entrar en sesión, por lo que es deseable evitar introducir mal la contraseña.

1.4. Emacs

1.4.1. Introducción a Emacs

Emacs es un editor de pantalla avanzado, auto–documentado, configurable, extensible y de tiempo real.

Decimos que Emacs es un *editor de pantalla* porque normalmente el texto que queremos editar es visible en la pantalla y se actualiza automáticamente mientras tecleas los comandos; es por esto último que decimos que es de *tiempo real*. El nombre "Emacs" fue escogido como abreviatura de "Editor MACros".

[6] Usaremos en adelante la notación C-u, que deberá leerse como *control u*, para denotar la combinación de la tecla Control oprimida al mismo tiempo que u.

Decimos que es *avanzado* porque tiene muchas más funciones de las que comúnmente encontramos en editores de pantalla, como son control de subprocesos, facilidades para la edición de archivos fuente de varios lenguajes de programación, así como una gran cantidad de utilerías para hacer nuestra vida más fácil.

C-h es Control *junto con* h

El que sea *auto-documentado* significa que en cualquier momento puedes teclear una secuencia especial, que es `C-h`, y saber cuáles son tus opciones; también puedes encontrar qué hace cualquier comando o encontrar documentación específica de un tema.

El que sea *configurable* significa que puedes cambiar las definiciones de los comandos de Emacs, es decir, controlar su comportamiento para que se adapte a tus necesidades o tus preferencias.

Extensible significa que puedes ir más allá de la simple configuración y escribir comandos completamente nuevos, usando un lenguaje de programación llamado Emacs-Lisp, del cual también aprenderás algo en estas notas en la Sección 9.2.

1.4.2. Emacs y tú

Si hay algo que debes aprender y atesorar de este libro, es Emacs. Una situación que le ocurre a todas las aplicaciones de software en Linux y en prácticamente cualquier sistema operativo es que pasan de moda, pierden vigencia rápidamente. Es probable incluso que para el momento que leas algunos capítulos de este trabajo, las aplicaciones aquí revisadas hayan sido reemplazadas por otras o que existan versiones nuevas y, con alta probabilidad por desgracia, serán distintas.

Emacs tiene una historia de muchos años, más de tres décadas en circulación, y algo importante es que no ha cambiado significativamente y, estamos seguros, será el caso en este momento que tú te inicias en Emacs. Esto es importante porque la inversión intelectual que realizas al aprender y dominar este editor te redituará por muchos años.

Por este motivo, a lo largo de todo el libro, encontrarás una sección titulada así: Emacs. En cada una de estas secciones te mostraremos qué hace tan interesante a este editor y descubriremos juntos muchas de las tareas que pueden atenderse a través de Emacs, por ejemplo: editar diversos archivos, programar en distintos lenguajes, acceder a tu correo electrónico, entre otras acciones y todo esto al mismo tiempo.

Si bien es cierto que puedes ejecutar distintas aplicaciones (explicadas en este mismo libro) para realizar muchas tareas de manera simultánea, cuando utilizas Emacs para realizar algunas o todas esas tareas, obtienes ciertas ventajas: memoria compartida, un excelente sistema de respaldo para copiar, pegar o reordenar datos, entre otras ventajas. Tu productividad se incrementará notablemente, te lo garantizamos.

Ambientes gráficos en Linux 2

2.1. Ambientes de escritorio

Un escritorio ofrece una interfaz gráfica de usuario[1] para interactuar con la computadora. La otra opción para interactuar con la computadora es mediante una interfaz de línea de comandos[2]. Para la mayoría de las personas el escritorio más común es Windows de Microsoft, para otras cuantas MAC OS. En Linux se tienen muchos escritorios que puedes utilizar en tu computadora. La base de todos estos escritorios es el *sistema de ventanas X*, que es un sistema que provee los elementos básicos para que se dibujen las ventanas y se interactúe con ellas. La interacción más común entre el servidor de X y el usuario es cuando el cliente está corriendo en la misma computadora que el servidor. Esto es lo que ocurre cuando inicias tu computadora de forma normal. Existen otras posibilidades como exportar todo tu escritorio desde otra computadora o exportar algunas ventanas.

Tomando como base a X, existen varios escritorios para Linux. Como en la mayoría de las aplicaciones en UNIX, existen más de una para realizar la misma tarea. Algunos de los más comunes son:

- KDE
- GNOME

[1] GUI, del inglés *Graphic User Interface*
[2] CLI, del inglés, *Command Line Interface*

- Enlightenment
- Xfce

2.2. KDE

KDE es un escritorio libre para estaciones de trabajo UNIX. La motivación de los desarrolladores es tener un escritorio para UNIX con el que sea fácil trabajar, y que tenga una aplicación equivalente a las disponibles en los sistemas operativos propietarios. KDE hace uso de QT, una caja de herramientas (*toolkit*) gráfico de la compañia Trolltech. Cuando se tomó esta decisión en 1996, por el autor Matthias Ettrich[3], hubo varias críticas por esa decisión. El problema es que en esas fechas QT no tenía una licencia de software libre y a la gente del proyecto GNU le preocupaba que se usara algo así para crear un escritorio que quería ser libre. Dada esta situación se inició otro proyecto, GNOME, que no haría uso de QT y sería totalmente libre. En Noviembre de 1998 Trolltech creó una nueva licencia *Q Public License* bajo la cual distribuía a QT. Finalmente en Septiembre del 2000, Trolltech liberó la versión de QT para UNIX bajo la GPL, además de la QPL, por lo que ahora KDE es totalmente libre.

2.2.1. Configuración de KDE

El comportamiento de KDE lo puedes configurar mediante el uso del centro de control de KDE, el cual puedes ejecutar mediante el comando kcontrol. Para ejecutar una aplicación sin tener que hacer uso de la consola puedes utilizar el atajo por omisión Alt+F2. En la figura 2.1 se muestra la ventana inicial del centro de control, dividida en 2 secciones. En la sección 1 se muestran las categorías mediante las cuales se puede configurar a KDE. En la sección 2 se presenta la información pertinente al módulo que se está configurando. Al iniciar, KControl nos presenta en esta sección cierta información elemental:

1. Versión de KDE
2. Nombre del sistema
3. Nombre del usuario
4. Versión del núcleo del sistema

[3] El correo que inició todo lo puedes ver en
http://groups.google.com/group/de.comp.os.linux.misc/msg/cb4b2d67ffc3ffce

Figura 2.1 KControl

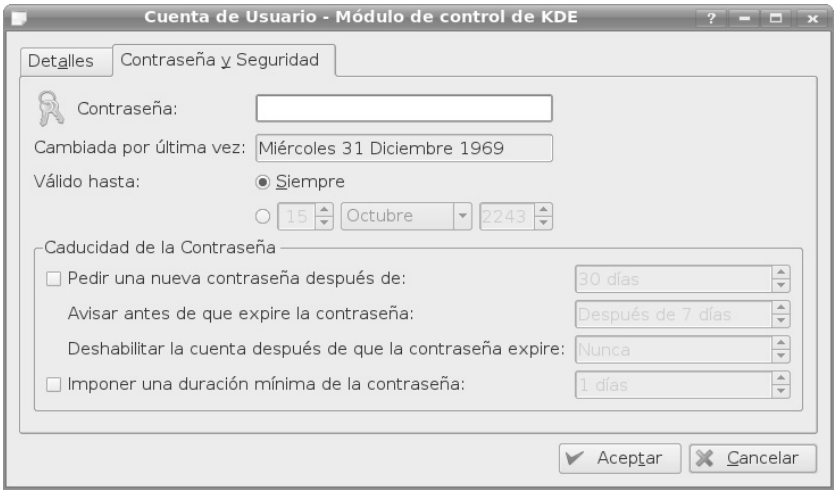
Administración de usuarios

En la categoría *Administración del sistema* se encuentra el módulo *Administración de usuarios*. Desde ahí es posible revisar los detalles de los usuarios existentes y agregar nuevos usuarios. Cuando se intenta modificar la configuración del sistema es necesario contar con los permisos adecuados y presionar el botón *Modo administrador*; de no ser así sólo puedes modificar los datos que pertenecen a tu propia cuenta. Para agregar un nuevo usuario es necesario el botón *Nuevo*. El centro de control nos presentará el diálogo que se muestra en la figura 2.2. En él es necesario ingresar los datos que corresponden al nuevo usuario entre los que se encuentran el nombre de usuario, su grupo primario, los grupos a los que pertenece, su directorio personal y el intérprete que va a utilizar.

Como se ve en la figura 2.3 en la pestaña *Contraseña y Seguridad* se pueden establecer diferentes opciones para la caducidad de la contraseña. Puedes decidir si es que la contraseña caduca o no, y cuáles son las acciones que el sistema debe tomar en caso de que esto ocurra.

Figura 2.2 Administración de usuarios

Figura 2.3 Administración de contraseñas



Escritorios múltiples

Accesos rápidos

En la categoría *Regional y Accesibilidad*, se encuentra el módulo *Accesos rápidos de teclado*. Mediante ellos es posible asociar una acción a una combinación de teclas. Es de esa forma que al presionar las teclas Alt+Tab cambias de una ventana a otra. Existen varias combinaciones predeterminadas como Alt+F4 para cerrar una ventana. Desde aquí puedes crear combinaciones de teclas para ejecutar tus acciones más comunes sin tener que separar las manos del teclado.

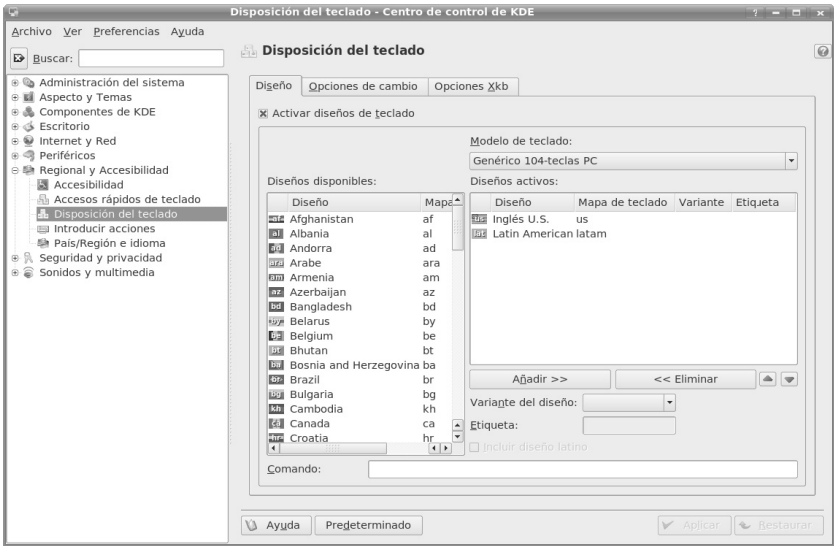
Internacionalización

Hay ocasiones en las que puedes desear tener más de una idioma configurado en tu sistema. De esa forma puedes hacer que KDE despliegue todos sus mensajes en el idioma de tu preferencia. Para poder hacer eso es necesario que instales el paquete de internacionalización de KDE que corresponde al idioma que quieres utilizar. El nombre de estos paquetes es de la forma *kde-i18n-XX* en donde XX es el código de dos letras para el idioma que quieres instalar. Para español el nombre del archivo es kde-i18n-es mientras que para italiano es kde-i18n-it. Para instalar dicho paquete es necesario que ejecutes apt-get install kde-i18n-es. Una vez hecho eso puedes regresar al centro de control y dentro de la categoría *Regional y Accesibilidad*, en el módulo *País/Región e idioma* podrás agregar el idioma que acabas de instalar.

Disposición del teclado

Además de cambiar el idioma con el que se presenta KDE, también puedes cambiar la disposición del teclado. De esa forma es posible que tu teclado se comporte como si fuera un teclado en un idioma distinto. Para hacer eso es necesario que vayas a la categoría *Regional y Accesibilidad* y ahí vayas al módulo *Disposición del teclado*, tal como se muestra en la figura 2.4. Deberás seleccionar la opción *Activar diseños de teclado* y luego seleccionar los idiomas que deseas en tu teclado. En la pestaña *Opciones de cambio* puedes seleccionar cómo quieres que KDE te muestre la bandera que corresponde al teclado que estás utilizando. Además puedes seleccionar si cada vez que cambias la disposición del teclado afecta a todas las aplicaciones o sólo a la aplicación que está activa en el momento en el que realizas el cambio. Para cambiar la disposición del teclado de forma rápida puedes usar el atajo por omisión de KDE que es CTRL+ALT+K; o si seleccionaste la opción para mostrar la bandera, puedes hacer click sobre ella.

Figura 2.4 Disposición del teclado



2.2.2. KDE en 1, 2, 3

Cuando acabas de entrar a KDE, verás el escritorio que se muestra en la figura 2.5. Ahí hemos resaltado algunas secciones, en la sección 1 se resalta el *Menú K*, a partir de él puedes acceder a las aplicaciones que estén instaladas en tu sistema. En la sección 2 tienes accesos rápidos a aplicaciones utilizadas frecuentemente. En la sección 3 se muestra el navegador de los escritorios virtuales (que veremos un poco más adelante). En 4 se muestran las aplicaciones que estén ejecutándose en un momento dado. Finalmente en 5 puedes ver applets de aplicaciones que están ejecutándose en el background. Veamos ahora cómo realizar algunas de las tareas comunes en KDE.

2.2.3. Quemado de discos con K3b

Desde hace algunos años la forma de transmitir información en un medio físico ha cambiado radicalmente; las computadoras actuales ya no tienen bahías para discos flexibles y, aunque en los últimos años las memorias flash son cada vez más populares, los CD's prevalecen como un medio útil. Dentro de KDE la aplicación para quemar discos es K3b, este

provee al usuario de una interfaz sencilla para poder quemar CD o DVD. Hay que señalar que K3b es sólo la interfaz gráfica ya que el trabajo real lo llevan a cabo las aplicaciones cdrecord, cdrdao y growisofs. En la figura 2.6 se muestra la ventana inicial de K3b.

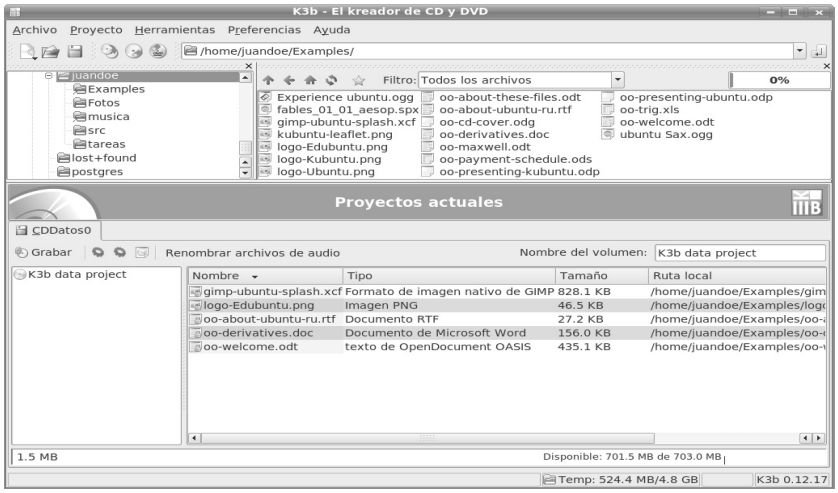
Figura 2.5 El escritorio

Quemado de un disco de datos

Al seleccionar la opción *Nuevo proyecto de CD de Datos*, K3b te presenta con una interfaz para navegar el sistema de archivos y seleccionar cuáles archivos son los que deseas incluir en tu CD. Para agregar un archivos nuevo a la compilación sólo es necesario encontrarlos en el navegador que está en la parte superior y seleccionarlo, como se muestra en la figura 2.7. Otra opción es arrastrarlos desde Konqueror o desde el escritorio hacia la parte baja de K3b.

Figura 2.6 K3b

Figura 2.7 Quemado de un disco de datos



Cuando ya seleccionaste todo lo que deseas agregar al CD, basta presionar el botón *Grabar* para seleccionar las opciones con las que deseas crear el CD. Las pestañas más im-

portantes son *Grabando* y *Descripción del volumen*. En la primera es necesario seleccionar cuál dispositivo se va a usar para quemar el disco y a qué velocidad, qué modo de grabación y qué opciones queremos emplear. En los modos de grabación puedes seleccionar uno de los siguientes:

DAO	Disc at once, copia todo el disco en una sola pasada.
TAO	Track at once, genera pausas entre cada una de las pistas del disco.
En bruto	No procesa los datos (útil para copiar imágenes).

Entre las opciones que puedes seleccionar están:

Simular No lleva a cabo la copia, sólo la simula, para ver si no hay problemas.

Al vuelo Copia los datos directamente, sin generar una imagen intermedia.

Sólo crear imagen En lugar de quemar el disco crea una imagen, para ser quemada posteriormente.

Eliminar imagen Si se creó la imagen para quemarla, se puede eliminar una vez completado el proceso.

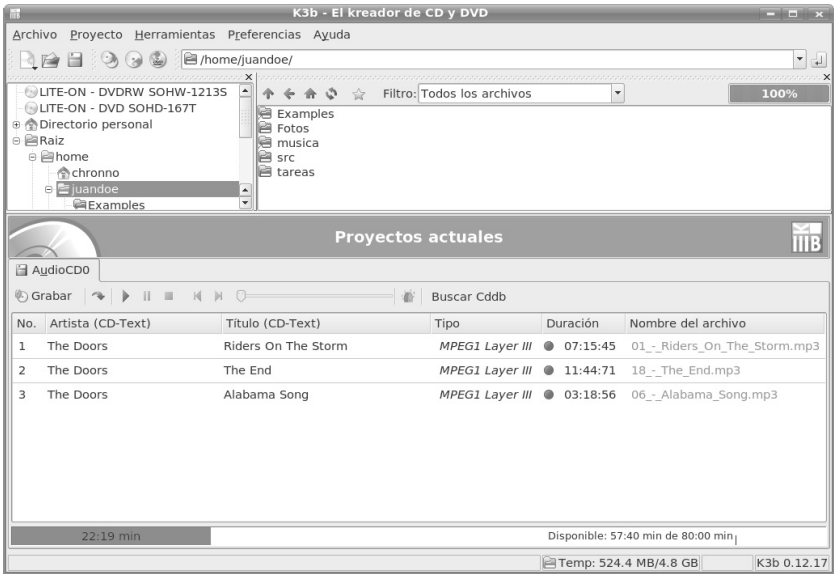
Verificar los datos grabados Una vez terminado el proceso de quemado, se comparan los datos para ver que hayan sido copiados exitosamente.

En la pestaña *Descripción del volumen* puedes ingresar los datos con los que quieres que se identifique al disco. Puedes ingresar la etiqueta del disco, el nombre de la persona que preparó los datos, el nombre de la persona que los editó, en qué sistema operativo se realizó la copia y con qué aplicación.

Quemado de un disco de música

Al seleccionar la opción *Nuevo proyecto de CD de audio*, K3b te presenta una interfaz similar a la utilizada para quemar datos; la diferencia es cómo presenta la información que quieres quemar en el disco. Ahora presenta cada uno de los tracks que quieres grabar y la duración de cada uno de ellos, como se muestra en la figura 2.8. Para poder utilizar archivos mp3 dentro de k3b es necesario instalar una biblioteca adicional. Debes instalar el paquete libk3b2-mp3, y lo haces con el comando sudo apt-get install libk3b2-mp3. El resto de las opciones para grabar son similares a las presentadas cuando quemas un disco de datos.

Figura 2.8 Quemado de un disco de audio



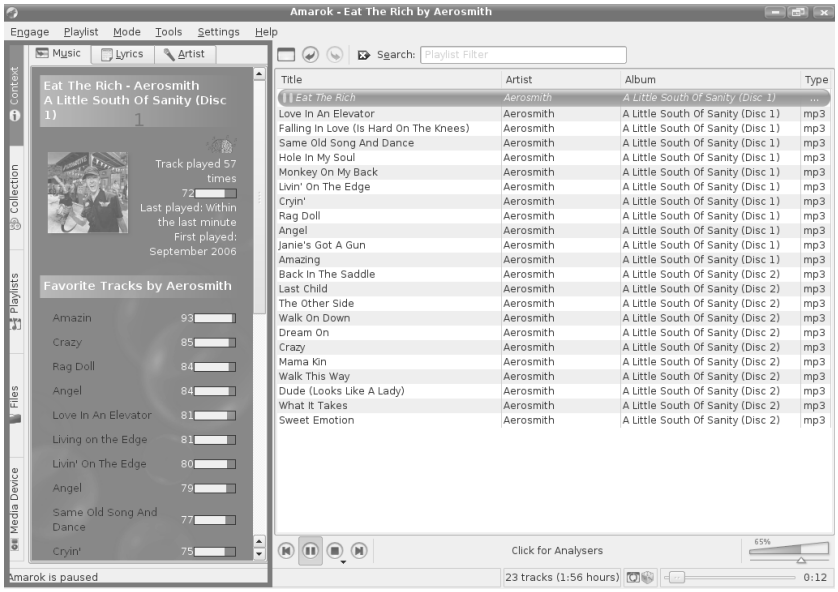
2.2.4. Amarok

Además de usar la computadora para trabajar o para jugar, una de las tareas más comunes es la de escuchar música. Para poder hacer eso dentro de KDE puedes usar Amarok. Amarok se encarga de tener un solo lugar en donde puedes escuchar música de CD, archivos .mp3, archivos .ogg e incluso flujos desde internet. Cuando Amarok inicia por primera vez se ve como en la figura 2.9. Para agregar archivos que desees escuchar es necesario que vayas al menú *Lista de reproducción* y selecciones el elemento *Añadir medio*. Después deberás navegar el sistema de archivos hasta que encuentres los archivos que quieres agregar.

Cuando Amarok inicia por primera vez te pedirá que crees una colección de música; si aceptas te presentará una vista en la que puedes construir tu colección de música, decidiendo en qué directorios deseas buscar. Ésta es una parte esencial para que puedas utilizar todos los beneficios que te ofrece Amarok. Cuando Amarok termine de generar tu colección podrás agregar cualquier canción que ahí aparezca a la lista de música que deseas escuchar[4].

[4]Si cuando intentas escuchar la música te hace falta alguna biblioteca, Amarok te lo indicará y la instalará por ti

Figura 2.9 Amarok



Navegadores

Los navegadores en Amarok te permiten acceder a tus canciones de distintas formas. Éstos se encuentran en la parte izquierda de Amarok, como se muestra en la sección 1 de la figura 2.9.

Contexto Aquí Amarok te presenta información acerca de la canción que estás escuchando en ese momento. Puedes navegarla y revisar la letra de la canción, el disco en donde aparece o la biografía del artista.

Colección Te permite navegar tu colección de música.

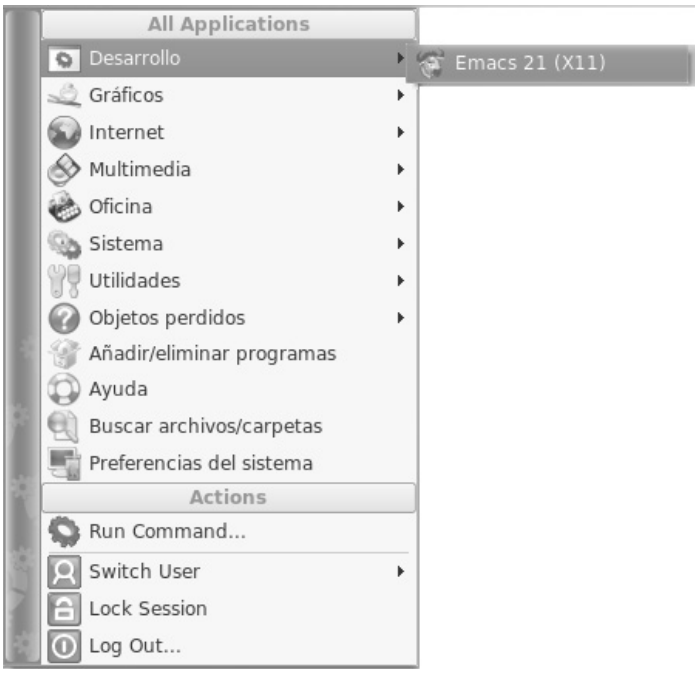
Listas de reproducción Aquí puedes administrar las diferentes listas de música, para que puedas cargar la música que quieres de forma rápida.

Dispositivo de medios Si conectas un reproductor de MP3 a tu computadora, desde aquí puedes hacer que Amarok toque esa música en tu computadora.

2.3. Emacs

Cada vez, a partir de este capítulo, que te encuentres con una sección titulada *Emacs*, te recomendamos iniciar el editor y probar todo lo que te vamos explicando. La mayoría de las secciones sobre Emacs te muestran una faceta completamente distinta; por ejemplo, Emacs como editor de texto (Sección 4.3); o cómo ejecutar comandos; o un intérprete de comandos de Unix dentro de Emacs (Sección 3.6). En esta sección simplemente te mostramos como iniciar Emacs desde el menú de KDE.

Figura 2.10 Emacs en el menú de KDE



2.4. Comenzando con Emacs

```
$ smart
install emacs
```

Primero que nada, debes saber cuál Emacs está instalado en tu sistema, porque hay

muchas implementaciones de editores parecidos a Emacs.

La versión más importante y reconocida de Emacs es la del proyecto GNU[5], la cual, junto con XEmacs[6] – otra versión muy popular y completa de Emacs – son las opciones más comunes hoy en día; si tu sistema operativo es algún Unix de distribución libre (Linux, FreeBSD, Hurd, etc.) muy probablemente ya tienes instalado uno o ambos editores.

El lugar usual para Emacs en el menú de KDE es en la sección de desarrollo, reafirmando que Emacs es una buena opción para desarrolladores y diseñadores de sistemas.

[5]Para mayor información sobre Emacs y el proyecto GNU, visita *http://www.gnu.org*.

[6]XEmacs es otra versión de Emacs. Su énfasis está en el soporte de interfaces gráficas de usuario. Algunos de los autores del presente trabajo usamos XEmacs y otros Emacs; sin embargo, prácticamente todo lo que veremos aquí puede ser usado, sin cambio alguno, en ambas versiones. Si quieres más información sobre XEmacs visita el sitio, *http://www.xemacs.org*.

Sistema de Archivos 3

3.1. El sistema de archivos

El sistema de archivos de Unix es la forma en la que los usuarios y el sistema operativo organizan su información. Éste consiste de un conjunto de archivos, cada uno de los cuales tiene un nombre llamado nombre de archivo. Hay tres clases de archivos:

- Archivos regulares, que contienen información.
- Directorios[1], que contienen un conjunto de archivos. Los conjuntos de archivos se identifican por el nombre del directorio que los agrupa.
- Archivos especiales de varias clases:
  - Archivos especiales de dispositivo, que proveen acceso a dispositivos tales como terminales o impresoras.
  - Archivos especiales FIFO (*First In First Out*, cola), algunas veces llamados *named pipes*, que proveen un canal para comunicación entre procesos independientes y que tienen un nombre asociado.

Como la mayoría de los sistemas operativos modernos, Unix organiza su sistema de archivos como una jerarquía de directorios. La jerarquía de directorios es un árbol (*tree*), pero dibujado como un árbol de cabeza o de lado. Un directorio especial, `root`, es la

[1] Los directorios también se conocen como carpetas.

raíz de la jerarquía. Más adelante veremos comandos para navegar y modificar el árbol de directorios.

Un directorio puede contener subdirectorios, archivos regulares y archivos especiales. Unix trata a los subdirectorios como tipos particulares de archivos. Si uno despliega los contenidos de un directorio, los subdirectorios se listan junto con los demás archivos.

Unix ve a un archivo, no importando su tipo, como una sucesión de bytes, almacenados en dispositivos externos como un disco duro, un diskette, un CD, un DVD o una unidad de almacenamiento USB. Los programas que usan al archivo identifican cualquier estructura interna que el archivo pueda tener.

Un nombre de archivo puede tener hasta 255 carácteres en la mayoría de los sistemas; antes se limitaba el tamaño de un nombre a 14 carácteres.

El nombre de un archivo puede contener cualquier carácter excepto '/' o el carácter nulo; sin embargo, algunos otros como '&' y ' ' (espacio en blanco) pueden causar problemas por sus significados especiales para su interpretación en la línea de comandos. Si tu sistema permite carácteres fuera del conjunto ASCII, tales como letras acentuadas (como nuestra letra *ñ*), también se puedes usarlas para nombrar archivos. Los nombres de archivos son sensibles a las mayúsculas y minúsculas, esto es, archivo y Archivo identifican a dos archivos distintos. Por convención, los archivos cuyo nombre comienza con un punto ('.'), llamados *dot files*, por lo general se usan como archivos de configuración para programas.

Algunos programas esperan que sus archivos de entrada y salida estén compuestos de un identificador seguido de un punto y luego una extensión. En general es una convención muy útil para el usuario utilizar extensiones en los nombres de sus archivos para identificarlos fácilmente. Por ejemplo:

txt	Se refiere a archivos de texto
jpg	Se refiere a imágenes en formato JPEG
java	Programas de `Java`
cc	Programas de `C++`
tex	Documentos para LaTeX
c	Programas de C

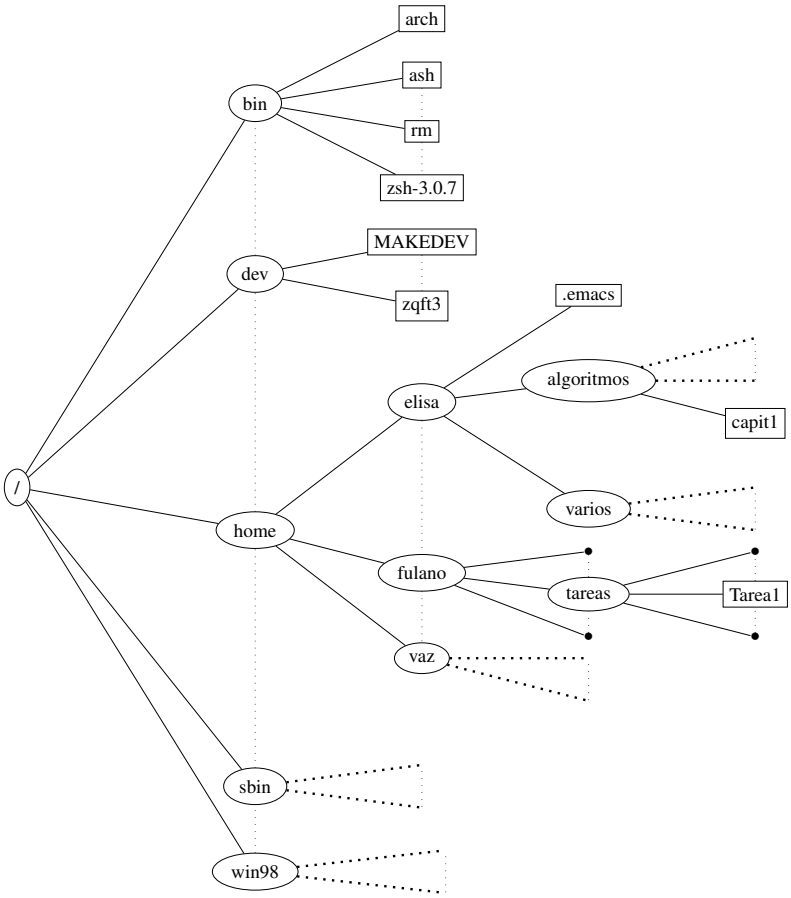
En el caso de los sistemas Unix, el número de carácteres en la extensión **no** está limitado, más que en cuanto a que el nombre total del archivo no puede tener más de 255 carácteres.

En Unix, como ya mencionamos y al igual que en otros sistemas operativos, los archivos se almacenan en una estructura jerárquica de árbol, cuyo primer nivel es el directorio raíz, el cual se denota por el carácter '/'. Dicho directorio almacena todos los archivos del sistema, que a su vez pueden ser directorios.

En los sistemas Unix, cada usuario tiene un directorio personal, en el cual almacena sus archivos. Dicho directorio se conoce como el *hogar (home)*. Una vez que entres en sesión, cada programa que se ejecuta se encuentra situado en un *directorio de trabajo*. El shell inicia, por omisión, en el directorio hogar del usuario. Un ejemplo simplificado de un

directorio en un sistema Unix se muestra en la figura 3.1. En esta figura encerraremos en óvalos a los directorios y en rectángulos a los archivos. Mostramos puntos suspensivos cuando suponemos que en ese lugar aparecen un número considerable de archivos y/o directorios, pero por falta de espacio no los mostramos.

Figura 3.1 Sistema de archivos en un sistema Unix



3.1.1. Rutas absolutas y relativas

Las *rutas* son expresiones que se usan para identificar un archivo o un conjunto de archivos. Se construyen bajando por el árbol del sistema de archivos, como el que se muestra en la figura 3.1.

Para referirse al directorio raíz se utiliza el carácter '/'; a partir del directorio raíz se puede descender en la jerarquía de directorios, siguiendo ramas del árbol, hasta llegar al archivo o directorio deseado, separando cada directorio con el carácter '/'. Por ejemplo, para referirse al archivo llamado ash que se encuentra en el directorio bin, podemos usar la ruta /bin/ash.

Cuando se hace referencia a un archivo o directorio a partir del directorio raíz utilizando una ruta, se dice que es una *ruta absoluta*. No es necesario referirse a un archivo o directorio a partir de la raíz, sino que también se puede hacer a partir del directorio de trabajo, es decir, en el directorio en el que el usuario se encuentra parado. Cuando una referencia a un archivo o directorio se hace a partir del directorio de trabajo, se dice que es una *ruta relativa*.

La forma de presentar una ruta relativa es similar a la usada por las rutas absolutas, con la diferencia de que al principio no se pone el símbolo /. Así, el descenso en la jerarquía por las ramas del árbol no se lleva a cabo a partir de la raíz sino a partir del directorio de trabajo. Por ejemplo, supón que tu clave de usuario es fulanito y tu directorio hogar es /home/fulanito. Al iniciar una sesión te encontrarás en este directorio. Supongamos que te quieres referir al archivo Tarea1 que está en el directorio tareas de tu propio directorio. Dicho archivo se representa usando la ruta absoluta

/home/fulanito/tareas/Tarea1

y usando una ruta relativa

tareas/Tarea1.

El directorio actual se denota por '.' y su padre se llama '..'. Haciendo uso de dicha notación puedes usar trayectorias que vayan hacia arriba o hacia abajo en la jerarquía de directorios. Siguiendo con el ejemplo anterior, si el directorio de trabajo es /home/fulanito/tareas, entonces la ruta relativa del archivo Ejemplo.java en el directorio algoritmos del usuario fulanito es

../algoritmos/Ejemplo.java

El significado de esta ruta es: Estando en el directorio tareas vamos al padre del mismo que es fulano utilizando '..'. De ahí bajamos por la rama algoritmos. Siguiendo esa ruta es que encontramos al archivo Ejemplo.java. A esta ruta relativa es equivalente la ruta absoluta

/home/fulano/algoritmos/Ejemplo.java

Nota que si el usuario se encontrara en otro lugar del árbol, la ruta relativa para alcanzar a Ejemplo.java sería otra. Es, precisamente, relativa al punto en el árbol en el que se encuentre el usuario.

Dado que el concepto de directorio base o *home* es tan importante, hay una notación especial para él, reconocida por cualquier intérprete de comandos de Unix excepto Bourne: el carácter '∼' al inicio de una trayectoria se refiere al directorio home del usuario actual, mientras que '∼vaz' hace referencia al directorio home del usuario vaz. También utilizaremos esa notación extensamente. Por ejemplo, para el usuario elisa, '∼/alguno.txt' se refiere al archivo /home/elisa/alguno.txt.

Una notación menos conveniente para referirte al directorio *home* es $HOME, que nos regresa el valor de la variable de ambiente HOME que cualquier intérprete de comandos, incluyendo Bourne, soporta. Por lo tanto, $HOME/alguno.c se refiere al archivo alguno.c en tu directorio *home*.

El *sistema de archivos* de Unix se puede dividir en distintos discos o particiones de un mismo disco o incluso en distintos medios de almacenamiento. El sistema de archivos se encarga de hacer esto transparente a los usuarios, haciendo parecer que todo vive en el mismo lugar. Existen distintas razones para hacer esto:

- Los discos y demás dispositivos que contienen los directorios y sus archivos pueden ser insuficientes para mantener la jerarquía completa, así que puede surgir la necesidad de distribuir la jerarquía en varios dispositivos.
- Para conveniencia en la administración, puede ser útil partir un disco muy grande en varias particiones.
- Los dispositivos de almacenamiento pueden no estar permanentemente unidos a la computadora. Un disco flexible puede contener su propia estructura de directorios. El sistema de archivos de red permite que si las computadoras están conectadas entre sí tengan un sistema de archivos compartido.

Unix acomoda los sistemas de archivos externos asociándoles *puntos de montaje*. Un punto de montaje es un directorio en un sistema de archivos que corresponde al directorio raíz del sistema de archivos externo. El sistema de archivos primario es el que emana del directorio raíz real y se llama '/'. Un sistema de archivos externo se liga al sistema primario por medio del comando mount. Generalmente, todo esto es transparentes para los usuarios comunes, así que no debes preocuparte demasiado al respecto, aunque es bueno saberlo.

3.2. Moviéndose en el árbol del sistema de archivos

Por omisión, el shell no muestra en qué directorio estás trabajando. Es conveniente poder preguntar cuál es el directorio de trabajo (o directorio actual) y esto se hace con el comando:

```
pwd
```

(*Print Working Directory*), a lo que el sistema responde con la ruta absoluta del directorio en el que te encuentras.

Para cambiarte de directorio puedes utilizar rutas relativas, si es que a donde te quieres mover está hacia abajo en el árbol desde el directorio de trabajo; o bien, utilizar una ruta absoluta para moverte arbitrariamente a cualquier otro directorio de trabajo. El comando que te permite cambiar de directorio de trabajo es

cd *[ruta]*

(*Change Directory*). Es importante mencionar que no te puedes mover a un archivo, sino únicamente a un directorio. Esto es porque al moverte, lo que se pretende es cambiar de directorio de trabajo.

Por ejemplo, si estando en el directorio de trabajo home quieres ubicarte en el directorio varios del usuario elisa, lo puedes hacer de las dos maneras siguientes:

```
cd elisa/varios/
cd /home/elisa/varios/
```

El primer comando usa una ruta relativa, mientras que el segundo comando usa una ruta absoluta.

Actividad 3.1 *Cámbiate a los siguientes directorios. Después de cada cambio, pregunta por el directorio de trabajo:*

(a) Directorio raíz
(b) Directorio sbin
(c) Directorio hogar con tu clave

Para listar el contenido de un directorio usaa:

ls *[parámetros] [ARCHIVOS]*

Ambos modificadores se pueden omitir. Los *[parámetros]* se refieren al formato que puedes querer para la lista de archivos, mientras que *[archivos]* se refiere al lugar en el árbol del sistema de archivos del que quieres la lista. Si omites los *[parámetros]* la lista que se muestre será breve, sin mayor información más que el nombre del archivo o de los archivos del directorio de trabajo. Dos de los parámetros más comunes se refieren a pedir una lista extendida, que contenga, entre otra información, el tipo de archivo, su dueño, su tamaño, el día de su última modificación. Puedes querer que se listen también aquellos archivos que empiezan con algo que no es una letra, como un punto. Para ello, tecleamos el comando de la siguiente manera:

```
ls -al /home/fulanito/
```

y se mostrará en pantalla la lista de archivos y directorios que se encuentran en el directorio /home/fulanito/, pero únicamente los que cuelgan directamente de ese nodo. Si omites *[archivos]*, la lista que se dará será de los archivos y directorios inmediatamente colgando del directorio de trabajo.

Actividad 3.2 *Pide la lista breve de archivos que se encuentran en* /usr/local/

Actividad 3.3 *Pide la lista extensa de archivos que se encuentran en* /bin/. *¿Cuál es la información que aparece?*

Como puedes ver de la actividad anterior, muchas veces los archivos y directorios en un directorio son tantos, que si pides la lista extensa pudieses no alcanzar a verla completa. Para ello, haces lo que se llama *entubar* (en inglés, *pipe*), que consiste en redirigir la salida de un comando y usarla como entrada de un segundo comando. Quieres que el segundo comando sea uno que nos enseñe la información por páginas. Tenemos dos de estos comandos, que se usan precedidos por el símbolo de entubamiento `'|'`:

less
more

Si utilizas el comando less para entubar:

```
fulanito@[estnum fulanito] % ls -al /bin | less
```

la respuesta se muestra "por páginas", esto es, muestra el número de renglones que le cabe en la pantalla, y se espera en la parte inferior de la misma a que teclees un espacio o que oprimas la tecla AvPág, para seguir con la siguiente página. Si tecleas q, el listado se suspende.

Veamos el resultado que se te muestra con el comando anterior en la figura 3.2 .

Si el comando que se utiliza es less, entonces se podrá regresar sobre el listado, oprimiendo RePág; y avanzar tecleando espacio o AvPág.

También puedes "entubar" a que vaya a la impresora, si es que hay alguna conectada accesible, con el comando lpr.

Actividad 3.4 *Utiliza el comando* less *para entubar el listado del directorio* /usr/bin *y dí cómo responde.*

Figura 3.2 Resultado de entubar ls -al con less

```
total 6364
drwxr-xr-x    2 root    root       4096 Apr 19  2000 .
drwxr-xr-x   22 root    root       4096 Jun  6 05:08 ..
-rwxr-xr-x    1 root    root       2612 Mar  7  2000 arch
-rwxr-xr-x    1 root    root      60592 Feb  3  2000 ash
-rwxr-xr-x    1 root    root     263064 Feb  3  2000 ash.static
-rwxr-xr-x    1 root    root       9968 Feb  3  2000 aumix-minimal
lrwxrwxrwx    1 root    root          4 Feb 17  2000 awk -> gawk
-rwxr-xr-x    1 root    root       5756 Mar  7  2000 basename
-rwxr-xr-x    1 root    root     316848 Feb 27  2000 bash
-rwxr-xr-x    1 root    root     446800 Feb  2  2000 bash2
lrwxrwxrwx    1 root    root          3 Feb 17  2000 bsh -> ash
-rwxr-xr-x    1 root    root       9528 Feb  7  2000 cat
```

Continúa en la siguiente página

Figura 3.2 Resultado de entubar ls -al con more

Continúa de la página anterior

```
-rwxr-xr-x    1 root     root        12044 Mar  7  2000 chgrp
-rwxr-xr-x    1 root     root        13436 Mar  7  2000 chmod
-rwxr-xr-x    1 root     root        11952 Mar  7  2000 chown
-rwxr-xr-x    1 root     root        49680 Mar  6  2000 consolechars
-rwxr-xr-x    1 root     root        33392 Mar  7  2000 cp
-rwxr-xr-x    1 root     root        45712 Feb  9  2000 cpio
lines 1-18
```

3.2.1. Permisos de archivos

Los *permisos de acceso* a un archivo, o simplemente *permisos*, especifican quién puede hacer qué a un archivo. Los permisos de un archivo son asignados por su dueño y se pueden ver con el comando ls -l y modificarlos con el comando chmod.

Dado que los permisos de un archivo son un atributo del archivo mismo, todas las ligas a ese archivo tienen los mismos permisos.

Permisos para operaciones básicas

Las tres operaciones básicas que puedes llevar a cabo en un archivo son *lectura* (*read*), *escritura* (*write*) y *ejecución* (*execute*). Los permisos requeridos para realizar estas operaciones están denotados por 'r', 'w' y 'x' respectivamente. El permiso x es necesario para programas compilados y cualquier guión de shell que intentes usar como comando. Por ejemplo, si el archivo qq contiene un guión de shell pero no tiene el permiso x, no puedes ejecutar el comando qq por sí mismo. Si lo intentas, obtendrás un mensaje de error como:

```
command not found: qq
```

Sin embargo, puedes ejecutar qq con el comando

```
sh qq
```

aun cuando qq no tenga permiso de ejecución. En este ejemplo también necesitas que el archivo qq tenga permisos de lectura para poder ejecutar el script ya sea directa o indirectamente.

Los permisos r, w y x pueden especificarse independientemente para el dueño del archivo, para aquellos que se encuentran en el mismo grupo que el dueño y para todos los demás. Estas clases están representadas simbólicamente por u (*user*), g (*group*) y o (*others*).

Los permisos que asignas a un archivo pueden protegerlo no sólo de acceso no deseado sino también de tus propios errores. Por ejemplo, puedes proteger un archivo contra modificaciones no intencionales o de borrarlo accidentalmente suprimiendo su permiso w.

Permisos para directorios

Los permisos de archivos también se aplican a los directorios.

- El permiso r para un directorio te permite ver qué hay dentro, pero es insuficiente para acceder a los archivos que se encuentran dentro.
- El permiso w para un directorio se requiere para poder añadir o borrar archivos de él. Sin embargo, w no es necesario para modificar un archivo listado en el directorio o borrar su contenido.
- El permiso x para un directorio te permite operar con los nombres en ese directorio si los conoces, pero no te permite ver cuáles son si no los conoces. Normalmente siempre que r se concede, también es así con el permiso x.

Otros permisos de archivos

Además de los permisos rwx, Unix tiene otras clases de permisos usados principalmente (pero no exclusivamente) por el administrador del sistema. Los describimos a continuación.

El set-uid bit. El bit set-uid le permite a un programa correr con los permisos de su dueño en lugar de con los permisos del usuario que lo llama. Puedes especificar el bit set-uid con s cuando usas el comando chmod. En un listado, se indica reemplazando la x con una s.

Normalmente, cuando llamas un programa y ese programa accede a un archivo, los privilegios de acceso del programa son los mismos que tú tienes. Como ejemplo, supón lo siguiente:

- El usuario patito es dueño del ejecutable qq.
- El programa qq utiliza el archivo privado de patito llamado tarea.
- El usuario mamá-pato ejecuta el programa qq.

Los permisos que se aplican a tarea durante la ejecución son aquéllos de "otros" (ya que mamá-pato no es dueña de tarea); no se utilizan los permisos del usuario patito. Por lo tanto, si patito no le dio permisos a su archivo tarea para que "otros" lo pudieran leer (todo el mundo), el programa qq no puede acceder a él. Una posible solución a esto es que patito le dé permisos de acceder al archivo tarea a todo el mundo, pero entonces mamá-pato podría hacer cualquier cosa con el archivo (incluso, cosas que patito no tenía intensión de permitir.)

El bit set-uid es una solución a esto. Cuando un proceso ejecuta un programa, invoca la función exec del sistema, especificándole a ésta la ruta del programa a ser llamado. Si los permisos del programa incluyen el bit set-uid, entonces el *ID* efectivo del proceso se convierte en aquél del dueño del programa, así que el programa se ejecuta con los privilegios del dueño del programa.

En efecto, el bit set-uid habilita a un usuario para acceder a un archivo como tarea indirectamente – con mamá-pato como agente – pero no directamente.

El ejemplo clásico del uso del bit set-uid es el comando passwd. El dueño del archivo /etc/passwd es root (el superusuario) y debe ser escrito solamente por el programa passwd. Los usuarios ordinarios deben ser capaces de poder ejecutar el comando passwd, pero no de modificar directamente el archivo passwd. Por lo tanto, el bit set-uid está encendido[2] para el programa passwd, lo que le permite escribir en el archivo /etc/passwd, mientras que esa habilidad permanece negada para usuarios ordinarios.

El bit set-gid. El bit set-gid es como el set-uid excepto que se aplica a los permisos de grupo en lugar de a los permisos de dueño. El bit set-gid se especifica con una s en la posición de la x en los permisos del grupo.

El locking bit. Si el locking bit de un archivo está prendido, un programa que esté leyendo o escribiendo el archivo puede bloquear otros intentos de leer o escribir en él al mismo tiempo. Prender este bit previene que accesos simultáneos a un archivo puedan corromper su significado y dejarlo en un estado inconsistente. El *locking bit* está representado por una l reemplazando a la x en los permisos del grupo.

El sticky bit. Cuando se aplica a un directorio, el sticky bit previene que los archivos dentro de él sean borrados o renombrados por otra persona que no sea su dueño. Cuando se aplica a un archivo ejecutable, es la herramienta adecuada para retener al programa en memoria cuando dicho programa puede ser compartido por varios usuarios. Sólo el superusuario o el dueño del archivo puede activar el *sticky bit*. Este bit está indicado por una t en el lugar de la x en los permisos de otros.

Construcción de permisos

Los comandos chmod, umask y find hacen uso de la habilidad para construir conjuntos de permisos simbólicamente. Esto lo puedes lograr especificando un modo *simbólico* que les indique cómo modificar un conjunto existente, posiblemente vacío, de permisos.

Un modo simbólico consiste de una o más cláusulas separadas por comas. Cada cláusula, en turno, consiste de cero o más letras "quién" seguidas por una secuencia de una o más acciones a aplicar a la categoría designada por las letras "quién". Cada acción consiste de un operador seguido ya sea de una o más letras de permisos, de una letra "quién" o de nada en absoluto. Estas son las letras "quién":

u Permiso para el usuario del archivo.

g Permiso para el grupo del archivo.

[2] En este contexto, utilizaremos indistintamente prendido, encendido, arriba o puesto, para indicar que ese bit ha sido activado por medio de alguna utilería válida como el comando chmod.

o Permiso para otros (i.e. el resto del mundo.)

a Permiso para cualquiera (equivalente a ugo.)

Si omites las letras "quién" que preceden al operador, se supone a y la máscara de creación (de la cual hablaremos más adelante). Éstos son los operadores:

+ Añade estos permisos.

- Quita estos permisos.

= Prende exactamente estos permisos, quitando cualquier otro para las letras "quién" indicadas.

Éstas son las letras de los permisos:

r *Lectura*

w *Escritura*

x *Ejecución*

X *Ejecución* sólo si el archivo es un directorio o algún permiso x ya está puesto

s *Establece el ID de usuario o grupo*

t *Bit pegajoso*

l *Bloqueo durante el acceso*

Si especificas una letra "quién" después de un operador, entonces se copian los permisos asociados con la letra indicada.

Representación octal de los permisos

Los permisos para un archivo pueden ser especificados como un número octal. Esta forma es obsoleta y no recomendada, pero mucha documentación antigua de Unix utiliza la forma octal muy seguido; también algunos sistemas (no nuestro caso, en la Licenciatura en Ciencias de la Computación) no soportan aún la forma simbólica en todos los contextos. El número octal se obtiene sumando lógicamente los números de la siguiente lista:

4000 Establece el ID del usuario en ejecución.

20d0 Establce el ID del grupo en ejecución si *d* es 7,5,3 o 1 (permiso de ejecución otrogado); habilitar el bloque de otra forma.

1000 Habilita el bit pegajoso.

0400 Establece el permiso de lectura para el dueño.

0200 Establece el permiso de escritura para el dueño.

0100 Establece el permiso de ejecución para el dueño.

0040,0020,0010 Establece permisos de lectura, escritura o ejecución para el grupo.

0004,0002,0001 Establece permisos de lectura, escritura o ejecución para el resto de los usuarios.

Permisos para archivos recién creados

Cuando un programa crea un archivo, especifica un conjunto de permisos para ese archivo. Conjuntos típicos son u=rw g=rw o=rw (rw-rw-rw-, o 666 en octal) para archivos de datos y u=rwx g=rwx o=rwx (rwxrwxrwx, o 777 en octal) para archivos ejecutables. Los permisos iniciales son pocos debido a la *máscara de creación de archivos*, también conocida como la *umask* ("user mask").Tú puedes cambiarla con el comando umask. En otras palabras, los bits en la máscara de creación de archivos representan permisos que serán negados a los archivos recién creados.

Los permisos de un archivo recién creado se calculan restando lógicamente los bits en la máscara de creación de archivos de los permisos especificados por el programa que lo crea. Por lo tanto, si tu máscara vale algo como rwxrwxr-x (octal 002), que excluye el permiso de escritura de otros, aquéllos fuera de tu grupo no podrán escribir o borrar archivos que tú crees a menos que les cambies los permisos más tarde (con chmod, por ejemplo).

Un valor típico para umask es rwxr-xr-x (octal 022), que niega los permisos de escritura a todos menos a ti. Con este valor de máscara, un archivo creado con permisos especificados como u=rw g=rw o=rw es en realidad creado con permisos u=rw g=r o=r. La reducción de permisos por el valor de *umask* se aplica tanto a la creación de nuevos directorios como de nuevos archivos.

3.2.2. Archivos estándar y redireccionamiento

Muchos comandos de Unix leen su entrada de la *entrada estándar* a menos que especifiques archivos de entrada; y escriben su salida a la *salida estándar*. Por omisión, tanto la entrada como la salida están asociados a tu terminal (generalmente la salida es el monitor y la entrada es el teclado). Otro archivo estándar es el *error estándar*, que se usa para mensajes de error y otra información acerca del funcionamiento de un comando. La información enviada al error estándar también llega a la terminal.

Esta convención funciona bien porque puedes redireccionar tanto la salida como la entrada estándar. Cuando redireccionas la entrada estándar para ser tomada del archivo *archivo*, el programa que lee la entrada estándar leerá del archivo *archivo* en lugar de hacerlo desde la terminal. Algo similar sucede cuando redireccionas la salida estándar. Cuando se redirecciona la entrada se usa el operador $<$ precediendo a una trayectoria, mientras que para el redireccionamiento de la salida se usa el operador $>$.

Por ejemplo, el comando cat copia la entrada estándar a la salida estándar, por lo tanto el comando

cat $<$ felix $>$ gato

copia el archivo felix al archivo gato[3]. Si utilizas el operador $>>$ antes de una trayectoria, entonces la salida es agregada al final del archivo en lugar de escrita directamente. En el

ejemplo anterior, el contenido de gato es sobrescrito con el contenido de felix; pero si haces en su lugar

cat < felix >> gato

el contenido de felix es agregado al final del contenido de gato. Discutiremos más formas de redireccionamiento en breve.

El error estándar también se puede redireccionar pero la sintaxis para hacerlo es dependiente del tipo de shell que se utilice.

- Para el shell Bourne (sh) y la mayoría de los que pertenecen a su familia, como zsh, bash, korn, etc., la construcción '2> archivo' especifica que cualquier cosa que sea enviada al error estándar será redireccionada al archivo archivo.

- Para el shell C, se utiliza la construcción > & archivo, que manda tanto la salida estándar como el error estándar al archivo archivo. Este shell (y de su familia sólo tcsh) no permite enviar la salida estándar y el error estándar a distintos archivos. De hecho, el uso del shell C está en decadencia, pero aún tiene algunos seguidores.

Una propiedad importante y valiosa del redireccionamiento es que un programa cuya entrada o salida es redireccionada no tiene por qué saberlo. En otras palabras, un programa escrito bajo la suposición de que lee de la entrada estándar y escribe a la salida estándar también lee y escribe a archivos, siempre y cuando estos últimos sean pasados como redireccionamientos. La misma propiedad se aplica a guiones de shell.

Un uso común de cat es la creación de archivos nuevos, que generalmente contienen unas pocas líneas que se pueden teclear directamente de la pantalla. Por ejemplo,

```
fulanito@[estnum fulanito]% cat >felix
Esta es una prueba de cat
para ver exactamente como funciona
^D
```

crea al archivo felix en el directorio de trabajo, y el contenido del archivo se tomó de la entrada estándar, que se dio por terminada al teclear C-d (representado por ^D en el listado), que corresponde a un fin de archivo.
Otra manera de crear archivos es mediante el comando

touch mimosa

En realidad este comando lo que hace es actualizar la fecha de última modificación del archivo mimosa. Sin embargo, si el archivo no existe lo crea.

Nota: A lo largo de estas notas diremos que un programa produce un *resultado* cuando manda algo a la salida estándar. En la terminología tradicional de Unix se dice que *imprime* el resultado. Esta terminología viene de los días en los que la mayoría de las estaciones de trabajo estaban conectadas a un teletipo en lugar de un monitor, pero no debe confundirse con la acción de enviar un archivo a la impresora.

[3] Nota que como no das más que el nombre del archivo, la trayectoria se refiere al directorio de trabajo.

Actividad 3.5 *Crea el archivo* felix*, con el contenido que tú quieras, y luego ejecuta los ejemplos que dimos para este comando, en el orden dado.*

Actividad 3.6 *Ve con* ls -al gato *los parámetros y la fecha de última modificación del archivo* gato*. Ejecuta* touch *sobre el archivo* gato*. Obtén nuevamente un listado extendido y describe la diferencia entre ambos listados.*

Entubamientos y filtros

Puedes usar la salida de un comando como la entrada de otro comando, conectando ambos comandos con un tubo (*pipe*). La construcción resultante es un *pipeline* o *entubamiento*. Al igual que con las formas de redireccionamiento discutidas anteriormente, los programas conectados no tienen por qué saber de dichas conexiones.

Sintácticamente, creas un entubamiento escribiendo dos comandos con un *pipe* (el símbolo |) entre ellos. Por ejemplo, la línea de comandos

```
grep pest phones | sort
```

llama al programa grep, que extrae del archivo phones todas las líneas que contengan la cadena pest y produce esas líneas como su salida estándar. Esta salida se entuba a la entrada del programa sort, que ve dicha salida como su entrada. La salida de la línea de comandos completa es una lista ordenada de todas las líneas en phones que contienen pest.

En una línea que contenga tanto redireccionamientos como entubamientos, los redireccionamientos tienen mayor precedencia: primero los redireccionamientos se asocian con comandos y después los comandos con sus redireccionamientos se pasan por los entubamientos.

La creación de un entubamiento implica la creación de un par de procesos, uno para el comando que produce la información entubada y otro para el proceso que consume esta información. Estos procesos los crea el shell que interpreta la línea de comandos.

Un *filtro* es un programa que lee datos de la entrada estándar, los transforma de alguna manera y luego escribe esta información transformada a la salida estándar. Uno puede construir un entubamiento como una sucesión de filtros; estas sucesiones proveen una forma muy poderosa y flexible de usar comandos simples para que al combinarlos alcanzar grandes metas, posiblemente muy complejas. Generalmente este tipo de entubamientos se citan como ejemplos de la *filosofía Unix*.

Hay otras herramientas en los Unix estándar para simular estos entubamientos: archivos especiales *FIFO*, *sockets* y *streams*. Pero son mucho más específicos y sofisticados y definitivamente fuera del alcance del presente trabajo.

3.3. Otros comandos de Unix

Un *comando* es una instrucción que le dice a Unix que haga algo. La forma usual de pasar un comando a Unix es a través del shell, como respuesta a un *prompt* que éste nos proporciona.

El término *comando* se utiliza también para referirse a instrucciones que esperan los editores de texto como vi o emacs y otro tipo de programas.

3.3.1. Sintaxis estándar de comandos

POSIX[4] define una sintaxis estándar para comandos, pero no todos los comandos la siguen. Muchos comandos definidos por POSIX tienen una forma moderna y una sintaxis obsoleta, por compatibilidad con sistemas viejos. La razón de esto es que muchos programas importantes – y muy viejos – que aún están en uso, siguen sintaxis vieja de algunos comandos. Siempre que hablemos de la sintaxis de un comando, trataremos de apegarnos a la sintaxis moderna de POSIX, pero mencionaremos de vez en cuando la sintaxis vieja para algunos de ellos.

Aun en sistemas modernos hay una gran cantidad de comandos que no se apegan al estándar. Por ejemplo, el estándar especifica que las opciones de los comandos (no así, los operandos) pueden aparecer en cualquier orden, pero algunos comandos pueden requerir que las opciones aparezcan en un orden particular y muchas veces esto no está mencionado explícitamente en la página del manual del comando. En estos y otros casos hacer experimentos siempre es una buena idea.

Un comando consiste de una sucesión de palabras separadas por espacios en blanco (estos espacios en blanco pueden ser espacios sencillos, tabuladores e incluso – en algunos shell, solamente – nuevas líneas escapadas[5]). La primera palabra es el nombre del comando y las palabras subsecuentes son sus *argumentos* u *opciones*. El nombre del comando se refiere a un programa o guión de shell a ser ejecutado. Los operandos consisten de las opciones del programa, si hay alguna, seguidas de sus operandos, si hay alguno. Las opciones controlan o modifican lo que el comando hace. Los operandos especifican nombres de trayectorias o con lo que va a trabajar el comando.

Especificación de opciones. Las opciones se especifican con una sucesión de palabras. Cada palabra es un *grupo de opciones* o una *opción argumento*. Las opciones generalmente se denotan por letras. Se pueden escribir en cualquier orden y se pueden

[4] Portable Operating System Interfaz es un estándar definido por la IEEE – Institute of Electrical and Electronics Engineers – y ANSI – American National Standards Institute –. POSIX tiene como finalidad definir un sistema operativo que se *comporte como* Unix, sea o no Unix. Han existido varias versiones de POSIX, siendo la más importante POSIX.2, parte 2.

[5] Una nueva línea "escapada" es una que está precedida por una diagonal invertida (\).

combinar varias opciones en un solo grupo (siempre y cuando no reciban ningún argumento). Cada grupo de opciones está precedido por un guión (-). Por ejemplo, los comandos

```
ls -al
ls -l -a
```

son equivalentes e indican que el comando ls (*list archives*) sea llamado con las opciones a (*all files*) y l (*long listing*).

La última (o única opción) de un grupo de opciones puede ir seguida por una palabra que especifique uno o más argumentos opcionales para ella. Por ejemplo, el comando

```
sed -f qqscript qqarchivo
```

causa que el editor sed edite el archivo qqarchivo usando el guión de shell qqscript; en este caso, qqscript es un argumento para la opción -f y qqarchivo es un argumento para el comando sed mismo.

Cuando se reciben varios argumentos para una misma opción, generalmente se separan por comas, pero se pueden separar por blancos siempre y cuando se encierre la lista de argumentos entre comillas (dobles o sencillas), o si se pone una diagonal inversa frente a cada espacio en blanco (escapando los espacios en blanco). En cualquier caso, los argumentos deben formar una sola palabra. Los dos ejemplos que siguen muestran ambas convenciones:

```
prog −o zed1,zed2 −y "eres o no eres"
```

o alternativamente,

```
prog −o zed1,zed2 −y eres\␣o\␣no\␣eres
```

Otras convenciones para argumentos. Hay otras dos convenciones muy comunes para especificar argumentos:

'--' Indica el final de las opciones. Esto es muy útil cuando quieres pasar argumentos que empiezan con - a un comando. Por ejemplo,

```
rm - - -qq
```

es una forma de indicar que `−qq` es un archivo y no una opción, por lo que el comando borrará el archivo -qq. Si sólo se escribe rm -qq, obtendríamos un error sobre una opción q que no reconoce.

- Un solo guión representa a la entrada estándar en un contexto en que el programa espera una trayectoria. Por ejemplo, el comando

```
diff - qq
```

encuentra las diferencias entre la entrada estándar y el comando qq

Citas (*quotations*)

Todos los shells estándar asignan significados especiales a ciertos carácteres, llamados *meta-carácteres*. Para usar estos carácteres como carácteres ordinarios en la línea de comandos, debes citarlos (marcarlos de manera especial para que el shell **no** los interprete). Cada shell tiene su propio conjunto de meta-carácteres y sus propias convenciones de cita, pero algunas reglas generales siempre se cumplen:

- Una diagonal inversa (\) siempre sirve como carácter de escape para uno o más carácteres que le sucedan, lo cual le da a esos carácteres un significado especial. Si un carácter es un meta-carácter, el significado especial es – generalmente – el carácter mismo. Por ejemplo, \\ usualmente significa una sola \ y \$ el signo de pesos. En estos casos, la diagonal inversa le quita su significado a los carácteres, generalmente llamados carácteres escapados. La diagonal inversa misma es **el** *carácter de escape*.
- Cuando un texto se encierra entre comillas (" "), la mayoría de los meta-carácteres que estén en dicho texto son tratados como carácteres normales, excepto por $, que generalmente indica sustituciones a realizar.
- Otra forma de citar muy similar a la anterior, pero más fuerte es usar comillas sencillas ya que ni siquiera el carácter $ es interpretado.

Los espacios y tabuladores, si se encuentran dentro de algún texto citado, siempre son tratados como carácteres normales.

A continuación damos una breve lista de algunos comandos de Unix que consideramos muy importantes.

Tabla 3.1 Comandos de Unix importantes

Comando	Descripción
cat	El comando cat copia y concatena archivos. Tiene algunos usos comunes que no implican concatenación y que lo hacen uno de los comandos más útiles de Unix (en prácticas posteriores lo usaremos exhaustivamente). La forma general de cat es: cat [-estuv] [archivo...]
ln	El comando ln crea una liga a uno o más archivos existentes. Las nuevas ligas proveen nombres adicionales para esos archivos. La forma del comando ln es: ln [-f s] archivo ruta ln [-f s] archivo... dir

Continúa en la siguiente página

Tabla 3.1 Comandos de Unix importantes *Continúa de la página anterior*

Comando	Descripción
mv	El comando mv mueve uno o más archivos de un directorio, el directorio fuente (*source*), a otro directorio, el directorio destino (*target*). Si tanto el fuente como el destino están en el mismo sistema de archivos, mv simplemente cambia sus ligas y no mueve ningún dato de lugar. La forma del comando mv es: mv [-i f] archivo ruta mv [-i f] archivo... dir
cp	El comando cp copia uno o más archivos. A diferencia de ln y mv, cp sí copia los datos y no sólo cambia las ligas de los archivos involucrados. cp no es la única forma de copiar archivos en Unix, ya que cat provee un método alternativo. Más adelante les diremos cómo. pax con las opciones -rw es otra forma (pero no lo veremos aquí); cpio con la opción -p es otra (tampoco lo veremos) y tar también provee otra opción (éste sí lo veremos, pero con un propósito distinto al de copiar archivos). La forma de cp es: cp [-fip] iarchivo oarchivo cp [-fiprR] iname... odir
rm	El comando rm borra ligas a archivos, ya sean archivos ordinarios o directorios. Un archivo es eliminado cuando todas sus ligas han sido borradas, así que borrar la única liga a un archivo (el caso más común) borra el archivo mismo. La forma del comando rm es: rm [-firR] archivo...
rmdir	El comando rmdir borra ligas a directorios. La forma de la línea de comandos es: rmdir [-ps] dir... donde dir... es una lista de directorios.
more	El comando more te permite examinar uno o más archivos en tu terminal una página a la vez. También lo puedes usar para examinar la entrada estándar y por lo tanto examinar la salida de un programa haciendo un entubamiento hacia more (espero que después de este párrafo te des cuenta de la cantidad de terminología que ya debes más o menos manejar; si no has leído el material que se encuentra antes de esta sección, estás en problemas). La forma de la línea de comandos para more es:

Continúa en la siguiente página

Tabla 3.1 Comandos de Unix importantes *Continúa de la página anterior*

Comando	Descripción
	more [-ceisu] [-n number] [-t tag] [-p cmd] [archivo...]
	Nota que todos los parámetros son opcionales.
less	Hace lo mismo que more, pero extiende algunas de las capacidades interactivas de more.
most	Hace lo mismo que more, pero con una interfaz parecida a emacs. También extiende las capacidades de more utilizando comandos y enlaces de teclas "a la" Emacs.
lpr (lp)	Este comando ocasiona que un conjunto de archivos sea enviado a la cola de impresión. Recibe varias opciones, pero la forma usual de usarlo es:
	lpr archivo...
lpstat	Despliega el estado de las solicitudes de impresión.
lpc	Hace lo mismo que lpstat.
lpq	Muestra la cola de impresión.
lprm	Borra un archivo de la cola de impresión (debes ser dueño del archivo o el superusuario para poder eliminar un archivo de la cola de impresión).
find	El programa find busca en partes especificadas del sistema de archivos de Unix, archivos que casen con un cierto criterio. La forma del comando es:
	find ruta...[criterio]
	Aquí ruta... es una lista de trayectorias de archivos y directorios (usualmente sólo directorios). En el caso más simple, el criterio de búsqueda es una sucesión de patrones básicos, cada uno indicado con un guión. Cada patrón primario prueba si un archivo candidato cumple cierta propiedad; algunas de estas pruebas primarias siempre se satisfacen (son tautologías, :-)) y se ejecutan sólo por sus efectos secundarios. En general, un criterio es una combinación lógica arbitraria de patrones que se construye a partir de los patrones primarios.
	La prueba primaria -print es muy útil. Siempre es verdadera y, como efecto secundario, ocasiona que la ruta absoluta de los archivos encontrados (casados, apareados) sea enviada a la salida estándar.

Continúa en la siguiente página

Tabla 3.1 Comandos de Unix importantes *Continúa de la página anterior*

Comando	Descripción
which	El comando which te permite localizar un programa ejecutable; esto es, determina su trayectoria absoluta. La forma de la línea de comandos para which es: which progname...
file	El comando file (archivo) intenta clasificar un archivo examinando sus primeros bytes. La forma del comando es: archivo [-cL] [-f archivo] [-m archivo] archivo...
cmp	El comando cmp compara dos archivos. La forma del comando cmp es: cmp [-l | -s] archivo1 archivo2
diff	El comando diff analiza las diferencias entre dos archivos, tales como diferentes versiones del mismo documento. La forma del comando diff es: diff [-bcefhr] [-C n] archivo1 archivo2
chmod	El comando chmod cambia los permisos de un archivo. Para que este comando funcione, necesitas permiso de búsqueda en el directorio que contenga al archivo (lee lo relativo a permisos si algo de esto no te quedó claro). La forma del comando chmod (*change mode*) es: chmod [-Rf] perms archivo...
umask	El comando umask te permite cambiar o examinar la máscara de creación de archivos, también llamada *umask value*. El comando umask está incluido en el shell en lugar de ser un programa independiente (de hecho, en shells avanzados, como bash o zsh, la mayoría de los comandos de esta sección están incluidos en el shell). La sintaxis de la línea de comandos es: umask [-S] [perm]
chown chgrp	Los comandos chown y chgrp te permiten transferir la propiedad personal y grupal de un conjunto de archivos y directorios a alguien más. La forma general de estos comandos es: chown [-R] owner[.group] archivo... chgrp [-R] group archivo...

Continúa en la siguiente página

Tabla 3.1 Comandos de Unix importantes *Continúa de la página anterior*

Comando	Descripción
wc	El comando wc cuenta el número de carácteres, palabras y líneas en un archivo o conjunto de éstos. Una palabra es considerada como una secuencia (no vacía) de carácteres delimitada por espacio en blanco. La forma de la línea de comandos de wc es: wc [-clw] [archivo...]
touch	El comando touch (toca) un archivo, actualizando sus tiempos de acceso y modificación. Si el archivo no existe, se crea a menos que se especifique lo contrario. Tocar un archivo es una forma fácil de crear un archivo nuevo. La forma de la línea de comandos de touch es: touch [-acm] [-r ref-archivo | -t fecha] archivo...
tee	El comando tee provee una forma de capturar los contenidos de un entubamiento en un archivo, sin perturbar el flujo de información a través del entubamiento. Toma su nombre por la similitud con una conexión "T" como las de las tuberías de agua en tu casa. La forma del comando es:
tee	tee [-ai] [archivo...] tee copia la entrada estándar a la salida estándar y también a los archivos archivo.... Por ejemplo, si tecleas: cat hola.txt | tee adios.txt | more verás el contenido del archivo hola.txt por páginas (que es lo que hace more) y encontrarás en tu directorio actual un archivo llamado adios.txt que será una copia exacta de hola.txt.

3.4. Agrupando nombres de archivos

Usualmente, es necesario hacer referencia a un conjunto de archivos o directorios. Para este propósito se utilizan construcciones con comodines. Un comodín es una construcción que se puede remplazar por un conjunto o secuencia de carácteres.

El comodín '?' se usa para representar un carácter cualquiera. Por ejemplo, todos los archivos en el directorio /bin/ que empiezan con cualquier carácter al que le sigue nada más sh se pueden referir por medio de la expresión /bin/?sh.

Actividad 3.7 *Ejecuta* ls *dándole como parámetro la expresión* /bin/?sh. *¿Cuál es el resultado?.*

El carácter '∗' es otro comodín que se reemplaza por cualquier secuencia de carácteres, incluso la secuencia vacía, que es aquella que no tiene ningún carácter.

Actividad 3.8 *Ejecuta otra vez el comando* ls*, pero ahora con el parámetro* /bin/*sh. *¿Cuál es el resultado? ¿Cuáles son las diferencias en el listado de archivos con la construcción* /bin/?sh*?*

La tercera y última forma de expresiones con comodines son las que usan la expresión [. . .]. Los carácteres dentro de los corchetes proporcionan una lista para que se trate de casar con alguno (y sólo uno) de ellos.

Actividad 3.9 *Ejecuta otra vez el comando* ls*, pero ahora ejecútalo con el parámetro* /bin/[abc]sh. *¿Cuál es el resultado? ¿Cuáles son las diferencias en el listado de cuando se utilizó con la construcción /bin/?sh?*

Actividad 3.10 *Escribe la ruta absoluta para referirte a todos los archivos que inician con* m *y terminan con* .ps *en el directorio* /usr/share/texmf/doc/metapost/base/.

Actividad 3.11 *Escribe las rutas relativas para referirte a los mismos archivos anteriores, desde los directorios* /, /usr/share/, /facultad/calculo/ayudantia/ *y* /usr/local/bin.

Actividad 3.12 *Crea un directorio llamado* miPractica *que contenga dos subdirectorios llamados* usuarios *y* archivo *y que estén en tu directorio base.*

Actividad 3.13 *Crea un archivo llamado* raizArchivos.dat *en el subdirectorio* archivos *del directorio* miPractica *que contenga un listado de los archivos y directorios en el directorio raíz. Utiliza* ls *redireccionando la salida.*

Actividad 3.14 *A todos los directorios que haz creado asígnales permisos de lectura y escritura para ti y sólo de lectura a los demás.*

Actividad 3.15 *A todos los archivos que haz creado asígnales todos los permisos para ti y ningún permiso para los demás.*

Actividad 3.16 *Crea una copia del directorio* miPractica *y todo su contenido, llámale* ejercicios.

Actividad 3.17 *Elimina el directorio* miPractica *y todo su contenido.*

3.5. Konqueror: administrador de archivos y algo más

Konqueror se podría definir como un administrador de archivos, pero es una definición bastante pobre, ya que es capaz de realizar muchas más cosas. Se trata de una de las aplicaciones estrella de KDE. Con esta aplicación se ha intentado y conseguido tener una única aplicación que recoja todas las necesidades de un usuario.

Konqueror es:

- Un manejador de archivos: proporciona funciones de administración de archivos, que van desde las operaciones simples de cortar, copiar y pegar hasta búsqueda de archivos a través de una red local y remota. Permite previsualizar el contenido de los archivos y directorios, examinar sus propiedades e iniciar aplicaciones de una forma simple.
- Un navegador web que cumple las especificaciones de HTML como son:
  - Soporte para JavaScript.
  - CSS (hojas de estilo).
  - Escrituras bidireccionales (árabe y hebreo).
  - Soporte para ejecución de applets de Java.
  - Extensiones de Netscape para visualizar Flash, RealAudio y RealVideo.
  - Comunicaciones seguras mediante SSL.
  - Llenado automático de URL y formularios.
- Un cliente FTP totalmente equipado.
- Una aplicación de visualización universal, capaz de mostrar imágenes y documentos sin necesidad de iniciar otras aplicaciones. Esto lo hace integrando componentes de otras aplicaciones: de KView para visualización de imágenes; de KDVI para visualizar DVIs; de KGhostView para documentos PostScript; y aplicaciones de Koffice.
- Un emulador de terminal, que funciona de la misma manera que una terminal normal.

3.1 Administrador de archivos

El navegadores de directorios e Internet dentro de gnome es Nautilus.

3.5.1. Iniciando Konqueror

Hay varias formas de iniciar la aplicación de konqueror:

- Para iniciar konqueror como un administrador de archivos se puede seleccionar el icono con una casita en el panel o en el escritorio.

- Para iniciar konqueror como un navegador se puede seleccionar el icono en forma de globo terráqueo en el panel.

- Desde el menú, seleccionando Internet->Navegador web Konqueror para iniciarlo como navegador o Inicio para iniciarlo en modo de administrador de archivos.

- Simplemente tecleando konqueror para iniciarlo como administrador de archivos.

3.5.2. Konqueror como administrador de archivos

Konqueror se ve como un administrador de archivos común:

- Lista los archivos y directorios contenidos dentro de una carpeta dada y proporciona información sobre ellos. La "ruta" de la carpeta actual se muestra en la barra de dirección.
  Además muestra los archivos y directorios utilizando:

  - Vista de iconos.
  - Vista multicolumna.
  - Vista de árbol.
  - Vista de lista informativa.
  - Vista de lista detallada.
  - Vista de texto.
  - Vista de tamaño de archivo.
  - Álbum de fotos.

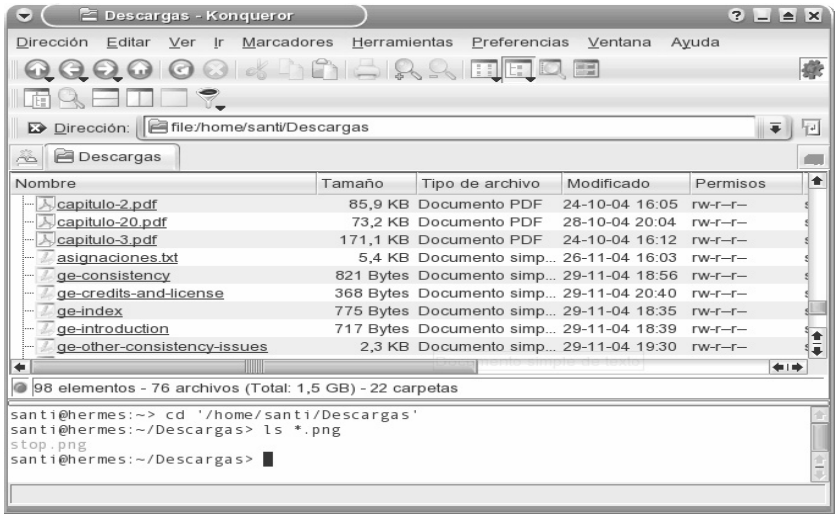
  La figura 3.3 muestra la funcionalidad estándar de konqueror como un administrador de archivos.

Figura 3.3 Konqueror como un administrador de archivos

- Permite ver dispositivos de montaje como CD-ROM, floppy o usb.
- Permite copiar, mover o borrar uno o varios archivos y directorios utilizando directivas de copiar, cortar y pegar o con "*drag and drop*".
- Permite crear nuevos archivos y directorios; ver y cambiar sus nombres, propiedades y atributos.
- Dispone de otra característica muy práctica, la opción Herramientas->Abrir terminal de la barra de menú, que abre una terminal dentro de Konqueror donde puedes introducir instrucciones básicas.

En la figura 3.4 se muestra konqueror con un emulador de terminal.

Figura 3.4 Konqueror con un emulador de terminal



3.5.3. Konqueror como un navegador web

Puedes utilizar Konqueror para navegar por la red muy fácilmente. Simplemente introduce una URL en la barra de dirección, presiona Enter, y ¡ya estás fuera!

El navegador de Konqueror permite:

- Navegar dentro de Konqueror igual que se haría en cualquier otro navegador. Incluso permite cargar varias páginas web en la misma ventana.
- Crear y utilizar accesos rápidos a sitios web, enviando directamente peticiones a un buscador sin tener que visitar primero el sitio.
  Por ejemplo, si tecleas gg:konqueror en la barra de direcciones y presionas la tecla **Enter** le pedirás a Google que busque todo lo relacionado con Konqueror.
- Guardar e imprimir páginas web, ya sean imágenes, paginas HTML, macros, enlaces y otros tipos más.
- Transferir archivos y directorios a través de la red utilizando el protocolo de FTP. Para ver un poco más de este protocolo puedes revisar la sección 5.8.
- Contar con soporte para extensiones de Netscape.

La figura 3.5 muestra Konqueror como un navegador de Internet.

Figura 3.5 Konqueror como un navegador de Internet

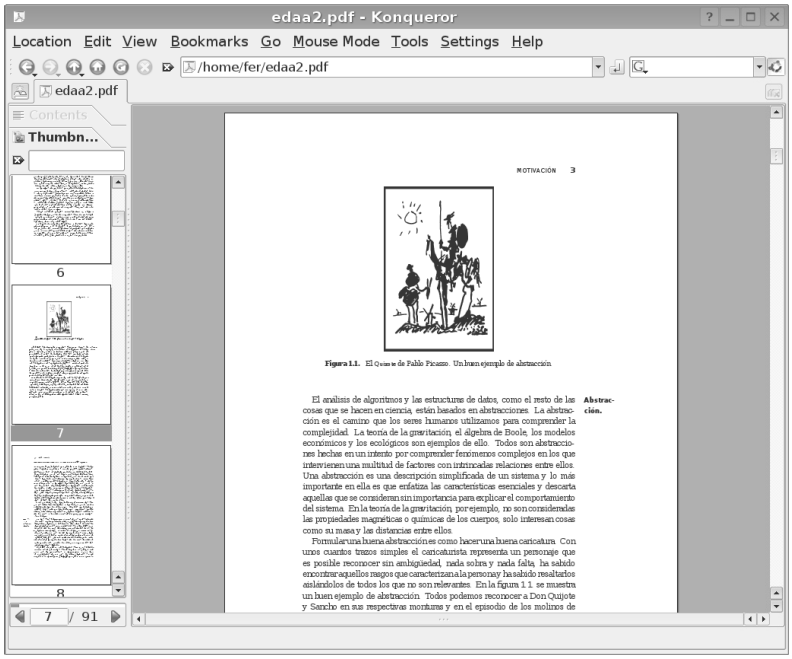
3.5.4. Konqueror como una aplicación integrada

Konqueror puede servir como un visor de muchos tipos de archivos. Es decir, si seleccionas una imagen la desplegará en la misma vista; Si seleccionas un archivo de texto, entonces te mostrará su contenido y lo mismo ocurrirá no importando si el tipo de archivo es postscript, dvi, KOffice o casi cualquier otro tipo de archivo. La figura 3.6 muestra una imagen dentro de Konqueror.

Figura 3.6 Konqueror deplegando una imagen

En la figura 3.7 aparece un documento postscript dentro de konqueror sin necesidad de instalar la aplicación.

Figura 3.7 Konqueror deplegando un documento postscrip.



Es interesante ver como estos visores no son parte del navegador y no fueron diseñados para eso; sin embargo, es posible integrar componentes de otras aplicaciones. El visor de imágenes es parte de KView; el visor de texto es parte de KWrite; el visor de DVI es parte de KDVI; el visor de postscript es parte de KGhostview; etc.

3.6. Emacs

3.6.1. Ejecutando Emacs desde la línea de comandos

XEmacs, encarnación de Emacs: xemacs

La forma más sencilla de iniciar Emacs, es ejecutando el comando (sin opciones), que corresponde al nombre del editor: emacs. Sin embargo, Emacs reconoce una sintaxis más

compleja para su línea de comandos; las siguientes son algunas de las opciones más importantes:

Tabla 3.2 Emacs: opciones de inicio

Opción	Descripción
-batch	Edición en lote. Con esta opción el editor mandará mensajes a la salida estándar. Debes usar las opciones -l, -f y eval para especificar archivos a ejecutar y funciones a llamar. Durante el curso de este trabajo, no habrá necesidad de usar esto y puede ser considerada una opción avanzada.
-nw	Utiliza la terminal actual (TTY), sin llamar el código de tu manejador de ventanas.
-debug-init	Si ocurre algún error mientras Emacs está arrancando, iniciará el depurador. Esta opción es útil cuando agregas cosas a tu archivo de configuración de Emacs y no están funcionando bien.
-q	No carga ningún archivo de inicio.
-user <user>	Carga el archivo de inicio del usuario especificado en lugar del tuyo.
-u <user>	Lo mismo que -user.
-help	Imprime el mensaje de uso de Emacs y sale.
-version	Imprime información sobre la versión de Emacs y sale.
-V	Lo mismo que -version.
-funcall <function>	Ejecuta la función de Lisp nombrada sin pasarle ningún argumento.
-f <function>	Lo mismo que -funcall <function>.
-eval <form>	Evalúa la forma de Lisp pasada como parámetro a eval. Ten la precaución de escapar adecuadamente la expresión para que el intérprete de comandos no la ejecute antes de que llegue a Emacs.
-load <archivo>	Carga el archivo nombrado de código en Lisp en Emacs.
-l <archivo>	Igual que -load <archivo>.

Por ejemplo la siguiente línea ejecutará Emacs dentro de la terminal:

```
% emacs -nw
```

El comando save-buffers-kill-emacs (C-x C-c) termina la ejecución de Emacs.

3.6.2. Conceptos esenciales

Dado que Emacs es más, mucho más, que un editor de texto, para integrar tantas operaciones distintas en un único paquete y mostrarlas en una sola ventana, se crearon varios conceptos para referirse a estas distintas partes de Emacs.

Además, y ésta es la característica que distingue a Emacs como un sistema para incrementar la productividad del programador, todo comando que puede ejecutarse puede y usualmente está asociado a una combinación de teclas. Puedes utilizar cualquier tecla como parte de estas combinaciones de teclas y a menudo involucran las teclas control (ctrl en algunos teclados) y meta (meta ya no existe en teclados modernos y usualmente se sustituye por alt).

Buffers

Emacs, y en general cualquier editor, no modifica directamente el archivo, sino que toman una copia del mismo y la coloca en un *buffer*, el cual contiene toda la información del estado actual de la edición. El estado actual del archivo no cambia sino hasta que le indicas a Emacs que deseas guardar los cambios en el disco.

Al igual que los archivos, los buffers también tienen un nombre, el cual generalmente se corresponde con el del archivo que se está editando. Sin embargo, en otros casos el nombre de un buffer no corresponde con el de ningún archivo, sino simplemente con algún propósito específico; es decir, no todos los buffers están ligados a un archivo sino que sirven para interactuar con Emacs. En este sentido es importante que entiendas la diferencia entre un buffer y un archivo.

Modos mayores y menores

La versatilidad de Emacs se debe, en buena medida, a que tiene diferentes modos de operación, de acuerdo con el tipo de texto que se esté editando o proceso que se esté ejecutando. Esto quiere decir que Emacs se adapta a las necesidades de edición de cada tipo de texto. Por ejemplo, para escribir una carta está el modo de texto; para editar un programa Emacs cuenta con modos para una gran cantidad de lenguajes de programación. Esto le permite a Emacs ser el tipo de editor que tú deseas.

Estos modos de operación se conocen simplemente como *modos*, y vienen en dos sabores: existen los modos mayores, que son los que proveen características que cambian radicalmente la manera en que se edita el texto; por ello un buffer siempre está asociado exactamente con un modo mayor.

Por otro lado existen los modos menores que añaden comportamientos específicos a un modo mayor. Por ejemplo, si deseas que se produzca un cambio de línea cuando rebases una cierta cantidad de carácteres en una línea, puedes usar un modo menor llamado Auto-

fill-mode. A diferencia de los modos mayores, un mismo buffer puede utilizar cualquier cantidad de modos menores.

La línea que aparece en la parte inferior de todo buffer se conoce como la línea de modo. Esto es porque parte de la información que nos proporciona es precisamente el modo o modos en el que se está editando el buffer (que aparecen entre paréntesis). Lo primero que aparece entre estos paréntesis es el modo mayor, seguido de todos los modos menores que el buffer en cuestión tenga activados.

Aparecen otros datos en la línea de modo, como el nombre del buffer, pero no son relevantes por el momento.

En el siguiente capítulo, en la sección 4.3, explicaremos el resto de los conceptos importantes de Emacs y los comandos (y su combinación de teclas asociadas) de edición. En lo que resta de este capítulo te mostraremos cómo Emacs puede ejecutar comandos de Linux e incluso un intérprete de comandos completo.

3.6.3. Dired

C-x C-f *(find-file) permite abrir archivos*

Es un modo utilitario para manejo de directorios. Este modo se activa cuando el archivo que se intenta abrir es un directorio. Por ejemplo, si ejecutas C-x C-f y luego completas o tecleas el nombre de un directorio, digamos ∼/pruebas, Emacs abrirá dicho directorio en el modo dired.

En este modo puedess observar los archivos en un directorio, cambiar sus permisos y propietarios, copiar, borrar o mover archivos y directorios. Algunos de los comandos que puedes usar en este modo son:

Tabla 3.3 Emacs: comandos del modo dired

Combinación	Descripción
v	Abre el archivo únicamente en modo de lectura.
o	Abre el archivo, pero en otra ventana.
C-o	Abre el archivo en otra ventana pero te deja en la que actualmente estás.
m	Marca los archivo o directorios (para copiar o mover, por ejemplo).
C	Copia los archivos marcados.
R	Mueve los archivos marcados.
d	Marca archivos para eliminación.
x	Elimina archivos marcados.

Continúa en la siguiente página

Tabla 3.3 Emacs: comandos del modo dired *Continúa de la página anterior*

Combinación	Descripción
+	Crea un directorio.
Enter	Abre el archivo o directorio en el que estás parado.

3.6.4. Ejecución de comandos de Unix desde Emacs

Emacs ofrece varios mecanismos para ejecutar comandos del sistema y procesar los resultados. El más sencillo de estos mecanismos lo ofrece el comando shell-command o M-!.

Cuando se ejecuta este comando, Emacs solicita se escriba el comando en el *minibuffer*[6], que se encuentra en la parte más baja de la ventana de Emacs, justo debajo de la línea de modo.

Por ejemplo, puedes ejecutar un comando para copiar el archivo /etc/passwd al directorio /tmp, siguiendo esta secuencia:

i. Primero ejecutas el comando M-!, tecleando esta combinación.

ii. A continuación, en el minibuffer, que dice *Shell command:*, escribes el comando de Unix: cp /etc/passwd /tmp

Emacs mostrará a continuación en la misma línea, pero que ahora se llama área de eco, *(Shell command succeeded with no output)*, indicando que todo salió bien y la ejecución fue exitosa.

Otro mecanismo muy utilizado es ejecutar un intérprete de comandos completo dentro de un buffer de Emacs, lo que se logra con el comando shell y puedes ejecutarlo con la secuencia: M-x shell. Emacs te llevará inmediatamente a un nuevo buffer que simula una terminal de Linux. Como es obvio, en este intérprete de comandos (porque es un intérprete de comandos dentro de Emacs) puedes ejecutar todos los comandos de Unix.

Actividad 3.18 *Dentro de Emacs ejecuta un shell, con* M-x shell *y repite todas las actividades de este capítulo.*

[6]Emacs también utiliza esta área para informar al usuario y en este caso se conoce como *área de eco* y no como minibuffer. Por ejemplo, cuando escribes un comando como C-x C-f, si haces una pausa prolongada después de escribir C-x, Emacs presenta esa parte en el área de eco como ayuda. Si escribes la secuencia del comando completa muy rápidamente, Emacs no muestra nada en el área de eco para no distraerte.

Edición 4

4.1. La guerra de los editores de texto

En la *cultura geek*[1] hay un gran debate entre la comunidad de programadores sobre cuál editor de texto de propósito general es el mejor. Aunque es una guerra que siempre tiene nuevos contendientes, los grandes campeones hasta la fecha, sobre todo en el mundo Unix, son los seguidores de Emacs y los de Vi.

La comunidad de *hackers*[2] trata con reverencia a sus piezas de software favoritas, muchas veces rayando en el fanatismo, y los editores de texto son probablemente las piezas de software más difundidas y apreciadas. Es cierto, hay otras guerras y muy violentas, como las de los sistemas operativos o los lenguajes de programación, pero la de los editores Emacs y Vi es digna de leyendas e historias de encuentros salvajes y cruentos entre sus seguidores.

Nosotros, los autores de este compendio, nos contamos y a mucha honra entre los seguidores de Emacs. Vi es malo, de hecho, para los iniciados en el arte de la edición (lo cuál serás tú también al final de este capítulo); es claro que Vi es el editor del infierno y nos

[1] Geek es el sobrenombre que reciben aquellas personas fascinadas, probablemente en forma obsesiva, con algunos temas específicos u obscuros de algún área del conocimiento o de la imaginación. En años recientes, geek se utiliza generalmente para los obsesionados con tecnología o computación. Es probable que si estás interesado en los temas del presente libro, tú mismo(a) califiques como geek, ¡bienvenido(a) al club!

[2] *hacker* es aquella persona que crea o modifica software o hardware de computadora, incluyendo tareas de programación, administración de sistemas y seguridad. Hack, por otro lado, tiene una connotación negativa y es algo así como un *chicano fix*, una compostura poco elegante para un programa o sistema.

referiremos a él por su nombre completo: vi vi vi (666, en romano), el editor de la bestia. Hablaremos de Emacs más adelante, en la sección 4.3. Linux, y de manera más general el software libre, te ofrece una única cosa y hay que valorarla: la facultad de elegir.

4.1.1. Vieja guardia contra nueva guardia

Hoy día contamos en Linux con más editores de los que podríamos dominar o explicar en una vida, menos aún en este capítulo. Sin embargo, en KDE existe una nueva camada de editores que han llamado nuestra atención porque hacen lo que un buen editor debe hacer, pero además se integran al resto del ambiente, lo cual, a la larga, ofrece otra gama de posibilidades.

En este capítulo revisamos los editores Kate en la siguiente sección y Emacs en la sección 4.3. En capítulos subsecuentes revisaremos otros editores especializados, como Kile en el capítulo 6. Es importante que tomes en cuenta, sin embargo, que durante tu vida académica y profesional te encontrarás en situaciones donde no contarás con un ambiente gráfico para trabajar, como KDE, y es aquí donde Emacs muestra su superioridad, ya que también puede ejecutarse en una terminal o consola de texto.

4.2. Kate: un editor para programadores

4.2.1. Introducción

```
$ smart
install kate
```

Kate es el editor de texto preferido por programadores en KDE. Algunas características de Kate que revisaremos en este libro incluyen resaltado configurable de sintaxis para una gran variedad de lenguajes, desde C o C++ hasta XML o guiones de shell; la habilidad para crear y mantener proyectos; una interfaz para varios documentos (MDI por sus siglas en inglés) y un emulador de terminal integrado.

4.1 Editores de texto

El editor nativo en Gnome es gedit, un editor de texto eficiente y sencillo. Este editor, sin embargo, no es directamente comparable con Kate, que revisamos en esta sección, y tampoco con Emacs, que veremos más adelante.

Kate es ideal para programadores. Sin embargo, una de sus fortalezas más importantes no está relacionada con su manejo de archivos fuente de diversos lenguajes de programación, sino con la facilidad para abrir varios documentos simultáneamente. Más aún, para

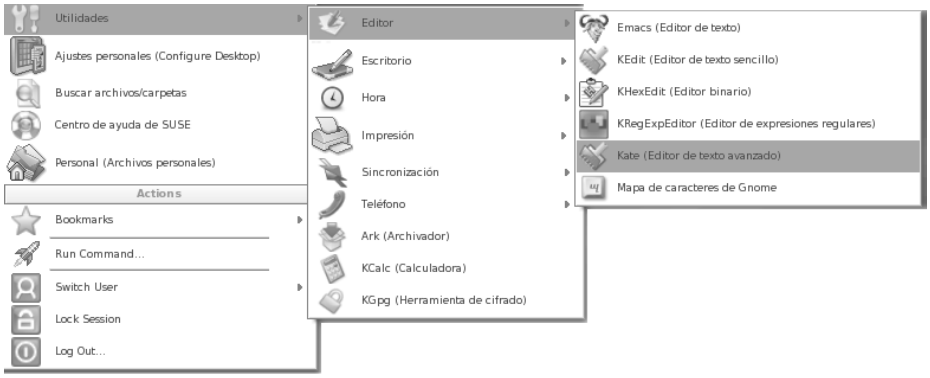
programar en lenguajes modernos como Java o C# contamos en Unix con editores más poderosos y revisaremos algunos en el capítulo 8.

4.2.2. Empezando a utilizar Kate

En esta sección te mostraremos las operaciones básicas de Kate para que puedas iniciar la edición de documentos. Conforme avances veremos algunas de las opciones avanzadas y las posibilidades de configuración de Kate.

Puedes iniciar Kate desde el menú de KDE, como se muestra en la figura 4.1, o en una terminal con el comando kate. Utilizando el menú de KDE localiza la sección de editores, que mostrará todos los editores disponibles en tu sistema y selecciona kate.

Figura 4.1 Kate en el menú de KDE



Por omisión Kate te mostrará el último archivo que editaste en tu sesión anterior. Si este comportamiento no te gusta, lo puedes cambiar en el menú de preferencias.

Puedes utilizar, como ya mencionamos, el comando kate para iniciar el editor; si le proporcionas el nombre de un archivo como parámetro, abrirá el archivo o lo creará si no existe.

```
% kate miarchivo.txt
```

Si tienes una conexión activa a la red y los permisos adecuados, incluso puedes abrir un archivo en un servidor remoto con algún protocolo soportado por KDE:

```
% kate ftp://ftp.fciencias.unam.mx/pub/moderno/prueba
```

En el capítulo 5 se describen varios protocolos de red; KDE ofrece soporte transparente[3] para varios de ellos.

Sesiones

Kate soporta sesiones, que son una forma de mantener distintas listas de archivos y configuraciones. Puedes tener tantas sesiones como desees y éstas pueden ser anónimas o tener un nombre.

Por omisión, cuando inicias Kate se inicia en la sesión predeterminada, pero cuando lo inicias con una sesión, el nombre de ésta aparece en el título de la ventana. Puedes iniciar Kate desde la línea de comandos e indicarle que abra una sesión:

```
% kate -s nombre lista-de-archivos
```

Si la sesión llamada *nombre* no existe, Kate la creará. En caso de existir una sesión abierta con ese nombre, los archivos especificados se anexarán a esa sesión.

También se pueden crear y manejar sesiones desde un ejemplar de Kate en ejecución a través del menú, eligiendo Sesiones.

La lista de documentos

La lista de archivos muestra todos los documentos actualmente abiertos en Kate. Los archivos que han sido modificados muestran un icono de disco magnético a la izquierda para indicar su estado.

Si hay dos o maś archivos abiertos con el mismo nombre (que viven en distintos directorios), Kate antepone $< 2 >$ al nombre del segundo, $< 3 >$ al nombre del tercero y así sucesivamente. La ventana de información te mostrará la ruta completa del archivo para que selecciones el adecuado.

Para mostrar en la ventana activa el archivo deseado, basta con que hagas click sobre el nombre en la lista.

La lista de archivos puede ordenarse de distintas formas; para ello Kate ofrece un botón a la derecha que te permite ordenar con alguno de los siguientes criterios: por orden de apertura, por nombre (alfabético) y por URL.

Esta lista aparece por omisión, aunque puede moverse, al lado izquierdo de la ventana de edición.

El selector de archivos

El selector de archivos es un visor de carpetas, que te permite navegar tu espacio de archivos y abrir el archivo adecuado. Su funcionamiento es idéntico al de otras aplicaciones KDE, particularmente Konqueror.

[3] Cuando decimos que algún proceso es *transparente* nos referimos al hecho de que funciona como si no estuviera ahí, que no se le ve.

Emulador de terminal

Kate integra un emulador de terminal que es una copia de la aplicación Konsole de KDE. Está disponible de varias formas, a través del menú Ventana en vista de herramientas, a través del botón en la parte inferior llamado terminal o bien pulsando la tecla F7.

Esta terminal está aquí para facilitarte acceso a comandos de Linux y al sistema de archivos.

Herramientas externas

En el menú de Herramientas encontrarás uno de herramientas externas, que te permite invocar aplicaciones externas a Kate con datos relacionados con él o los documentos actuales, por ejemplo, su URL, directorio, texto.

Qué herramientas se incluyen es configurable por el usuario a través del panel de configuración de herramientas externas. Las herramientas que se incluyen por omisión están relacionadas con el sistema para manejo de control de versiones CVS, otra que te permite realizar una búsqueda en el web con Google (utiliza Konqueror) con el texto seleccionado en el documento y una más para ejecutar cualquier programa sobre el documento.

Arrastrar y soltar

Kate soporta el protocolo de KDE para arrastrar (*drag*) y soltar (*drop*), por lo que puedes arrastrar un archivo desde el escritorio o una ventana de konqueror y dejarlo caer o soltarlo encima de Kate y éste abrirá el archivo.

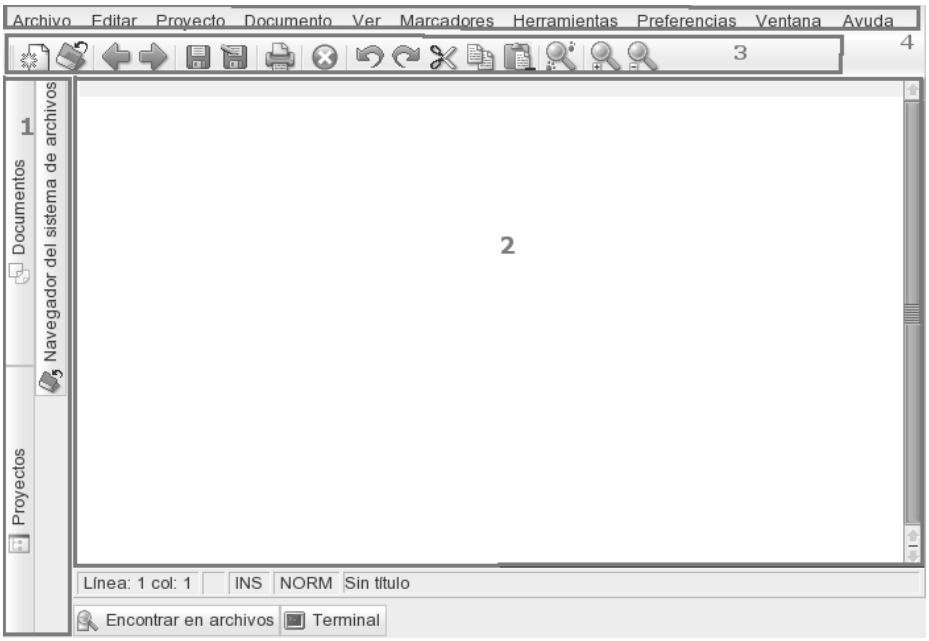
4.2.3. Editando archivos

En la figura 4.2 puedes apreciar la ventana inicial de Kate. Notarás que hemos resaltado algunas secciones con rojo y están numeradas. La sección 1 es similar en varias aplicaciones que funcionan en KDE y ofrece varios selectores: de documentos, proyectos y archivos. Cada selector ofrece distintas posibilidades para localizar un archivo o proyecto a editar.

La sección marcada con 2, que es la más grande de la ventana de Kate, es donde se despliega el contenido del archivo que editas. Dado que Kate tiene la facultad de editar más de un archivo a la vez, cuando existe más de un archivo abierto, en la parte superior de esta ventana se crean varias pestañas, una por cada archivo abierto.

Es posible dividir esta sección en dos, ya sea de forma horizontal o vertical. En ambos casos, las dos partes o marcos, contendrán el archivo que estés editando, pero sólo uno de ellos será el marco activo, lo que se representa por medio de un indicador verde.

Figura 4.2 Ventana de Kate cuando inicia



La sección 3 es la barra de herramientas de edición principal y la sección 4 contiene el menú general de Kate. Existen otros elementos en la ventana principal; por ejemplo, en la parte baja hay un botón marcado *Encontrar archivos* que te permite localizar archivos en tu sistema haciendo búsquedas sobre su contenido; también hay un botón más para abrir una terminal dentro de Kate.

Acceso rápido

Tecla Control *es C.*

Kate integra una serie de acciones que puedes realizar a través de enlaces de teclas y que te permiten un acceso rápido. A continuación se muestra una tabla con los enlaces por omisión, pero puedes seleccionar otros utilizando el menú *Preferencias* y la opción *Configurar accesos rápidos*.

Tabla 4.1 Kate: enlaces de tecla para acceso rápido

Enlace	Descripción
←	Mueve el cursor un carácter a la izquierda.
→	Mueve el cursor un carácter a la derecha.
↑	Mueve el cursor una línea arriba.
↓	Mueve el cursor una línea abajo.
Avpág	Mueve el cursor una página hacia arriba.
Repág	Mueve el cursor una página hacia abajo.
Ins	Pasa de modo inserción a modo sobreescritura y viceversa. Cuando está en modo inserción el editor añadirá cualquier carácter introducido al texto y empujará los carácteres previamente introducidos a la derecha del cursor de texto. El modo de sobreescritura hace que la entrada de cada carácter elimine el carácter inmediatamente a la derecha del cursor de texto.
BackSpace	Borra el carácter a la izquierda del cursor.
Inicio	Mueve el cursor al comienzo de la línea.
Fin	Mueve el cursor al final de la línea.
Supr	Borra el carácter a la derecha del cursor (o cualquier texto seleccionado).
Shift–←	Marca el texto un carácter a la izquierda.
Shift–→	Marca el texto un carácter a la derecha.
F1	Ayuda.
Shift–F1	Ayuda: ¿qué es esto?
C–f	Buscar.
F3	Repetir la búsqueda.
C–b	Insertar un marcador.
C–c	Copiar el texto marcado al portapapeles.
C–n	Nuevo documento.

Continúa en la siguiente página

Tabla 4.1 Kate: enlaces de tecla para acceso rápido *Continúa de la página anterior*

Enlace	Descripción
C–p	Imprimir.
C–q	Salir - cerrar la copia activa del editor.
C–r	Reemplazar.
C–s	Salvar archivo.
C–v	Pegar.
C–x	Borrar el texto marcado y copiarlo al portapapeles.
C–z	Deshacer.
C–Shift–z	Rehacer.

Trabajando con la selección de texto

Hay dos formas de seleccionar secciones de texto en Kate, una de ellas con el ratón y la otra con el teclado.

Para seleccionar con el ratón debes mantener presionado el botón izquierdo y arrastrar el puntero hasta cubrir la región deseada. El texto se selecciona mientras arrastras.

Para seleccionar con el teclado debes mantener presionada la tecla Shift y utilizar una combinación de las teclas de cursor, Inicio y Fin.

Una vez que has creado una selección, puedes por ejemplo copiarla con la opción Editar ⇒ Copiar o bien C-c y posteriormente puedes copiar el texto, ahora en el portapapeles, con C-v.

Para eliminar la selección actual, puedes utilizar alguna de las tres opciones: (a) elegir del menú Editar ⇒ Deseleccionar, (b) C-a o bien (c) hacer click con el botón izquierdo del ratón en cualquier parte del editor.

Buscando y reemplazando

Los diálogos de búsqueda y sustitución de Kate son muy similares, salvo porque el segundo incluye un espacio para indicar la cadena por la que habrá de sustituir la o las presencias del texto buscado.

Los diálogos contienen estas opciones:

Tabla 4.2 Kate: opciones para buscar y reemplazar

Enlace	Descripción
Texto a buscar	Aquí va la cadena de búsqueda.
Expresión regular	Si está activado, la cadena de búsqueda será interpretada como una expresión regular.
Mayúsculas y minúsculas	Si está activada, la búsqueda distinguirá entre mayúsculas y minúsculas.
Desde el cursor	Si está marcado, la búsqueda iniciará desde el punto actual del cursor.
Buscar hacia atrás	Si está activado, la búsqueda se hará desde el punto actual hacia el inicio del documento.
Reemplazar con	Aquí va la cadena de sustitución.
Preguntar al reemplazar	Si está activado, aparecerá un diálogo preguntando la acción que debe llevar a cabo Kate cada vez que encuentre la cadena de búsqueda. Las posibles respuestas son: sí, no, todo y cerrar.

Para iniciar una búsqueda puedes utilizar el menú Editar ⇒ Buscar o bien C-f; para reemplazar debes utilizar Editar ⇒ Reemplazar o C-r.

Marcadores

Los marcadores sirven para señalar líneas interesantes en el documento y volver a ellas fácilmente en el futuro. Puedes establecer y eliminar marcadores, moviendo el cursor a esa línea y activando la opción Marcadores ⇒ Cambiar o presionando C-b. Después, para recorrer los marcadores en el documento puedes utilizar en el menú Marcadores ⇒ Siguiente (C–Avpág) o ⇒ Anterior (C–Repág).

Ajustes de línea y sangrado

Kate tiene una opción para que le indiques que debe cortar las líneas que sean más largas que cierto tamaño. Esto lo puedes configurar en las preferencias, en la página de Editor.

Cuando está activada esta opción de ajuste de línea, al llegar al número máximo de carácteres permitidos, Kate insertará (en un espacio en blanco apropiado) un salto de línea.

En cuanto al sangrado del documento, Kate soporta una amplia gama de sangrados automáticos especializados en distintos tipos de entrada. Puedes utilizar uno de los modos disponibles a través del menú Herramientas ⇒ Sangrado. En la siguiente tabla se

muestran algunos de los sangrados disponibles en Kate:

Tabla 4.3 Kate: sangrados disponibles

Enlace	Descripción
Ninguno	Seleccionando este modo se desactiva completamente el sangrado automático.
Normal	Este sangrado simplemente mantiene un sangrado similar al de la línea anterior con cualquier contenido diferente a un espacio en blanco. Puedes combinar éste con las acciones sangrar y borrar sangrado para satisfacer tu gusto personal.
Estilo C	Es un sangrado para los lenguajes C y similares, como C++, C#, Java, Javascript. Sin embargo, no funciona con lenguajes de script como Perl o PHP.
SS C	Es un sangrado alternativo para los lenguajes C y similares, con las mismas restricciones.
Python	Un sangrado específico para el lenguaje de script python.
XML	Un sangrado automático para XML. Sin embargo, no intentes utilizarlo con HTML o XHTML, ya que falla con las etiquetas HTML de estilo antiguo (etiquetas abiertas como)

4.3. Emacs

Emacs es el editor de los Dioses, por directa contraposición con su odiado archienemigo vi (asegúrate de leer dos veces la introducción de este capítulo, en la sección 4.1).

En el capítulo anterior (sección 3.6.2) te mostramos algunos de los conceptos esenciales de Emacs y en esta sección concluimos la exposición de conceptos y te enseñaremos a utilizar una plétora de comandos para edición. No tienes que aprenderlos todos para utilizar Emacs, pero es importante los conozcas y practiques cada vez que tengas oportunidad.

4.3.1. Comandos

Meta –x *sirve para ejecutar comandos*

Los comandos en Emacs tienen un nombre auto-descriptivo. En muchas ocasiones estos comandos están formados por dos o más palabras separadas por un guión y, como puedes imaginar, ejecutar estos comandos llamándolos por su nombre puede resultar engorroso.

Por esta razón, Emacs crea un vínculo entre un comando y una secuencia de teclas y puedes ejecutarlo ya sea invocándolos por su nombre, por ejemplo Meta–x find-file, o bien tecleando la secuencia asociada, en este caso C-x C-f.

Las secuencias asociadas lucen complicadas al principio, pero después de utilizarlas un tiempo te darás cuenta que están ahí por muy buenas razones. De hecho, notarás que los comandos más usuales tienen asociadas secuencias de teclas cortas. Emacs fué diseñado por programadores, para programadores, por lo que la eficiencia lo es todo.

Cada vez que tienes que despegar una de tus manos de la parte central del teclado para buscar con el ratón un comando en los menús, pierdes tiempo valioso. Intenta: compara el tiempo que te toma teclear la secuencia C-x C-f contra el tiempo que te toma encontrar en el menú el mismo comando find-file. Ahora imagina una sesión de edición donde escribirás una decena de cuartillas y ejecutarás varios cientos de comandos.

Entre más utilices Emacs menos tendrás que pensar en las secuencias de los comandos, tus manos se harán cargo. Esto se conoce como memoria dactilar. Una vez que te sientas cómodo con los comandos de Emacs que dominas, regresa a este capítulo e investiga algunos de las secciones que a continuación presentaremos y aprende nuevas secuencias.

En la sección 3.6.2 te dijimos que las teclas Control y Meta (alt y alt gr) son usualmente utilizadas en secuencias de teclas. De hecho, ya hemos utilizado algunas de éstas a lo largo de este material, pero en nomenclatura de Emacs no mostramos el nombre de estas teclas completo, sino que las sustituimos así: Ctrl es C y Meta es M. Entonces una secuencia así: C–x, significa mantener presionada Ctrl y sin soltar presionar x.

Sucede lo mismo con M, aunque aquí tenemos una pequeña variante y es que la tecla de escape, esc, puede utilizarse en sustitución de Meta. Cuando se ejecuta una secuencia M–x utilizando esc, sin embargo, no se mantienen presionadas de manera simultánea. Es decir, primero presionamos esc, soltamos, y a continuación x.

El guión nos sirve para indicar que esas teclas se presionan juntas, los espacios en la secuencia nos indican que debemos soltar la(s) tecla(s) anterior(es) antes de iniciar con las teclas después del espacio. Por ejemplo, C–x C–f se teclea así: simultáneamente Ctrl y x, liberamos ambas y de manera simultánea presionamos Ctrl y f.

Escribir texto o ejecutar comandos

Puedes estarte preguntando si eventualmente Emacs sirve para editar texto o sólo para ejecutar comandos. Por raro que parezca, Emacs sirve exclusivamente para ejecutar comandos y el más ejecutado es self-insert-command, que es ejecutado por cada tecla y cuya única función es imprimir el carácter que representa la tecla en el buffer actual, o sea, la que acabas de oprimir.

Para redactar una carta, escribir tus tareas, programar o hacer una presentación, simplemente tienes que ejecutar Emacs, visitar un archivo (con nuestro ya viejo amigo: C-x C-f) y comenzar a teclear. Más adelante te decimos como salvar el contenido del buffer (sección 4.3.11); recuerda teclear C-x C-c para terminar la ejecución de Emacs.

4.3.2. Movimiento

Existe una posición privilegiada que es donde se efectúan las operaciones de Emacs, conocida como *el punto*. Generalmente, este punto está marcado con el principio de un cursor, que es un simpático cuadrado que nos indica dónde estás parado con respecto al buffer que estás editando. El punto en realidad se refiere a la posición entre el carácter cubierto por el cursor y el inmediato anterior.

Como es natural, las primeras operaciones que desearás hacer son las de movimiento del punto dentro del buffer, las cuáles se pueden efectuar a muchos niveles, desde carácter por carácter, línea por línea, palabra por palabra, por párrafo o por pantalla.

La mayoría de las operaciones de movimiento se pueden efectuar tanto hacia adelante como hacia atrás. A continuación presentaremos una tabla de movimientos tratando de explicar, cuando no sea claro, las funciones de estos comandos.

Tabla 4.4 Emacs: comandos de movimiento

Unidad	Adelante	Atrás
carácter	C-f	C-b
palabra	M-f	M-b
línea	C-p	C-n
enunciado	M-a	M-e
párrafo	M-{	M-}
pantalla	C-v	M-v

Es posible que necesites desplazarte de manera instantánea a ciertas posiciones privilegiadas del texto. Emacs ofrece los siguientes comandos para realizarlo:

Tabla 4.5 Emacs: movimiento a lugares especiales

Elemento	Inicio	Fin
línea	C-a	C-e
buffer	M–<	M–>

4.3.3. Matando y borrando

Hasta el momento sabes cómo agregar texto al buffer y cómo moverte en él, pero ¿qué opciones ofrece Emacs para borrar texto? Ésas te las mostramos aquí, pero antes te presentamos un comando muy útil undo o C-x u, que deshace lo que hizo el último comando. Si ejecutas repetidamente este comando puedes retroceder tanto como quieras en el proceso de edición del buffer actual.

Los comandos de Emacs para borrar se dividen en dos grandes categorías, los que *matan* y los que *borran*. Como una manifestación del optimismo de Emacs, la muerte no es para siempre, por lo que todo aquello que matas puede *reencarnar*. Para lograr esto, Emacs envía a todos los muertos a una estructura conocida como el anillo de la muerte (kill-ring).

Por el contrario, cuando borras algo es para siempre, a menos, claro, que utilices el comando undo. Para evitar que los usuarios de Emacs pasemos malos ratos por eliminar cosas sin quererlo, los únicos comandos para borrar que veremos funcionan sobre carácteres.

En la siguiente tabla te mostramos los comandos para borrar y matar.

Tabla 4.6 Emacs: comandos para matar y borrar

Unidad	Adelante	Atrás
carácter (borrado)	C-d	BackSpace
palabra	M–d	M–BackSpace
línea	C-k	C-u 0 C-k
enunciado	M–k	C-x Supr

4.3.4. Reencarnación de texto

Cuando matas alguna sección, los carácteres que la conforman entran al kill-ring. Si eres perspicaz te preguntarás: *¿de qué sirve que se vayan a este lugar si no sé como traerlas de regreso?* Ésta es una pregunta válida: en efecto existe una manera de traer de regreso aquellos carácteres que han muerto en este buffer (o en cualquier otro buffer); a esta ope-

ración se le conoce como yank y el comando para realizarla es C-y. La ejecución de este comando inserta el último texto muerto en el punto actual.

Como es de esperarse, puedes reencarnar texto que hayas matado con anterioridad. Esto se logra con el comando M–y, el cual, si el comando inmediato anterior fue un yank, reemplaza el texto que éste insertó con el texto inmediato anterior en el kill-ring. Esto último se puede repetir sacando sucesivamente lo que esté guardado en el kill-ring.

Los elementos del kill-ring son el resultado de acomodar adecuadamente bloques de operaciones *matar*; por ejemplo si matas dos líneas seguidas, estas dos líneas forman parte de un mismo bloque en el kill-ring.

4.3.5. Regiones

Las divisiones o unidades que hemos revisado no son todas; durante una sesión de edición puedes requerir tomar partes arbitrarias del texto y operar sobre ellas. A estas divisiones se les conoce como *regiones*.

Una región comprende los carácteres que se encuentren entre una *marca* y el punto. La marca es otra posición especial dentro del buffer y se fija por medio del comando C–Space. El punto es la posición que tenga el cursor. Es decir, se coloca la marca en la posición donde se encuentra el punto y mueves el punto ampliando o contrayendo la región, de nuevo entendida como los carácteres entre la posición donde está la marca y la posición actual del punto. Emacs indicará visualmente la extensión de la región, ya sea mostrándola sombreada o cambiando el color del texto.

Existen muchas operaciones que puedes hacer sobre una región. Por el momento sólo mencionaremos dos. La primera es *matar* una región, que se lleva a cabo con el comando C-w y la otra, copiar una región al kill-ring sin matarla, que se lleva a cabo con el comando M–w. Esta última operación sólo mete la región al kill-ring sin quitarla de su ubicación actual.

4.3.6. Rectángulos

En Emacs, además de poder hacer selecciones de carácteres contiguos, también puedes seleccionar áreas con forma de rectángulos, con la esquina superior izquierda en la marca y la esquina inferior derecha en el punto, o viceversa. Los comandos que se usan para lidiar con rectángulos son:

Tabla 4.7 Emacs: comandos para manejar rectángulos

Enlace	Operación
C-x r k	Cortar (kill) un rectángulo
C-x r y	Pegar (yank) un rectángulo
C-x r o	Abrir un rectángulo
M–x clear-rectangle	Borrar un rectángulo

4.3.7. Registros

En Emacs existen una especie de *cajones*, en donde puedes guardar cadenas de texto, rectángulos o posiciones. Estos cajones reciben el nombre de *registros*, y sus principales comandos son:

Tabla 4.8 Emacs: comandos para manejar registros

Enlace	Operación
C-x x	Copia la región en un registro. Te pregunta por el nombre del registro.
C-x r r R	Guarda el rectángulo seleccionado en el registro R, donde R puede ser un carácter o un número.
C-x r s R	Guarda la selección en el registro R.
C-x r Space R	Guarda la posición del cursor en el registro R.
C-x r i R	Inserta el contenido del registro R, si éste es un rectángulo o una cadena.
C-x r j R	Salta al punto guardado en el registro R.
C-x r t	Agrega el texto (que te será pedido en el minibuffer) a todas las líneas en la región rectangular, desplazando el texto hacia la derecha.

4.3.8. Archivos

Como mencionamos en el capítulo anterior (sección 3.6.3), para abrir un archivo en Emacs utilizamos la secuencia: C–x C–f (find-file). Si te equivocas de archivo, inmediatamente después de utilizar C–x C–f, puedes utilizar el comando C–x C–v (find-alternate-file).

4.3.9. Buscar

Emacs incluye varios tipos de búsqueda, el comando search-forward utiliza el *mini-buffer* para solicitarte la cadena que deseas buscar.

Probablemente el comando más útil para buscar en Emacs es isearch-forward (nota la "i" al inicio del comando), que sirve para realizar una búsqueda de manera incremental. Esto significa que Emacs comienza a buscar conforme tú escribes la cadena de búsqueda. Es más usual utilizar el enlace C–s que el comando anterior.

Para buscar hacia atrás (o hacia arriba) de manera incremental utilizamos el comando isearch-backward o C–r.

Una vez que inicias una búsqueda incremental, ya sea hacia adelante o hacia atrás, puedes utilizar nuevamente C–s o C–r para buscar la siguiente presencia hacia adelante o hacia atrás, respectivamente. Cuando llegas al final (o inicio) del buffer, Emacs utiliza el área de eco para avisarte que no existen más presencias de la cadena buscada; si vuelves a insistir en buscar hacia ese mismo lado, Emacs *da la vuelta* y continúa la búsqueda desde el otro extremo del buffer.

4.3.10. Reemplazar

Habrá ocasiones en las que quieras buscar y cambiar una, algunas o todas las presencias de una cadena de texto por otra. Realizar esta tarea manualmente es tedioso, aun cuando puedes utilizar los comandos de búsqueda para localizar rápidamente el texto a cambiar.

Por este motivo Emacs te ofrece dos comandos principales para reemplazar texto. El primero replace-string, cambia todas las ocurrencias de una cadena por otra (utiliza el *mini-buffer* para solicitar ambas cadenas).

Existe sin embargo, otro comando llamado query-replace-string que se comporta similar, pero antes de realizar cada cambio presenta una serie de opciones para decidir si esa presencia particular se cambia o no y también para decidir reemplazar todas las presencias que se encuentren más allá del punto actual. Este último comando es más popular y por ello está asociado a una secuencia corta de teclas: M–%.

4.3.11. Guardar

Ya te hemos comentado que mientras editas un documento los cambios existen únicamente dentro de Emacs, en el buffer. Para hacer estos cambios permanentes debes salvar el archivo y esto puedes lograrlo con el comando: save-buffer o C–x C–s.

Con el comando save-some-buffers (C–x s) Emacs preguntará si deseas guardar cada buffer modificado en tu sesión. Finalmente, el comando write–file o C-x C-w te permite salvar el contenido del buffer en un archivo distinto. Emacs te preguntará el nombre de dicho archivo utilizando el *mini-buffer*.

4.3.12. Ventanas

Una ventana en Emacs es una subdivisión del área de trabajo en donde podemos desplegar un buffer. Con C–x 2, se crea otra ventana en forma horizontal. Para crear una ventana vertical puedes utilizar C–x 3.

Para cambiar entre ventanas debes utilizar C–x o. En cualquiera de estas ventanas puedes realizar todo tipo operaciones, ejecutar comandos de Emacs, etc.

Para cerrar todas las ventanas excepto en la que estás parado teclea C–x 1; para cerrar la ventana en la que estás parado teclea C–x 0.

4.3.13. Marcos (*frames)*

Además de las ventanas, en X también podemos crear lo que Emacs llama *frames*, que en terminología estándar de X se llama ventanas. ¿Confundidos? Es muy fácil: estos frames de Emacs lucen como ejecuciones distintas de Emacs, mientras que las ventanas son sub-divisiones de un mismo frame.

4.3.14. Ayuda en línea

No podríamos decir que Emacs es un editor de texto auto–documentado si no contara con un sistema de ayuda en línea. Si tecleas la secuencia C–h ? ?, Emacs te mostrará una lista con las opciones de ayuda que tiene a tu disposición. Por ejemplo, con la secuencia C–h k, Emacs te pedirá teclear una secuencia de teclas y te dirá todo lo que sabe de esa secuencia, a qué comando está asociada y la documentación del comando.

4.3.15. Lista de buffers

En Emacs podemos tener una gran cantidad de buffers sobre los cuales estamos trabajando, tantos que es fácil perder de vista o recordar el nombre exacto del buffer que deseamos editar. Para estas situaciones existe el comando list-buffers o su enlace C–x C–b, el cual despliega una lista de los buffers que tenemos abiertos en ese momento. En este buffer se despliega información básica de los buffers, su nombre, su tamaño, y, en caso de estar asociados con un archivo, el nombre del archivo. Otra información muy útil que nos proporciona este comando es si el buffer ha sido modificado desde la última vez que se guardó en disco.

También desde este buffer podemos ir a los diferentes buffers, posicionándonos en el reglón correspondiente y presionando Enter.

4.3.16. Anzuelos (*Hooks*), *setq* y otros

En esta sección no entraremos en muchos detalles, ya que para entender bien se requiere de conocimientos de *Emacs-Lisp*, que es el lenguaje para extender Emacs. Muchos de los modos mayores, los comandos, etc. que hemos revisado en este libro están escritos en Emacs-Lisp.

Con los *hooks* podemos hacer que Emacs ejecute funciones cuando suceda algún evento. Se usan mucho para llamar a ciertas funciones cuando abrimos un archivo de cierto tipo. Por el momento no entraremos en detalles acerca de los hooks.

Con setq le decimos el valor que queremos que tenga tal o cuál variable.

Cada modo tiene su mapa de teclado (por ejemplo, text-mode-map es el mapa para el modo mayor text) para indicar la acción que realiza cada tecla; estos mapas pueden modificarse para cambiar los comandos a ejecutar por estas teclas.

Con require hacemos que Emacs haga un reconocimiento de un archivo de definiciones.

4.3.17. Ortografía

Una carácterística interesante de la mayoría de los editores de texto modernos es que te ayudan a revisar la ortografía de tus documentos. Emacs no es la excepción, pero como muchas cosas con este gran editor, y en general en Linux, para lograrlo debes utilizar componentes externos, otros programas.

```
$ smart
install
aspell-es
```

Para lograr que Emacs revise tu ortografía vamos a utilizar la biblioteca *ispell.el*, que utiliza algún programa de Linux para realizar la revisión ortográfica. El programa predilecto

en Linux para revisar ortografía se llama *Aspell* y, dependiendo de tu instalación, puede o no contener un diccionario en español.

Con las siguientes líneas en tu ∼/.emacs le vamos a indicar varias cosas a Emacs. En la línea 1 le indicamos que queremos utilizar el programa de Linux aspell para revisar ortografía y, en las líneas 2 y 3 le indicamos detalles sobre el tipo de carácteres que incluye el español y algunos argumentos para el programa mismo. Por ejemplo, la secuencia -d es le indica a aspell que utilice el diccionario llamado *es*. Finalmente, en la última línea le indicamos que utilice por omisión este diccionario en todos los buffers.

```
1 (setq ispell-program-name "aspell"
2       ispell-extra-args '("-W" "3")
3       ispell-dictionary-alist
4       (cons
5        '("spanish" "[[:alpha:]]" "[^[:alpha:]]" "['.-]"
6          nil ("-B" "-d" "es") nil utf-8)
7        ispell-dictionary-alist)
8       )
9 (set-default 'ispell-local-dictionary "spanish")
```

Es importante sepas que con el comando ispell-change-dictionary puedes cambiar a otro diccionario. Estos cambios no son globales y funcionarán únicamente para el buffer actual.

Utilizando ispell

Ahora que hemos configurado un revisor ortográfico en Emacs, en la siguiente tabla te mostramos los comandos más importantes, con los que puedes revisar la ortografía de una palabra o de una porción del buffer.

Tabla 4.9 Emacs: comandos para revisar ortografía

Enlace	Operación
M–x flyspell-mode	Activa el modo *Flyspell* que resalta todas las palabras incorrectas.
M–$	Revisa y corrige errores ortográficos en la palabra en el punto.

Continúa en la siguiente página

Continúa de la página anterior

Enlace	Operación
M–Tab	Completa la palabra antes del punto basado en una ortografía del diccionario.
M–x ispell	Revisa la ortografía en la región o buffer activo.
M–x ispell–buffer	Revisa la ortografía en el buffer.
M–x ispell–region	Revisa la ortografía de la región.

4.3.18. Extendiendo Emacs

Dado que Emacs es y ha sido utilizado por miles de programadores a lo largo de varias décadas, la manera más sencilla de extenderlo es investigar si esa extensión deseada no fue ya implementada por alguien más. Utiliza un buscador de web para localizar extensiones.

Una vez localizada la extensión o paquete, debemos decidir entre realizar una instalación global (para todos los usuarios del sistema) o una instalación personal. Si este equipo es utilizado por mucha gente, conviene hacerlo de manera global, pero después, cuando actualices tu sistema operativo, debes repetir todo el procedimiento. En general, es más fácil y recomendable instalar extensiones a Emacs de manera personal y aquí te mostramos cómo hacerlo.

Preparatorios

Los siguientes pasos te servirán para instalar cualquier programa para extender Emacs, por lo que es importante realizarlos de una vez.

i. Crea el directorio ∼/elisp (puedes usar otro nombre, pero entonces recuerda cambiar elisp por el nombre que elegiste más adelante).

La variable load-path

ii. En tu ∼/.emacs ahora agrega este directorio a la ruta de búsqueda de Emacs. Esta ruta se parece mucho a la variable de ambiente PATH de Unix y, en realidad, tienen una función similar:

```
(add-to-list 'load-path (expand-file-name "~/elisp/"))
```

y listo: a partir de ahora, cada vez que entres a Emacs, éste sabrá que todos los programas (archivos que tienen extensiones .el o .elc dentro de el directorio elisp son extensiones.

Instalando paquetes en 1, 2, 3

Aunque el procedimiento para instalar paquetes varía grandemente de uno a otro, en general tienes que seguir los siguientes pasos:

1. Localizar el archivo y descargarlo de la red. Si está comprimido, tienes que expandirlo.
2. Copiarlo o moverlo a tu directorio ∼/elisp. Si es un archivo con extensión .el, pasa al siguiente paso; si es un directorio, tienes que agregar el directorio a la ruta de búsqueda. Por ejemplo, si el directorio se llama *foo*, agregas esto a tu ∼/.emacs:

   ```
   % (add-to-list 'load-path (expand-file-name "∼/elisp/foo"))
   ```

   Usualmente cuando el paquete viene en un directorio, incluye instrucciones especiales para instalación. Generalmente incluyen un archivo README o INSTALL con instrucciones. En estos casos puedes revisar el procedimiento explicado en el Anexo B para instalar programas paso a paso.
3. Después tienes que leer el archivo principal del paquete y copiar las líneas de iniciación a tu ∼/.emacs. Generalmente tienen la forma: (require 'foo), donde *foo* es el nombre del archivo con extensión .el.

En los siguientes capítulos de este libro utilizaremos este procedimiento para instalar algunas extensiones importantes de Emacs. A manera de ejemplo, sin embargo, a continuación agregaremos una extensión muy sencilla para modificar los colores de Emacs.

Poniendo color a tu vida

Dependiendo del tipo de archivos que editas, Emacs reconoce distintas unidades sintácticas de esos archivos. Por ejemplo, cuando programas reconoce palabras reservadas, cadenas, etc. Para activar los colores de manera global agrega lo siguiente a ∼/.emacs:

```
(require 'font-lock)
(add-hook 'font-lock-mode-hook 'turn-on-fast-lock)
(global-font-lock-mode 1)
```

¿Qué puedes hacer si no te gustan los colores de Emacs? La respuesta sencilla sería cambiarlos. Por desgracia, la cantidad de variables que controlan los colores en Emacs es muy grande, por lo que esta tarea sería muy complicada.

Afortunadamente hay alternativas; por ejemplo, podemos instalar la extensión de Emacs, color-theme, que puedes obtener de la siguiente dirección:

http://download.gna.org/color-theme/

Después de instalar y compilar el programa, agrega esto a tu ∼/.emacs:

```
(add-to-list 'load-path (expand-file-name "~/elisp/color-theme"))
(require 'color-theme)
(color-theme-initialize)
```

Ahora puedes ejecutar el comando M-x color-theme-select y color-theme creará un nuevo buffer con una lista de los temas disponibles para darle color y vida a tu Emacs.

Puedes recorrer la lista y con Enter puedes seleccionar el tema en la línea actual. Cuando seleccionas un tema, inmediatamente Emacs ajustará todos los colores.

Recorre la lista y selecciona el tema que más te guste y, una vez que tengas tu favorito, agrega el comando que lo activa a tu ∼/.emacs; así, cada vez que regreses, Emacs recordará tu tema predilecto. Por ejemplo:

```
(color-theme-charcoal-black)
```

Internet 5

5.1. Redes de computadoras

La palabra *red*, cuando se habla de computadoras, hace referencia a dos o más computadoras conectadas entre sí. Las computadoras se conectan en red con el propósito de compartir recursos.

Cuando las computadoras están conectadas directamente (usando algún tipo de cable), el sistema se denomina *red de área local o LAN*a (por su nombre en inglés *Local Area Network*). Una LAN típica se encuentra en un solo edificio o en algunas ocasiones en el mismo piso. Sin embargo, la conectividad no tiene por qué terminar en una red local. Varias LAN están conectadas a otras redes, formando lo que se denomina *Red de Área Amplia o WAN* (*Wide Area Network*).

En varias organizaciones las computadoras están conectadas entre sí por redes de área local. En ocasiones, estas LAN están conectadas con un enlace de alta velocidad (denominado soporte principal o *backbone*) que reúne a las LAN pequeñas para formar una red de área amplia. Las organizaciones grandes, con muchos departamentos, pueden tener más de un soporte principal. Esto significa, entre otras cosas, que es posible enviar un correo electrónico desde cualquier computadora dentro de la organización hacia cualquier otra computadora (siempre y cuando estén conectadas por la red). De hecho, enviar un mensaje que atraviese todo el país puede ser tan sencillo como enviar un mensaje hacia el otro extremo de un corredor.

5.1.1. Relación cliente-servidor

En términos propios de una red, cualquier programa que ofrezca un recurso recibe el nombre de servidor. Un programa que usa un recurso es un cliente.

Si un programa proporciona acceso a los archivos en una red, recibe el nombre de servidor de archivos; un programa que coordina la impresión de los datos usando varias impresoras se denomina servidor de impresión; en general, cualquier programa que ofrece un servicio y recibe solicitudes para el mismo recibe el nombre de servidor[1].

Las redes de muchas organizaciones están conectadas a redes regionales y nacionales muy grandes. Dentro de estos sistemas, ciertas computadoras, denominadas puertas o puentes (*gateway*), actúan como enlaces entre la red y el mundo exterior.

En todo el mundo las redes de área amplia más importantes están conectadas a un sistema conocido como *Internet*. Toda computadora en Internet puede conectarse a cualquier computadora que también se encuentre en Internet.

La mayoría de los países tienen redes nacionales grandes que están conectadas a Internet mediante puertas. Con eficiencia, este sistema enlaza cientos de miles de computadoras, formando una red mundial enorme. Esto significa que uno puede, con poco esfuerzo, enviar correo electrónico y archivos de datos a todo el mundo. Existen incluso puertas para redes de cómputo comerciales. Por ejemplo, también se pueden intercambiar mensajes con personas que usen América Online, CompuServe, MCI, Mail, Delphi, Prodigy, AT&T Mail, entre otros prestadores de servicios.

Cuando una computadora da el servicio de servidor y permite, a su vez, trabajo en ella, a esta computadora se le conoce como anfitrión (*host*).

Son expertos los que configuran y administran las conexiones de red descritas anteriormente. Estas conexiones, por lo general, usan cables o líneas telefónicas rentadas. Las conexiones a gran escala usan enlaces vía satélite. Sin embargo, puede usarse una línea telefónica común y hacer que una computadora llame por teléfono directamente a la computadora anfitrión o a un ruteador (*router*), estableciéndose la conexión. En términos técnicos, la conversión de señales de computadora a señales telefónicas se llama modulación. El proceso inverso se denomina demodulación. Por tal motivo, el dispositivo que se requiere para conectarse a una computadora se llama *módem* (modulador/demodulador).

Ahora bien, recuerda que la única manera de trabajar con una computadora anfitrión UNIX es mediante una terminal. Como la computadora que se tiene en casa no es una terminal, el programa de comunicaciones que se utilice debe hacer que la computadora casera actúe como una terminal. Se dice entonces que el programa está emulando una terminal. Los sistemas UNIX pueden trabajar con muchas marcas y tipos de terminales, pero existe una que por costumbre es la que se emula cuando se trabaja con una línea telefónica. Se trata de la terminal tipo VT100.

[1] En ocasiones el nombre *servidor* se emplea para referirse a una computadora y no a un programa.

5.1.2. TCP/IP

Internet es una colección de redes en todo el mundo que contiene muchos tipos de sistemas. Algo debe mantenerlos unidos. Ese algo es `TCP/IP`.

Los detalles de TCP/IP son altamente técnicos y superan las expectativas de este material. Sin embargo, hablaremos de manera general sobre el tema.

TCP/IP es un nombre común que se aplica a una colección de más de cien protocolos (un protocolo es un conjunto de reglas que permiten que diferentes máquinas y programas se coordinen entre sí). Los protocolos TCP/IP derivan de dos protocolos básicos: TCP que es acrónimo de *Transmition Control Protocol* e IP que es acrónimo de *Internet Protocol*. En esencia, estos protocolos se utilizan en Internet (y en otras redes) para conectar computadoras, equipo de comunicaciones y programas. Cabe mencionar que no todas las computadoras en Internet ejecutan UNIX, pero lo que es seguro es que todas usan TCP/IP.

Dentro de un sistema TCP/IP, los datos transmitidos se dividen en pequeños paquetes. Cada paquete contiene el domicilio de la computadora del receptor junto con un número de secuencia. Por ejemplo, el sistema puede dividir un mensaje en diez paquetes, cada uno con su propio número de secuencia. Estos paquetes se envían por la red y cada uno se transporta individualmente hacia su destino. Cuando se han recibido todos los paquetes, el sistema destino usa los números de secuencia para juntarlos. Si por alguna razón uno de dichos paquetes llega en mal estado, el sistema destino transmite un mensaje al emisor señalándole que debe volver a enviar determinado paquete.

Tiene varias ventajas dividir los datos en paquetes:

- Las líneas de comunicación se pueden compartir entre varios usuarios.
  Se pueden transmitir al mismo tiempo distintos tipos de paquetes, ya que se ordenarán y combinarán hasta que lleguen a su destino.
- Los datos no tienen que enviarse directamente entre dos computadoras.
  Cada paquete pasa de computadora en computadora hasta llegar a su destino.
- Rapidez, ya que sólo se necesitan unos segundos para enviar un archivo de gran tamaño de una máquina a otra, aunque estén separadas por miles de kilómetros y pese a que los datos tienen que pasar por múltiples computadoras. De esta forma, si al enviar los paquetes algo sale mal con al menos uno de ellos, sólo es necesario volver a transmitir ese paquete y no todo el mensaje.
- Los paquetes no necesitan seguir la misma trayectoria.
  La red puede llevar cada paquete de un lugar a otro y usar la conexión más idónea que se encuentre disponible en ese preciso momento. Esto implica que no necesariamente los paquetes deben viajar por la misma ruta y por supuesto, tampoco tienen que llegar todos al mismo tiempo a su destino.
- El proceso que TCP sigue cuando se envía un mensaje es el siguiente:
  i. TCP divide los datos en paquetes.

ii. Ordena éstos en secuencia.

iii. Agrega cierta información para el control de errores.

iv. Y después los distribuye.

- El proceso en el otro extremo es:
  i. TCP recibe los paquetes.
  ii. Verifica si hay errores.
  iii. Y los vuelve a combinar para convertirlo en los datos originales.
  iv. Si existe algún error, el programa TCP destino envía un mensaje solicitando que se vuelvan a enviar determinados paquetes.

La tarea de IP es llevar los datos a granel (es decir, los paquetes), de un sitio a otro. Las computadoras que encuentran las vías para llevar los datos de una red a otra, ruteadores, utilizan IP para trasladar los datos.

Si el usuario está interesado en ver la ruta que va desde su computadora hasta otra, puede usarse el comando traceroute. El problema es que este comando no se encuentra disponible en todos los sistemas. El comando se usa de la siguiente manera:

```
% traceroute computadora
```

5.2. Web

Internet, como ya mencionamos, es una colección mundial de redes que transmiten datos usando el protocolo IP. Ahora la pregunta a resolver es ¿qué se puede hacer en Internet? Son siete los servicios principales que Internet puede ofrecer:

Correo: Se pueden enviar y recibir mensajes.

Transferencia de archivos: Se pueden copiar archivos de una computadora a otra.

Entrar a un sistema remoto: Podemos entrar a otra computadora y trabajar en ella como si nuestra terminal estuviera conectada directamente a ella.

Grupos de interés (Usenet): Se pueden leer y escribir artículos en los miles de grupos de interés de Usenet (también llamados foros).

Software compartido: Podemos obtener copias *gratuitas* de cualquier tipo de software y nosotros mismos podemos compartir nuestros programas con otras personas.

Acceso a información: Se puede buscar y recuperar cualquier tipo de información. Si no se está seguro de dónde buscar, existen programas especiales que nos pueden ayudar.

Comunicación con otras personas: Se puede sostener una conversación con otras personas usando el teclado y la pantalla (puede ser una sola persona o incluso un grupo).

5.3. Redes y direcciones

Una vez que se nos asigna una cuenta UNIX, podemos comunicarnos con todos los integrantes de la comunidad UNIX y transferir datos a todo el mundo. Lo único que se necesita es un domicilio y saber usar los programas en red.

5.3.1. Domicilios estándar en Internet

La clave para poder usar el Internet consiste en entender qué son los domicilios o correo electrónico. Lo más asombroso del esquema de los domicilios en Internet es que cada identificador de usuario y cada computadora necesitan un solo domicilio, sin importar el servicio que utilicen.

Un domicilio en Internet consta de un identificador de usuario, seguido por el signo @ y después el nombre de la computadora (compuesto por el mismo nombre de la computadora y el de su dominio). De tal forma que el dominio estándar de Internet luce así: *identificador@computadora.dominio*. Por ejemplo, el correo electrónico de nuestro usuario es Juan.Perez@ciencias.unam.mx.

En este ejemplo, el identificador de usuario es *Juan.Perez* y el dominio (el nombre o domicilio de la computadora) es *ciencias.unam.mx*.

Las partes del dominio que están separadas por puntos se llaman subdominios. El domicilio anterior tienen tres subdominios: *ciencias.unam.mx*. El subdominio más a la derecha (en este caso, *mx*) se denomina dominio de jerarquía superior.

Para leer un domicilio, se examinan los dominios de derecha a izquierda. Así vamos del nombre más general al más particular[2] .

5.3.2. Nomenclatura antigua e internacional

Existen dos tipos de dominios de jerarquía superior: en nomenclatura antigua y en nomenclatura internacional moderna. El formato antiguo se emplea principalmente en Estados Unidos y Canadá y consiste en siete dominios de jerarquía superior, que son:

[2]Nota: los domicilios de Internet no son sensibles a mayúsculas y minúsculas, por lo que es lo mismo teclear una dirección con letras mayúsculas o con letras minúsculas o con una mezcla de ambas. Aunque es una buena costumbre teclear cualquier dirección sólo en minúsculas.

Tabla 5.1 Dominios de jerarquía superior

Dominio	Significado
com	Organización comercial
edu	Institución educativa
gov	Institución gubernamental
int	Organización internacional
mil	Ejército
net	Organización de redes
org	Organización no lucrativa

Este tipo de domicilio se desarrolló para la antigua red *Arpanet*, ancestro de Internet, y se diseñó sólo para usarse en Estados Unidos.

Cuando Internet se expandió, fue claro que se necesitaba un mejor sistema. La solución consistió en usar dominios de jerarquía superior que representaban a países. Para la mayoría de los países, el dominio de jerarquía superior es la abreviatura internacional de dos letras.

Fuera de los Estados Unidos, virtualmente todos los sitios enlazados con Internet usan la nomenclatura de domicilios internacional moderna, donde el dominio de jerarquía superior muestra el código del país. Dentro de los Estados Unidos (y parte de Canadá), la mayoría de las computadoras en Internet aún utilizan la antigua nomenclatura de domicilios; es decir el estilo con los dominios de jerarquía superior.

5.4. WWW

WWW significa *World Wide Web* o simplemente *web*. Es un proyecto desarrollado por Tim Berners. El web es una colección de documentos interconectados en todo el mundo.

Los documentos en el web son usualmente conocidos como páginas web y poseen *hipervínculos* que enlazan una porción de texto o una imagen a otro documento, normalmente relacionado con el texto o la imagen.

Para acceder al web debes utilizar un navegador (*browser*). El navegador lee documentos dispuestos en el web y puede buscar documentos de otras fuentes. La información es provista por los servidores de hipertexto de los cuales se pueden obtener los documentos.

Los navegadores pueden también acceder a archivos vía FTP, NNTP, y otros protocolos válidos para Internet. También permiten búsquedas en documentos y bases de datos. Los documentos que los navegadores despliegan son documentos de hipertexto. El término *hipertexto* se define como texto con apuntadores a otros textos. Los navegadores tratan con los apuntadores de manera transparente. La ventaja del hipertexto es que en un documento

en hipertexto, si quieres más información acerca de un tema en particular que fue mencionado, usualmente puedes hacer click sobre el tema para leer con más detalle. De hecho, los documentos pueden estar ligados a otros documentos de autores completamente diferentes, mucho como las referencias al pie de página, pero puedes sumergirte en ella de forma instantánea.

El término *hipermedia* es un super conjunto de hipertexto; esto es, cualquier medio con apuntadores a otro medio. Esto significa que los navegadores pueden desplegar no sólo archivos de texto, sino imágenes, sonidos o animaciones.

Existen diferentes sitios donde todos los usuarios pueden encontrar todo tipo de información y hacer búsquedas en la red. Por ejemplo:

- Google: *www.google.com*
- Yahoo: *www.yahoo.com*

5.4.1. URL: especificación de objetos en Internet

URL significa *Uniform Resource Locator*. Es un estándar para especificar un objeto en Internet, tal como el nombre de un archivo. Los URL se ven como los que acá siguen:

- ftp://ftp.fciencias.unam.mx/pub/cursos/677/tarea.txt
- http://www.fciencias.unam.mx/admin/directorio.html

La primera parte del URL, antes de los dos puntos, especifica el método de acceso. La parte del URL después de las dos diagonales indica el nombre de la máquina y cada diagonal separa la ubicación del archivo dentro de dicha máquina; en el caso del primer ejemplo mencionado anteriormente se lee como: dentro de la máquina ftp.fciencias.unam.mx, utilizando el protocolo FTP, en la ruta pub/cursos/677 encontrarás el archivo tarea.txt.

El método de acceso *http* es acrónimo de *HiperText Transfer Protocol* y es el método de acceso más común, por lo que si no aparece un método de acceso explícitamente, la mayoría de los sistemas suponen que se trata de *http*.

5.4.2. Navegadores

El primer navegador fue desarrollado por Tim Berners-Lee en el CERN a finales de 1990 y principios de 1991. Después surgió el navegador Mosaic, que funcionaba inicialmente en plataformas UNIX sobre X11; fue el primero que se extendió debido a las versiones para Windows y Macintosh que había. Después entró al mercado Netscape Navigator, el cual superó en capacidades y velocidad a Mosaic. Este navegador tiene la ventaja de funcionar en casi todos los UNIX, así como en ambientes Windows.

Microsoft poco después sacó el navegador Internet Explorer, desbancando así a Netscape Navigator. En años recientes Netscape Communications Corporation liberó el código

fuente de su navegador y de esta manera nació el proyecto Mozilla. Sin embargo éste fue reescrito desde cero tras decidirse a desarrollar y usar como base un nuevo conjunto de herramientas multiplataforma basado en XML llamado XUL.

A finales de 2004 apareció en el mercado Firefox, una rama de desarrollo de Mozilla que pretende hacerse con parte del mercado de Internet Explorer. Se trata de un navegador más ligero que Mozilla.

La comunicación entre el servidor web y el navegador se realiza mediante el protocolo HTTP. Su función principal es mostrar documentos HTML en pantalla. En la actualidad, no solamente se descargan este tipo de documentos sino también documentos con imágenes, sonidos e incluso videos en diferentes formatos. Además, permiten almacenar la información en el disco o crear marcadores (*bookmarks*) de las páginas más visitadas.

Algunos de los navegadores web más populares se incluyen en lo que se denomina una *Suite*. Estas Suites disponen de varios programas integrados para leer noticias de Usenet y correo electrónico mediante los protocolos NNTP, IMAP y POP.

Actualmente existen diferentes navegadores y todos cumplen con el mismo objetivo: permitir al usuario recuperar y visualizar documentos de hipertexto, comúnmente descritos en HTML, desde servidores web de todo el mundo a través de Internet. Algunos de los más utilizados en el mundo Linux son:

- Firefox: http://www.mozilla.com
- Opera: http://www.opera.com/
- Konqueror (integrado con KDE)
- Galeon (integrado con Gnome)

Navegadores web basados en texto:

- Lynx: http://lynx.browser.org/
- w3m: http://w3m.sourceforge.net/

Ya hemos utilizado Konqueror con anterioridad, pero lo hicimos para recorrer el sistema de archivos; ahora lo utilizaremos además para navegar el web.

Konqueror te da la oportunidad de teclear un URL directamente en el campo location (o en la ventana de dialogo de URL File|Open|Location). Al usar el URL, Netscape traerá la página especificada tal como si hubieses hecho *click* en la liga. Nota que la etiqueta en el campo location expresa la ubicación después de que trae la página, o Go to si editas el campo.

Actividad 5.1

(a) Abre un navegador y visita la página de la Facultad de Ciencias de la U.N.A.M. ubicada en http://www.fciencias.unam.mx

(b) Localiza los servicios en línea para estudiantes.

5.4.3. Google

Google es una compañía cuyo principal producto es el motor de búsqueda del mismo nombre. Fue fundada el 27 de septiembre de 1998 por dos estudiantes de doctorado de la Universidad de Stanford, Larry Page y Sergey Brin. Aunque su principal producto es el buscador, la empresa también cuenta con otros servicios: un comparador de precios llamado Froogle, un motor de búsqueda para material almacenado en discos locales llamado Google Desktop Search y un servicio de correo electrónico gratuito llamado Gmail, que ha revolucionado los sistemas de correo electrónico en el web.

A continuación se listan algunos de los servicios que ofrece Google actualmente:

Tabla 5.2 Servicios comunes de Google

Buscadores	**Servicios**
Imágenes	Gmail
Grupos	Video
Directorio	Maps
Noticias	Calendar
	Earth
	Talk
	Books

5.4.4. Wikipedia

La wikipedia es una enciclopedia gratuita a la cual todos los usuarios de Internet tenemos acceso. Está escrita por todos aquellos voluntarios que quieran colaborar con sus conocimientos y exponerlos en este medio. Actualmente existe mucha gente que va mejorando la wikipedia, con cientos de ediciones por hora, que son registradas específicamente en el historial y generalmente en los cambios recientes del sitio. Si alguna persona expone sus ideas en forma inapropiada o simplemente escribe disparates, las personas encargadas del contenido del sitio lo verifican y descartan este tipo de información.

La wikipedia se encuentra disponible en varios idiomas, con distintos niveles de avance, ya que la traducción de artículos y entradas de la enciclopedia la realizan voluntarios. Entre

otros idiomas encontramos inglés (el de mayor cobertura), español, francés y alemán.

La carácterística principal de la wikipedia es la posibilidad de editar los artículos de la enciclopedia. En general, a los sitios web que permiten la edición (en línea y sin mayores restricciones) de su contenido se les conoce como *wikis*.

El objetivo de la wikipedia es almacenar artículos sobre temas considerados enciclopédicos; sin embargo dentro de ésta puedes encontrar otros proyectos de contenido diverso, como:

Tabla 5.3 Otros servicios de la Wikipedia

Servicio	Descripción
Wikinoticias	El noticiario libre
Wikiversidad	La universidad libre
Wikilibros	La casa editorial libre
Wikisource	La biblioteca libre
Wikiquote	Citas y frases famosas
Meta-Wiki	Coordinación de proyectos
Wikiespecies	Directorio de especies

Cada uno de estos sitios se rigen bajo la misma filosofía, permitirle a los usuarios que así lo deseen, colaborar con la ampliación del contenido del mismo, para de esta forma ampliar cada uno de estos proyectos en el web de manera gratuita. Los temas ahí descritos son textos originales de dominio público.

5.5. Los programas de correo de Unix

Como ya habíamos mencionado, Unix viene con un sistema de correo incorporado. La parte más importante de este sistema es su interfaz de usuario, es decir el programa que se utiliza para enviar y leer correo.

Originalmente había en todo sistema Unix un programa sencillo llamado *mail*. Era adecuado aunque con limitaciones. Después fue reemplazado por *mailx* y *Mail*, que en realidad son bastante similares entre sí.

Existen otros más como *pine, elm, mush, rmail*, y algo que tienen en común estos sistemas es que extienden de gran forma la funcionalidad de *mail*, ofreciendo carácterísticas adicionales. Sin embargo nosotros veremos VM como lector de correo.

5.6. Netiquete

La palabra *netiquete* se deriva de dos raices, *Net*, que significa red y *etiquete* que se refiere az las reglas de etiqueta para comportarse en un cierto ámbito, por lo que el término se refiere a la etiqueta que debemos seguir cuando usemos la red, es decir, son ciertas reglas (de etiqueta) que sería deseable seguir cuando estemos utilizando cualquier forma de comunicación a través de la red. Estas reglas no son obligatorias pero si deseables. En resumen Netiquete se refiere a la forma adecuada de comportarse en Internet.

Antes de entrar en materia, es esencial que cada usuario en la red reconozca su responsabilidad, al tener acceso a servicios, sitios, sistemas vastos y personas, pues es éste el responsable por sus acciones al acceder a servicios de la red.

"Internet", o "la red", no es una simple red; es más que eso: un grupo de miles de redes individuales que han escogido permitir pasar tráfico entre ellas. El tráfico enviado a Internet puede atravesar varias redes diferentes antes de llegar a su destino. Por eso, los usuarios involucrados en estas redes interconectadas deben de estar informados de la carga que imponen en otras redes participantes.

Como un usuario de la red, se te permite acceder a otras redes (y/o los sistemas de computadora unidos a esas redes). Cada red o sistema tiene su propio grupo de políticas y procedimientos. Acciones que están rutinariamente permitidas en una red pueden estar controladas, o igualmente prohibidas en otra red. Es responsabilidad de los usuarios regirse por las políticas y procedimientos de estas otras redes. Recuerda, el hecho que un usuario "pueda" ejecutar una acción particular no implica que "deba" tomar tal acción.

El uso de la red es un privilegio, no un derecho, que puede temporalmente ser revocado en cualquier momento, por conducta abusiva. Tal conducta incluiría el poner información ilegal en un sistema; el uso abusivo o de, por otra parte, lenguaje inaceptable tanto en mensajes públicos o privados; el envío de mensajes que den como resultado la pérdida de trabajo del destinatario o de sus sistemas; el envío de çartas cadena"; o la "transmisión"de mensajes a listas o individuos y cualesquiera otros tipos de uso que causen congestión de la red o que interfieran con el trabajo de otros.

5.6.1. Algunos puntos del netiquete para comunicación electrónica

Éstas son algunas de las reglas que todo usuario de Internet debería cumplir:

- El volumen y mantenimiento del buzón electrónico de un usuario es responsabilidad del propio usuario.
- Cuando cites a otra persona, borra cualquier cosa que no sea directamente aplicable a tu respuesta. No permitas que tu email o software de Usenet cite automáticamente todo el cuerpo de los mensajes que se responden cuando no sea necesario. Tómate el

tiempo de editar lo citado a un mínimo como para proveer contexto a la respuesta. A nadie le gusta leer un mensaje largo citado tres o cuatro veces, sólo para ser seguido por una línea de respuesta como: "Sí, yo también".

- Usa palabras en mayúscula sólo para destacar un punto importante o distinguir un título o cabecera. El usar palabras en mayúscula que no sean títulos suele ser considerado como GRITOS.
- Nunca mandes cartas “cadena” por Internet. El hacerlo puede causarte la pérdida de tu acceso a Internet.
- Sé cuidadoso cuando uses sarcasmo y humor. Sin comunicaciones cara a cara tu chiste puede verse como una crítica. Cuando trates de ser gracioso, usa *emoticons* para expresar humor. (Voltea tu cabeza hacia la izquierda para ver el emoticon sonrisa: :-) = carita de humor feliz).
- Recuerda que algunas listas de discusión y grupos Usenet tienen miembros de muchos países.
  - No des por hecho que ellos van a entender una referencia a un programa de TV, películas, cultura popular o hechos actuales de tu propio país. Si debes usar la referencia, por favor explícala.
  - No des por hecho que ellos van a entender referencias geográficas que son locales o nacionales.
- **Si** alguien envía un comentario o pregunta fuera del tema de la lista, **no** respondas a la lista y **no** mantengas el tema de fuera de la lista en la conversación pública.

Los 10 mandamientos de ética para la computadora

1. No deberás usar una computadora para dañar a otras personas.
2. No deberás interferir con el trabajo de computadora de otras personas.
3. No deberás entrometerte en los archivos de otras personas.
4. No deberás usar una computadora para robar.
5. No deberás usar una computadora para dar falso testimonio.
6. No deberás usar o copiar software de manera que violes su licencia.
7. No deberás usar recursos de computadora de otras personas sin autorización.
8. No deberás apropiarte del trabajo intelectual de otras personas.
9. Deberás pensar acerca de las consecuencias sociales del programa que escribes.
10. Deberás usar una computadora en formas en que muestres consideración y respeto.

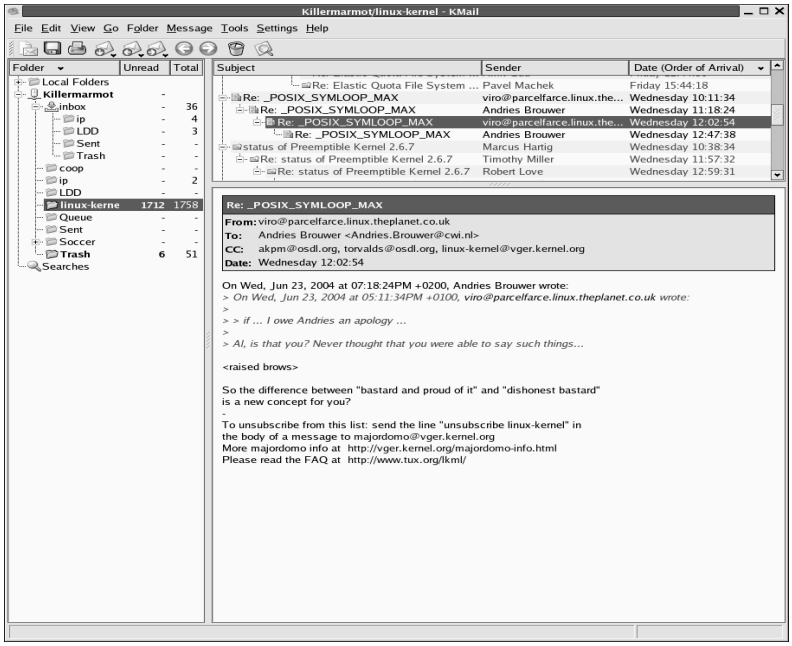
5.7. KMail: lector de correo electrónico

5.7.1. Introducción

$ smart install KMail

KMail es un cliente de correo electrónico dentro del ambiente de escritorio de KDE y parte del sistema Kontact. Soporta carpetas, filtrado, visualización de correo con HTML y carácteres internacionales. Permite enviar correo a través de un servidor de correo vía SMTP o Sendmail y recibir correo por los protocolos POP3 o IMAP. También tiene compatibilidad para filtrar mensajes de correo a través de antivirus o antispam que se encuentren instalados en el sistema. En la figura 5.1 puedes ver la pantalla de KMail.

Figura 5.1 Ventana principal de KMail



5.7.2. Utilizando KMail

Seguramente ya habrás utilizado algún otro lector de correo, pero aquí te presentaremos algunas de las ventajas que tiene KMail:

- **Filtros de mensajes.**
  Muchas veces se tienen problemas para ordenar los nuevos mensajes cuando llegan a la bandeja de entrada. Los filtros te permiten realizar ciertas tareas automáticamente sobre los mensajes entrantes y llevar a cabo acciones manualmente sobre los mensajes seleccionados en una carpeta.
  Los filtros constan de criterios de filtrado, cuyas reglas determinan si un filtro aplica a un mensaje; y una lista de acciones de filtro, que describe qué hacer con el mensaje. Los filtros se consideran uno después de otro, comenzando por el primero de la lista. El primero en el que coincida el patrón de búsqueda será ejecutado.
  - **Patrones de búsqueda:** Los criterios por los que un filtro puede buscar:

    Mensaje. Busca en todo el mensaje (es decir cabeceras, cuerpo y adjuntos, si los hay).
    Cuerpo. Busca en el cuerpo del mensaje (es decir, todo el mensaje salvo las cabeceras).
    Cualquier cabecera. Busca en las cabeceras del mensaje.
    Destinatarios. Busca en los campos "Para" y "CC" del encabezado del mensaje.
    Tamaño en bytes. Establece los límites inferiores o superiores del tamaño del mensaje.
    Edad en días. Establece los límites superiores o inferiores de la antigüedad del mensaje.
    Estado. Define las restricciones al estado del mensaje.
    Cualquier otro nombre. Busca el campo de la cabera indicado por ese nombre.

  - **Acciones de filtrado.**
    El uso más frecuente de los filtros es el de ordenar los mensajes entrantes en diferentes carpetas. Las posibles acciones son:
    Mover a carpeta. Esto moverá el mensaje a otra carpeta, eliminándolo de su carpeta actual si fuera necesario.
    Copiar en carpeta. Copiará el mensaje en otra carpeta.
    Definir identidad. Definirá la identidad que se va a usar si respondes al mensaje.
    Marcar. Esto permite marcar los mensajes como leído, importante, reenviado o contestado.
    Enviar MDN falso. Enviará un mensaje de notificación de disposición falso (es decir un recibo de lectura) al remitente del mensaje.
    Definir transporte. Esto establecerá el método de transporte (por ejemplo SMTP) que se va a usar si respondes al mensaje.
    Definir Responder. Modificará el campo Responder-a de este mensaje. Esto es útil para la listas de correo.
    Reenviar. Reenviará el mensaje en la línea a otras direcciones de correo electrónico.
    Redirigir. Redirigirá el mensaje como si hubiera sido para otra dirección de correo electrónico.

Confirmar entrega. Intentará devolver un mensaje al remitente indicando que el mensaje ha sido entregado correctamente, si es que el remitente lo solicitó.

Ejecutar orden. Ejecutará un programa, pero no modificará el mensaje. Aquí se pueden introducir líneas de comando complejas, ya que KMail hace uso de un intérprete de comandos para ejecutarlas.

A través de tubería. Esto enviará el mensaje a un programa.

Eliminar cabecera. Eliminará todos los campos de la cabecera del mensaje que contengan el nombre dado.

Añadir cabecera. Se añadirá un nuevo campo de cabecera con el nombre dado y el valor para el mensaje.

Reescribir cabecera. Buscará el campo de cabecera dado, modificará su contenido y lo volverá a escribir. La cadena de búsqueda se interpretará como una expresión regular sensible a las mayúsculas.

Reproducir sonido. Reproducirá el sonido especificado.

- **Optimización de filtrado.**
  El orden de los filtros tiene un impacto sobre la velocidad del proceso de filtrado. Algunas formas de mejorar el filtrado son:
  - Detén el proceso de filtros tan pronto como sea posible.
  - Considera el costo de la evaluación de las reglas de filtrado.
  - Comprueba el orden de tus filtros.

- **Firmar y cifrar mensajes con PGP o GnuPG.**
  Para configurar y usar el soporte PGP de KMail, es necesario tener PGP instalado y configurado adecuadamente.
  Asimismo, es necesario generar un par de claves (clave secreta y clave pública) para su identidad. Puedes hacer esto con la línea de comandos pgp –kg o gpg – –gen-key respectivamente.
  Para una mejor explicacion de este tema puedes revisar la seccion 5.10.
  Entre las opciones para firmar y cifrar mensajes se encuentran:

  - Firmar mensajes. Edita tu mensaje como de costumbre en la ventana del editor de KMail. Antes de enviar el mensaje, marca el icono Firmar mensaje en la barra de herramientas de la ventana del editor. Luego, puedes enviar el mensaje.
  - Cifrar mensajes. Para enviar un mensaje cifrado a alguien del que tienes una clave pública, simplemente edita el mensaje en la ventana del editor. Antes de enviar el mensaje, selecciona Cifrar mensaje en la barra de herramientas de la ventana del editor. Luego envía el mensaje.
  - Enviar tu clave pública. Elije en el menú de la ventana del editor el submenú Adjunt->Adjuntar clave pública. Esto adjuntará la clave pública que hayas definido para la identidad de ese mensaje.
  - Recibir un mensaje cifra. Selecciona el mensaje y escribe tu frase de paso. Luego,

KMail intentará descifrar el mensaje y lo mostrará en texto plano si ha conseguido descifrarlo con tu clave pública.

- Recibir una clave pública. Puedes recibir una clave pública como un adjunto, o vía http o ftp, para cifrar un mensaje al propietario de la clave. Luego puedes añadir esta clave junto con el resto de claves públicas tecleando

  pgp −−ka nombre_archivo

  en la línea de órdenes (usando PGP) o tecleando

  gpg −−import nombre_archivo

  en la línea de órdenes (usando GnuPG).

- **Anti correo basura.**
  KMail usa herramientas externas y especializadas para el correo basura. Algunas de las herramientas admitidas son:

  Bogofilter. Bogofilter es un filtro que detecta spam en la fase inicial de entrenamiento.

  SpamAssassin. SpamAssassin es una herramienta que no necesita entrenamiento y depende principalmente de su configuración.

  Filtro de correo de GMX. Usa el contenido del campo de encabezado del mensaje para determinar si es o no correo basura.

- **Antivirus.**
  KMail usa herramientas externas y especializadas para la detección de virus dentro de correos. La detección de mensajes que contienen virus se realiza definiendo acciones a través de una herramienta dentro de un filtro especial. Otro filtro tiene las reglas para verificar los mensajes con virus detectados y acciones para marcarlos.

- **Seguridad.**
  Las opciones de seguridad relativas a la lectura de mensajes son:

  - Convertir HTML a texto plano.
  - Permitir a los mensajes cargar referencias externas desde Internet.
  - Disponer de las notificaciones de mensajes.
  - No enviar MDNs en respuesta a los mensajes cifrados.
  - Importar automáticamente las claves y los certificados.

  Las opciones de validación S/MIME son:

  - Validar certificados usando CRL.
  - Validar certificados en línea (OCSP).
  - Validar URL del OCSP.
  - Validar firma del OCSP.
  - Ignorar la URL para el servicio de certificados.
  - No comprobar las políticas de certificados.
  - No consultar nunca una CRL.
  - Obtener certificados de los emisores que no se tengan.
  - No realizar ninguna petición HTTP.
  - Ignorar puntos de distribución de certificados de CRL por http.

- Usar el proxy HTTP del sistema.
- Usar proxy para peticiones http.
- No realizar ninguna petición LDAP.
- Ignorar los puntos de distribución de certificados de CRL por LDAP.
- Validar la máquina principal para las peticiones LDAP.

- **Groupware.**
  Hace posible almacenar las entradas de las aplicaciones de Kontact (KOrganizer, KAddressBook y KNotes).

5.8. FTP

FTP (*File Transfer Protocol*) se utiliza para transferir archivos entre dos computadoras, generalmente conectadas vía Internet. Cuando usas FTP, lo que realmente estás usando es un programa llamado *cliente*, que se conecta con otra computadora que mantiene los archivos, llamada *servidor*.

5.8.1. ¿Qué es un FTP anónimo?

Muchos sistemas de computadoras a través de Internet ofrecen archivos por medio de FTP anónimos. Esto significa que puedes acceder a una máquina sin necesidad de tener una cuenta en esa máquina (es decir no tienes que ser un usuario oficial del sistema). Estos servidores FTP anónimos contienen software, archivos para configurar redes, imágenes, canciones y todo tipo de información. Los archivos para las listas de correo electrónico con frecuencia también están almacenadas y todo está disponible a través de un FTP anónimo. Una enorme cantidad de información está almacenada en estas máquinas y está lista para cualquiera que la necesite.

5.8.2. Comandos para FTP

Puedes usar los siguientes comandos para trasladar archivos. Algunos comandos de FTP son los mismos para distintas computadoras, pero otros no. También, algunos sitios de FTP ofrecen distintos comandos, por eso es importante leer los archivos README en el sitio. Usualmente FTP lista los comandos si tecleas 'help' o (?). También los comandos de ayuda de la computadora pueden tener información acerca del FTP. Trata con cualquiera de los siguientes comandos: % man FTP, % man ftpd, % help ftp, % ftp /?, % fftp -?, o % ftp /h .

Algunos comandos disponibles en el sistema se muestran en la tabla 5.4

Tabla 5.4 Algunos comandos disponibles en el sistema

Comando	Descripción
ascii	Cambia a modo ascii.
binary	Cambia a modo binario.
cd	Cambia de directorio en la computadora remota.
dir	Lista los archivos en el directorio actual en la computadora remota.
ls	Es parecido a dir, pero a veces muestra menos información.
get	Copia un archivo de la computadora remota a la tuya.
help	Brinda ayuda sobre el uso de los comandos con el programa ftp.
lcd	Cambia de directorio en tu computadora (la 'l' es por local).
lpwd	Muestra el directorio actual (pwd) en tu computadora.
mget	Copia varios archivos de la computadora remota a la tuya.
pwd	Muestra el directorio actual de la computadora remota.

Un FTP anónimo es un servicio que ofrecen muchas máquinas en Internet. Esto permite que tú ingreses en la máquina con un nombre de usuario anónimo o seas un usuario con nombre ftp.

Los archivos residen en discos y están identificados por un $\langle nombre \rangle$, como NETDISK, y el archivo que está en ese disco se identifica como

NETDISK:[FAQ.INTERNET]FTP.FAQ.

Puedes cambiar de directorio tecleando:

```
% cd netdisk:[faq.internet]
% cd [faq.internet]
% cd faq
```

seguido de

```
% cd internet
```

Para acceso anónimo, teclea

```
% ftp.simtel.net
```

usando como tu login anonymous y como password tu dirección de correo.

Muchos sistemas brindan información acerca de cómo usar el sistema cuando entras a él. Usualmente los archivos están agrupados en grupos de archivos. Los formatos más comunes para preparar archivos para traslado o almacenamiento más económico son tar y zip.

Actividad 5.2

i. *Haz un ftp anónimo a la máquina que sea el servidor.*
ii. *Ya en la máquina servidor busca y baja los archivos* foo.tgz *y* bar.tgz*, que se encuentran dentro del servidor.*
iii. *Guárdalos en un directorio llamado* cosas_foo *y ábrelos.*

5.9. SSH: Secure Shell

$ smart install ssh

Secure Shell (*SSH*) es el reemplazo para rsh, rlogin, rcp y ftp. Encripta todo el tráfico y proporciona varios niveles de autenticación dependiendo de las necesidades. Entre las principales características de Secure Shell se encuentran accesos remotos seguros a otra máquina sobre una red, ejecución de comandos en una máquina remota, mover y copiar archivos de una máquina a otra. Proporciona una autenticación fuerte y comunicaciones seguras sobre canales inseguros. Adicionalmente, SSH proporciona conexiones seguras de X y direccionamiento seguro de conecciones TCP arbitrarias.

Hay dos versiones principales del protocolo Secure Shell: SSH1 y SSH2. La versión vieja SSH1 está siendo reemplazada por SSH2 para alcanzar una flexibilidad mejorada, mejor escalabilidad para organizaciones con cientos de usuarios y mejor seguridad. Actualmente hay funciones que se pueden encontrar en SSH1 que están siendo trasladadas a SSH2.

SSH trabaja cambiando y verificando información y usando llaves públicas y privadas para identificar servidores y usuarios. Después proporciona cifrado para la comunicación subsecuente, también usando criptografía de llaves públicas y privadas. Cada sistema tiene su propio par de llaves públicas y privadas que identifican a otros sistemas. Una llave pública de un sistema cliente puede colocarse en un archivo del servidor para permitir al servidor la autenticación del cliente por medio de un intercambio de llaves.

5.9.1. Accediendo a sistemas remotos

Para establecer una conexión interactiva a un sistema remoto debes usar el comando ssh. Supongamos que eres el usuario Juan.Perez en el cliente beta.fciencias.unam.mx y quieres conectarte a lambda.fciencias.unam.mx; debes de usar el comando ssh y como único parámetro el nombre del sistema remoto.

```
% beta# ssh lambda.fciencias.unam.mx
% Juan.Perez@lambda.fciencias.unam.mx's password:
% Authentication successful.
% lambda#
```

Ahora, si el nombre de tu cuenta en el sistema remoto difiere del que usas en el sistema local (desde donde te estás conectando) debes agregar al parámetro tu nombre en el sistema remoto seguido de una @, como en tu dirección electrónica. Por ejemplo, supón que el nombre en el sistema remoto es Juan.Perez. Así, para conectarse a lambda como el usuario Juan.Perez debemos usar:

```
% beta# ssh Juan.Perez@lambda.fciencias.unam.mx
% Juan.Perez@lambda.fciencias.unam.mx's password:
% Authentication successful.
% lambda#
```

La sintaxis simplificada para conexiones remotas con ssh es:

```
ssh [opciones] [usuario@]sistema_remoto
```

Actividad 5.3 *Inicia una sesión remota en la máquina que tengas a tu lado usando tu mismo nombre de usuario.*

Actividad 5.4 *Inicia una sesión remota en el.servidor.temporal como el usuario guest.*

5.9.2. Copiando archivos entre sistemas

Puedes copiar archivos del sistema local a una máquina remota o viceversa, o aún entre dos sistemas remotos usando el comando scp. Para especificar un archivo en un sistema remoto simplemente hay que agregar como prefijo el nombre del sistema remoto seguido de dos puntos (:).

La sintaxis simplificada para copiar archivos a sistemas remotos de forma segura con scp es:

```
scp [[usuario@]sistema_orig:]archivo [[usuario@]sistema_dest:]archivo_o_dir
```

Si no tecleas el nombre del archivo donde se copiará o no especificas un directorio, se usará el nombre del origen. Una forma sencilla de obtener una copia de un archivo remoto en el directorio actual es usando un punto como destino; por ejemplo, para copiar el archivo hola de la máquina beta al directorio donde estamos actualmente y usando el mismo nombre de usuario, hacemos lo siguiente:

```
% omega# scp @beta:hola .
foo's password:
hola | 154B | 0.0 kB/s | TOC: 00:00:03 | 100
omega# ls
hola
```

Para copiar el archivo hola-Mundo del usuario ivanx en la máquina lambda al usuario kar en la máquina beta usamos:

```
% kappa# scp ivanx@lambda:holaMundo kar@beta:
ivanx@lambda's password:
kar@beta's password:
holaMundo | 34B | 0.0 kB/s | TOC: 00:00:01 | 100
```

Nota que la copia fue en dos máquinas remotas, pues la máquina local es kappa.

Para copiar un directorio completo, tenemos que usar la bandera –r; por ejemplo, si quiero copiar el directorio tarea1, tecleo:

```
% omega# scp -r @beta:tarea1 .
ivanx's password:
```

Actividad 5.5

- *Copia un archivo de la máquina local a una máquina remota como el mismo usuario y después como un usuario diferente.*
- *Copia un archivo de una máquina remota a la máquina local como el mismo usuario y después como un usuario diferente.*
- *Copia un archivo entre dos máquinas remotas.*
- *Copia un directorio entre cualesquiera dos máquinas.*

5.10. PGP: *Pretty Good Privacy*

PGP (*Pretty Good Privacy*) es un sistema de autenticación y cifrado basado en una tecnología llamada *criptografía de llave pública*, escrito por Philip Zimmermann en 1991; permite que nadie, excepto uno mismo y el destinatario o destinatarios a los que vaya dirigido, pueda leer el mensaje, gracias a que los éstos están codificados. Asimismo puede usarse para comprobar la autenticidad del mensaje asegurándonos que en verdad lo ha escrito el remitente.

También se utiliza para enviar archivos a través de correo electrónico codificados en formato ASCII; resulta mucho mejor que otros sistemas como el uuencode, ya que PGP realiza una compresión zip al documento que va a codificar.

5.10.1. Funcionamiento de PGP

El funcionamiento de PGP es muy sencillo: cada usuario tiene dos llaves una *pública* y otra *privada*; la pública es la que distribuyes a la gente y sirve para que ellos puedan enviarte un mensaje codificado que sólo tú, mediante tu llave privada, podrás descifrar; también puede servir para firmar un mensaje poniendo una parte de tu llave privada (irreconocible claro) dentro de una firma. Esto es como un certificado de autenticidad, ya que al recibir el mensaje, PGP comprueba la firma y texto y lo compara con la llave pública que tenemos del remitente, dando un error si se ha cambiado algo en el texto o la firma no corresponde a la persona que nos envía el mensaje.

El cifrado de PGP utiliza la criptografía de llave pública e incluye un sistema que asocia las llaves públicas con cada uno de los usuarios. El cifrado de correo electrónico a través de PGP utiliza algoritmos de cifrado de llaves asimétricas: el remitente utiliza la llave pública del destinatario para cifrar una llave compartida; esta llave compartida se usa para cifrar el mensaje; después el destinatario que recibe el correo cifrado lo desencripta (o descifra) usando su llave privada.

Una estrategia similar se utiliza para detectar cuando un mensaje ha sido alterado o para autenticar que fue enviado realmente por la persona que dice ser el remitente. El remitente utiliza el cifrado de PGP para crear una *firma digital* para el mensaje con algoritmos de RSA o DSA. Para esto, PGP calcula un identificador único(*hash*) para el mensaje y después crea la firma digital de dicho identificador, usando la llave privada del remitente. El destinatario del mensaje calcula el identificador del mensaje recibido y después lo utiliza junto con la llave pública del remitente, dentro del algoritmo de la firma, para crear su propia firma digital del mensaje recibido. Si esta firma coincide con la firma que contenía el mensaje recibido, entonces se puede decir que el mensaje recibido no ha sido corrompido deliberadamente o accidentalmente, puesto que fue firmado correctamente.

No existe un método que pueda romper el cifrado PGP por medios criptográficos o de cómputo. Además, en comparación con otros protocolos de seguridad, como *SSL* que sólo protegen la información momentáneamente mientras están en tránsito a su destino, PGP puede ser utilizada para proteger la información en medios de almacenamiento de larga duración como discos duros. La seguridad en el método de cifrado PGP se basa en el hecho de que los algoritmos utilizados son inquebrantables con las técnicas del criptoanálisis moderno.

5.11. RSS: sistema de noticias

5.11.1. Introducción

RSS es parte de la familia de formatos XML (una especificación genérica para formatos de datos) desarrollado específicamente para sitios que se actualicen con frecuencia y por medio del cual se puede compartir la información y usarla en otros sitios web o programas. A esto se le conoce como *redifusión o sindicación*.

El RSS no es otra cosa que un formato de datos que es utilizado para difundir contenidos de un sitio web. El formato permite distribuir contenido sin necesidad de un navegador. Las iniciales RSS se usa para los siguientes estándares:

- Rich Site Summary (RSS 0.91)
- RDF Site Summary (RSS 0.9 y 1.0)
- Really Simple Syndication (RSS 2.0)

Los programas que leen y presentan fuentes RSS de diferentes procedencias se denominan **agregadores**. Gracias a los agregadores o lectores de *feeds* (programas o sitios que permiten leer fuentes RSS), se pueden obtener resúmenes de todos los sitios que se desee desde tu correo electrónico o por medio de aplicaciones web que funcionan como agregadores. No es necesario abrir el navegador y visitar decenas de paginas web.

5.11.2. Akregator

```
$ smart
install
Akregator
```

Akregator es un lector de RSS como parte de la integración de KDE utilizado principalmente en sistemas UNIX. RSS contiene noticias y actualizaciones de sitios Web y *Web logs*. Los fuentes de RSS pueden ordenarse dentro de categorías, se pueden hacer búsquedas incrementales para los títulos de todas las entradas de la base de datos y Akregator puede ser configurado para que actualice los fuentes en intervalos regulares de tiempo. Provee integración con Konqueror y un navegador interno para leer noticias.

5.1 RSS

El lector de RSS en Gnome es blam, está escrito en C# para el proyecto Mono y GTK#.

Utilizando Akregator

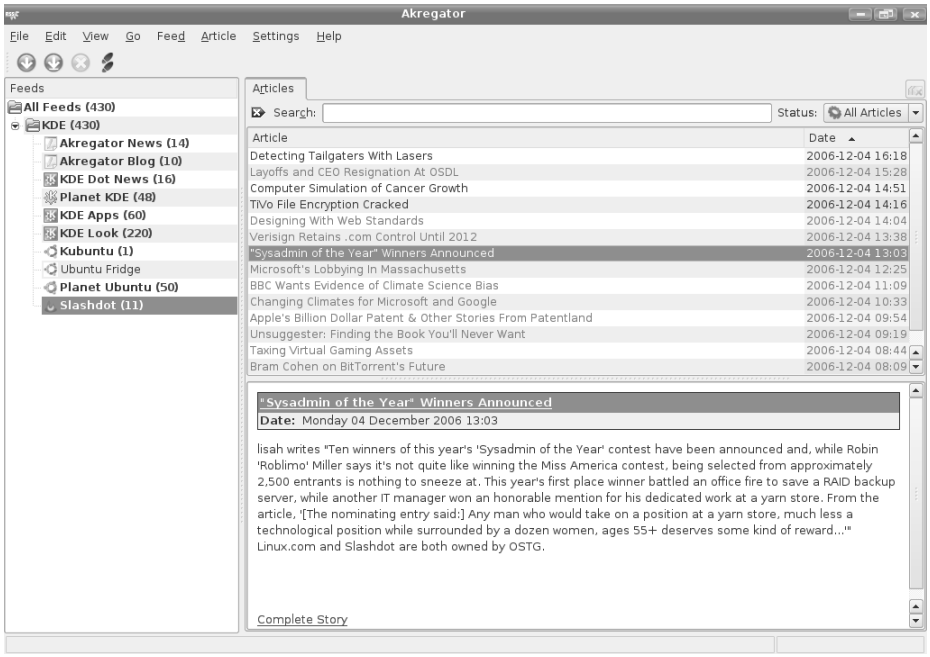
Una vez instalado, Akgregator se encuentra en en el menú principal, dentro de la categoría de Internet; o puedes iniciarlo desde una terminal con el comando akgregator. Akregator

viene pre-instalado con algunos sitios web RSS de KDE, así que puedes bajar los sitios y empezar a leer las últimas noticias de forma fácil y rápida.

Agregando fuentes de noticias

Si ya estás aburrido de leer noticias de KDE y quieres agregar más fuentes de otros sitios, aquí es donde la integración de KDE se da a lucir.

Figura 5.2 Vista principal de Akregator



Muchos sitios web y blog tienen fuentes de RSS. Si entras a un sitio con soporte para fuentes de RSS y estás utilizando el navegador de Konqueror, entonces aparece un icono de RSS con la leyenda *suscribe*, indicando que soporta RSS. Seleccionando este icono y escogiendo la opción *Agregar Fuente en Akregator* podrás leer las noticias de ese sitio web desde Akregator.

Desafortunadamente, no todas las noticias de RSS contienen toda la información en un artículo, lo cual es una lástima porque desperdician mucho de su propósito. Sin embargo, Akregator tiene una forma inteligente de solucionar esto, utilizando un navegador embebido dentro de un tabulador, que permite abrir una liga hacia el sitio web.

Integración

Dentro de las opciones de configuración de Akregator, puedes establecer que el sistema actualice todas las noticias en intervalos regulares de tiempo y que cada vez que ha sido descargada una noticia nueva el sistema te mande una notificación.

Akregator está integrado dentro de Konqueror en el sentido de que permite agregar noticias RSS al sistema de Akregator desde un navegador Konqueror y viceversa, Konqueror está integrado dentro de Akregator en el sentido en que estando dentro de Akregator es posible navegar a través de páginas web.

Si pensabas que no era posible tener mayor integración dentro de aplicaciones KDE, los usuarios de Kontact estarán acostumbrados a tener todas sus aplicaciones de información personal en un solo lugar y Akregator también permite tener un módulo de Kontact de forma que todas tus noticias de RSS están disponibles dentro de tu correo electrónico, calendario y demás entretenimientos.

5.12. Mensajería instantánea

5.12.1. Introducción

La mensajería instantánea (IM) es una manera para comunicarse de manera inmediata con personas a través de Internet. Te permite tener una conversación casi tan natural como en el teléfono o cara a cara, tecleando mensajes en una ventana compartida entre tú y la otra persona.

Además permite ver el estado en que se encuentran las personas, esto es, si están actualmente conectadas al mismo tiempo que tú, de manera que despliega un mensaje IM en la pantalla de la otra persona tan pronto como se envía. O se puede cambiar el estado si prefieres no ser interrumpido, y así otros sabrán que no deben contactarte.

5.12.2. Kopete

```
$ smart
install
kopete
```

Kopete es un cliente de mensajería instantánea, desarrollado por y para el ambiente gráfico de KDE. Te permite comunicarte con amigos utilizando varios servicios de mensajería instantánea. Proviene de la palabra chilena *copete* utilizada para referirse a bebidas alcohólicas.

Kopete soporta los siguientes protocolos:

- AOL Instant Messenger
- Gadu-Gadu

- ICQ
- Internet Relay Chat
- Jabber (XMPP)
- MSN Messenger
- Novell GroupWise
- SMS
- Skype (via Kopete Skype)
- Yahoo! Messenger
- Lotus Sametime

La idea es tener una única herramienta que permita acceder a los diferentes sistemas de mensajería instantánea de una manera fácil de utilizar. El sistema está integrado con otras aplicaciones KDE de manera que te permite acceder a los contactos de la agenda de direcciones y otras utilerías proporcionadas. También provee herramientas para guardar conversaciones, cifrado de mensajes, filtrado de contactos e intercambio de archivos; además es un sistema extensible al que se le pueden agregar fácilmente nuevos accesorios como protocolos de comunicación, edición en LaTeX, traductores de texto, etc.

Kopete introduce la idea de *metacontactos*, que permite combinar las diferentes formas que tienes de contactar a alguien en una sola persona. Otras aplicaciones listan a una misma persona con varias cuentas de mensajería de manera separada (como si se tratara de personas diferentes por tener diferentes cuentas de mensajería); un metacontacto es una persona y los contactos son las diferentes formas de comunicación con ésta. Así, los contactos se reconocen fácilmente mediante iconos pequeños que representan el protocolo de comunicación que puedes utilizar.

Empezando a utilizar Kopete

Probablemente ya cuentes con un servicio de mensajería instantánea, tanto porque ya uses IM, como porque necesitas usar el mismo servicio que tus amigos. Si no es así, te recomendamos utilizar un servicio de mensajería instantánea basado en estándares abiertos, porque éstos están diseñados para ser usados con software libre. Los servicios de mensajería, basados en estándares abiertos que Kopete soporta son Jabber e IRC.

La siguiente sección supone que ya estás registrado con un servicio de IM.

Puedes iniciar Kopete desde el menú de KDE, eligiendo Internet y dentro de este submenú Kopete; desde la barra de herramientas seleccionando el icono correspondiente, como se muestra en la figura 5.3; o en una terminal con el comando kopete. Utilizando el menú de KDE localiza la sección de internet, que mostrará todos los servicios de internet disponibles en tu sistema y selecciona kopete.

Figura 5.3 Kopete en el menú de KDE

5.2 Mensajería Instantánea

El cliente de mensajería instantánea en Gnome es gaim, un mensajero que resulta sencillo y también es multi-protocolo.

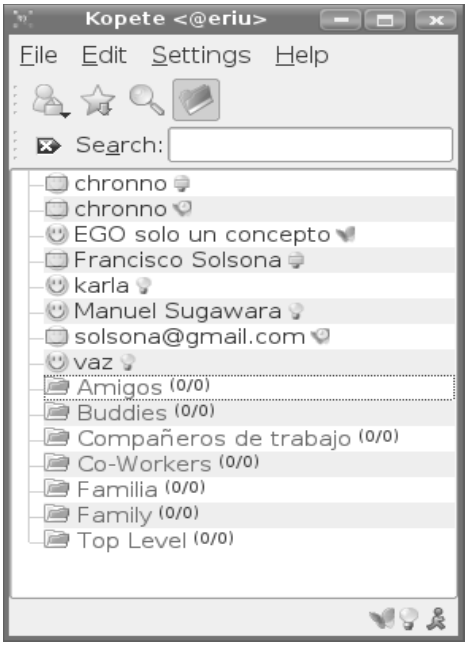
5.12.3. Creando cuentas

Para crear una cuenta, entra al menú Preferencias->Configurar Kopete para mostrar la ventana de configuración. Selecciona el icono Cuentas->Nuevo y escoje el servicio de mensajería que quieras usar.

Una vez especificados los detalles de IM, en la barra de estado de la parte inferior de la Lista de contactos de Kopete aparece un icono representando tu cuenta. Con el botón derecho podrás conectarte a través del menú que aparece. El icono de la barra de estado se animará mientras Kopete se conecta al servicio de IM.

Una vez conectado, se obtendrán tus contactos del servidor y se mostrarán en la Lista de Contactos. Para contactar a alguien, sólo tienes que seleccionar su nombre y aparecerá una ventana de plática, como se ve en la figura 5.4.

Figura 5.4 Vista principal de kopete



5.12.4. Funciones básicas

Las funcionalidades básicas de Kopete son:

Lista de Contactos. La Lista de Contactos es la ventana principal que aparece cuando se inicia Kopete.

Entre las acciones que puedes realizar dentro de la Lista de Contactos se encuentran:

- Establecer el estado que determina cómo estás visible en la red de IM.
- Mostrar todos los contactos, sus estados y los grupos escogidos para ellos.
- Iniciar una conversación con un contacto.
- Enviar mensajes a un contacto.
- Enviar archivos.
- Organizar contactos. Kopete introduce los *metacontactos*, que representan a una persona. Un metacontacto contiene todas las diferentes identidades de IM que tenga esa persona, haciendo fácil identificar si alguien está disponible, sin tener que recordar qué sistema de IM está usando en este momento.
- Agrupar contactos dentro de grupos.

- Agregar contactos.
- Renombrar contactos.
- Eliminar contactos.
- Salir de Kopete. Si solamente se cierra la ventana el programa continuará ejecutándose en la bandeja de sistema de KDE.

Ventana de conversación. La Ventana de conversación es el lugar donde realizas la conversación. Los mensajes aparecen en el orden en que se reciben, con los mensajes anteriores en la parte superior de la vista.
Entre los aspectos que contiene la ventana de conversación se encuentran:

Lista de miembros de conversación. Algunos sistemas de IM permiten conversar en grupo.

Área de entrada. Aquí es donde se pueden teclear mensajes antes de enviarlos.

Barra de estado. La barra de estado contiene mensajes temporales, la notificación de que alguien más está tecleando y el botón Enviar.

Pestañas. Kopete permite llevar múltiples conversaciones en una ventana, colocando cada una en su propia pestaña dentro de la ventana.

5.12.5. Extensiones

Kopete ofrece extensiones que proveen funciones que no son esenciales para la mensajería, pero son útiles, entre las que se encuentran:

- Auto reemplazo de texto, que permite corregir palabras mal escritas o guardar palabras usando abreviaturas.
- Criptografía, que permite usar GnuPG para cifrar conversaciones.
- Respuesta a mensajes particulares, como resaltar texto o reproducir sonidos.
- Guardar conversaciones usando cualquier sistema de IM.
- KopeteTex, que permite mantener conversaciones utilizando el lenguaje de marcas LaTeX.
- Auto-ausente por movimiento.
- Permite a la gente con la que estás conversando en ese momento saber qué música estás escuchando.
- Estadísticas con información sobre el patrón de actividad de tus contactos.
- Efectos de texto.
- Traductor que permite definir un idioma para cada contacto.

5.13. Rdesktop: *Remote Desktop Protocol Client*

Rdesktop es un cliente RDP(*Remote Desktop Protocol*)[3] para la mayoría de los sistemas Unix, como son Linux o FreeBSD. Fue creado por Matthew Chapman. Es una herramienta libre y de código abierto, escrita bajo la licencia de *GNU (General Public License)* que permite al usuario interactuar con terminales Windows NT/2000.

5.13.1. Utilizando *Rdesktop*

Rdesktop inicia indicándole al programa cuál es la computadora Windows a la que se quiere conectar. Puedes iniciar el programa desde una terminal con el comando

```
% rdesktop 192.168.1.10.
```

Esto lo que hace es arrancar el programa e iniciar una sesión en la máquina indicada, es decir, abre una ventana de autenticación de Windows en la máquina con dirección IP 192.168.1.10. El servidor utiliza por defecto el puerto TCP 3389.

5.3 rdesktop en Gnome

La interfaz gráfica dentro de Gnome para utilizar el cliente de Rdesktop es grdesktop y funciona de manera muy similar a krdc.

La manera más común de utilizar Rdesktop es dentro de aplicaciones linux distribuidas sobre clientes delgados, de forma que los clientes delgados arrancan dentro de una red y se conectan a un servidor Windows utilizando Rdesktop.

5.14. Kontact, una suite de productividad en KDE

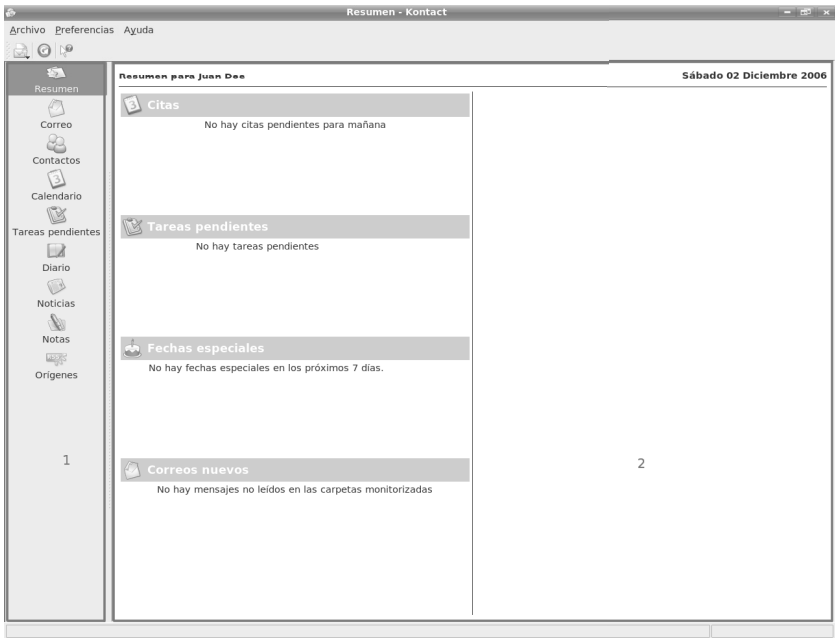
Kontact es un conjunto de programas para manejo de información personal que ofrece KDE. Está integrado a partir de varios componentes, de los cuales la mayoría son aplicaciones por sí mismas, pero aparecen embebidas dentro de kontact. Éste se encarga de integrarlos para tener un solo punto de acceso a ellos y permitir que interactúen entre sí.

[3] El protocolo RDP es un protocolo de aplicación multi-canal que permite a los usuarios conectarse a computadoras con sistemas operativos Windows; además no se necesita tener un ancho de banda muy grande (un modem de 56K es suficiente)

En la figura 5.5 se muestran las distintas partes de Kontact. En la sección 1 se muestran los componentes que agrupa Kontact y se resalta el componente activo. Los componentes que conforman a Kontact son:

- Resumen
- Correo
- Contactos
- Calendario
- Tareas pendientes
- Diario
- Noticias
- Notas
- Orígenes

Figura 5.5 Ventana inicial de Kontact



En la sección 2 se muestra el componente resumen, en el cual se despliegan los elementos relevantes de cada uno de los componentes de Kontact. Esta sección cambia cada vez que seleccionamos un componente distinto.

La cantidad de opciones que presentan la mayoría de las aplicaciones son muchas más de las que se pueden explicar brevemente en este capítulo; por esa razón te mostraremos

cómo configurar cada una de ellas para realizar sólo las operaciones más sencillas.

5.14.1. Resumen

Éste es un componente específico de Kontact, donde se le presenta al usuario un resumen de la información contenida en los demás aspectos de Kontact. Permite que de forma rápida te enteres de los eventos relevantes al día actual, como pueden ser las últimas noticias, fechas importantes (cumpleaños o aniversarios) o tal vez asuntos pendientes.

5.14.2. Correo, KMail

Kontact utiliza a KMail como el lector de correo. La configuración y uso de esta aplicación ya se revisó en la sección 5.7.

5.14.3. Contactos, KAddressBook

Para poder mantener de forma organizada la lista de todas las personas con las que tratas, es necesario que lleves una libreta de direcciones. La aplicación que te permite hacer eso en KDE es KAddressBook. Aquí puedes mantener toda la información de tus contactos y hacer uso de ella con distintas aplicaciones de KDE como KMail o Kopete.

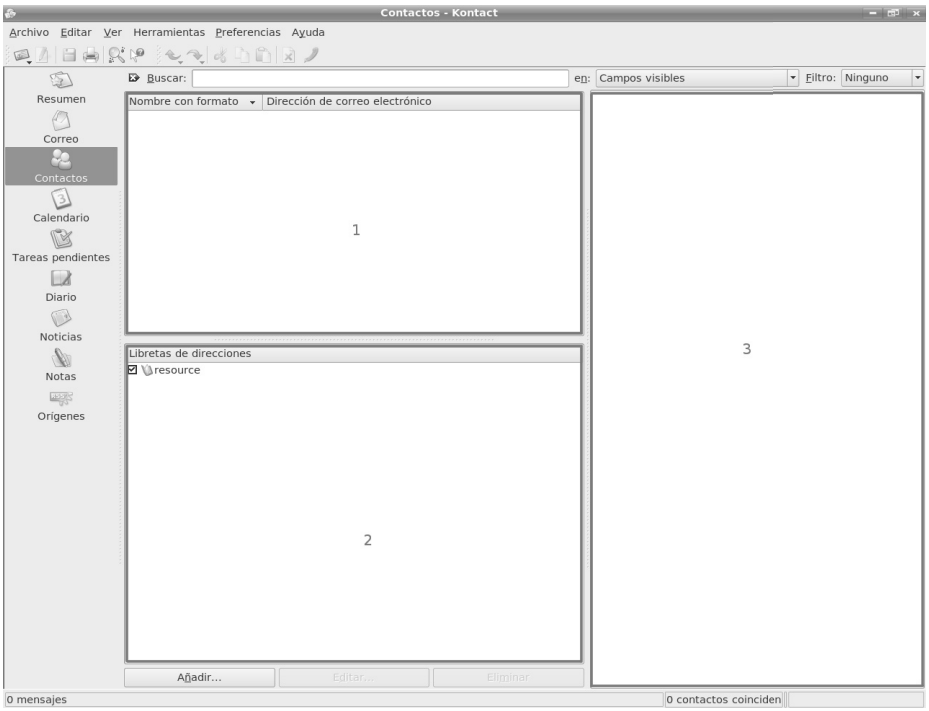
KAddressBook te permite organizar tus contactos en distintas agendas. Esto es útil si quieres tener una agenda con tus contactos personales en tu computadora, pero también quieres tener acceso a una agenda con los contactos de tu trabajo. Si esa agenda la exportan mediante algún servidor de groupware, KAddressBook puede acceder a ella y permitir que el resto de tus aplicaciones KDE hagan uso de esa información. Lo primero que se necesita para empezar a utilizar KAddressBook es tener una agenda, aun cuando te provee de una por omisión. En la figura 5.6 se muestra la ventana inicial de KAddressBook. En la sección 1 se muestran los contactos pertenecientes a las agendas seleccionadas. En la sección 2 se muestran las agendas que están disponibles, y cuales están seleccionadas. En la sección 3 se muestran los datos del contacto seleccionado.

Creando una nueva agenda

Cuando creamos una nueva agenda es necesario indicarle a KAddressBook en donde se encuentra la información correspondiente a nuestros contactos, esta puede estar en la misma computadora en la que se esta arrancando esta aplicación o bien puede estar en una máquina remota. La ventaja de tener una agenda en una máquina remota radica en poder compartir contactos con distintas personas, y poder acceder a esa información desde diferentes computadoras. Existen varias formas para hacer eso, pero en este capítulo veremos

como trabajar con una agenda local. Para crear una nueva agenda local es necesario realizar los siguientes pasos:

Figura 5.6 KAddressBook vacío



i. Presionar el botón *Añadir* que se encuentra en la parte baja de KAddressBook.

ii. Seleccionar *Archivo* (así guardamos toda la agenda en un solo archivo).

iii. Como se muestra en la figura 5.7, tenemos que ingresar un nombre para la agenda y especificar qué formato utilizará. *vCard* es un formato estándar que facilita el intercambio de información, por lo que usualmente ésa es la mejor elección.

iv. Indicar la ruta en la que se creará el archivo.

Una vez hecho eso, tenemos la agenda lista y puedes empezar a agregar contactos a ella.

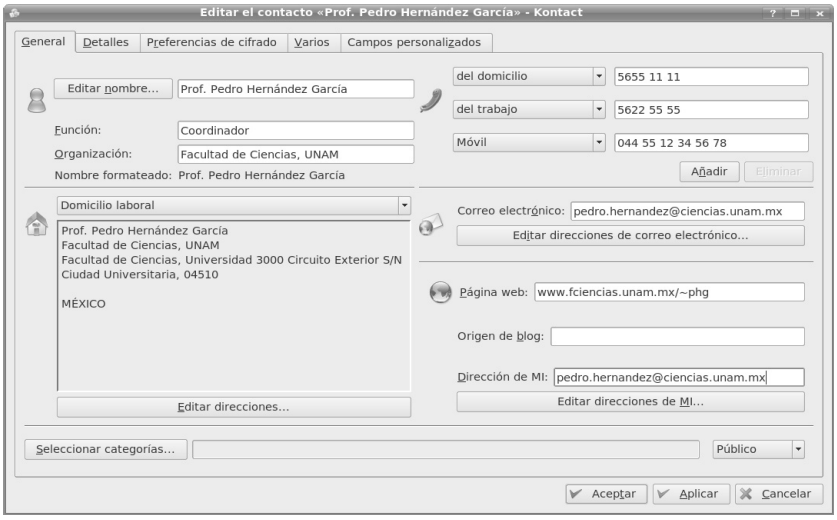
Figura 5.7 Creando una nueva agenda

Agregando un nuevo contacto

Dentro de la pantalla de KAddressBook, en la ventana superior, puedes hacer click derecho y seleccionar *Nuevo Contacto* o utilizar la abreviatura adecuada (por omisión C-N). El siguiente paso es seleccionar en cuál agenda queremos agregar el contacto.

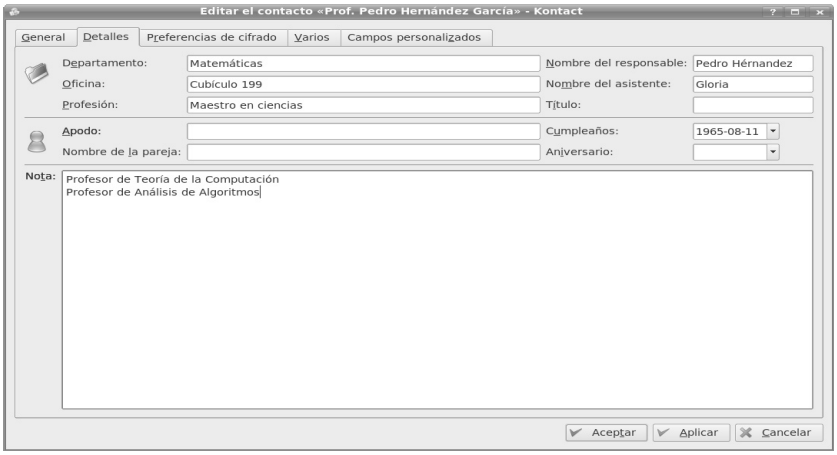
Con eso aparece la ventana que se muestra en la figura 5.8 en la que debes ingresar la información del contacto que vas a agregar. En la pestaña "General.ªgregas información acerca de este contacto. Esto puede ser su nombre, sus teléfonos, direcciones de correo, direcciones de mensajería instantánea y direcciones de inmuebles. Además es posible seleccionar si este contacto es público, privado o confidencial.

Figura 5.8 Creando una nuevo contacto



En la pestaña *Detalles* mostrada en la figura 5.9, puedes especificar información acerca de el lugar de trabajo de la persona, su puesto, su cumpleaños y hasta su apodo.

Figura 5.9 Pestaña de detalles

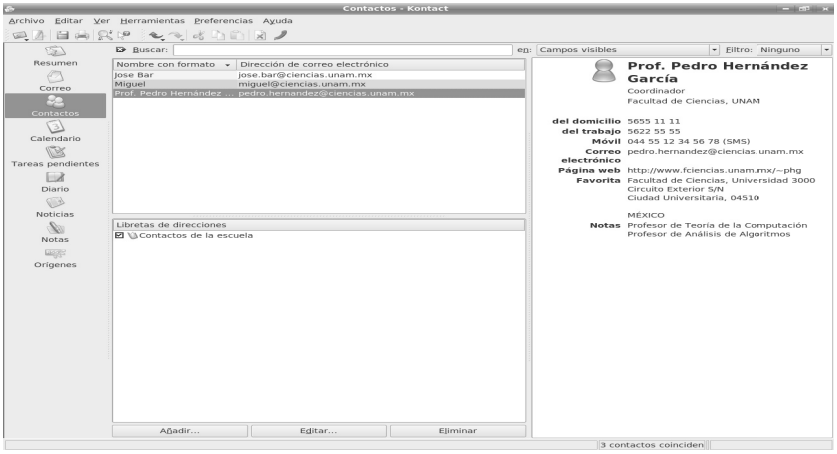


En la pestaña *Varios* puedes especificar la localización geográfica de esa persona, una foto suya y el logotipo de su compañía.

En la pestaña *Preferencias de cifrado* es posible especificar las preferencias de KMail con respecto a este contacto.

Cuando agregas algunos contactos y seleccionas uno de ellos, KAddressBook se ve como en la figura 5.10.

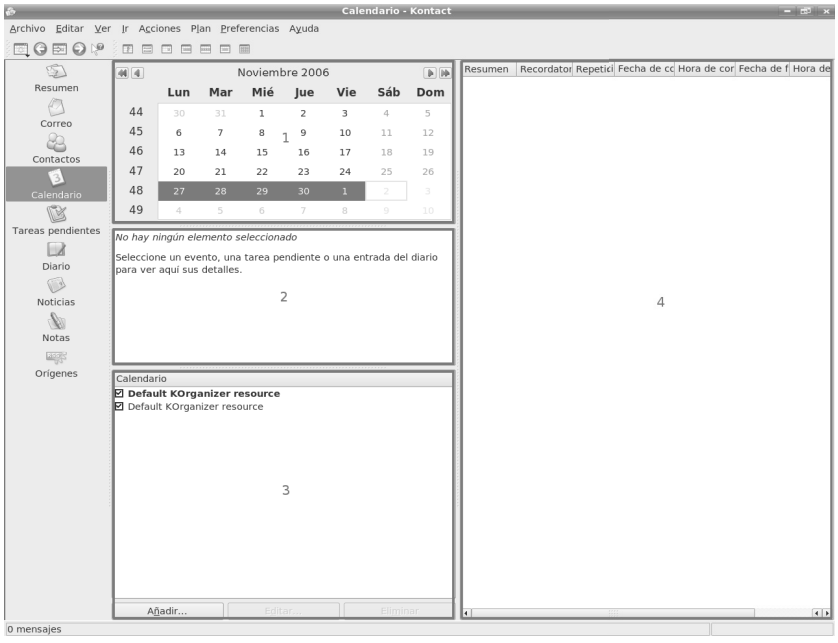
Figura 5.10 KAddressBook con algunos contactos



5.14.4. Calendario y pendientes con KOrganizer

KOrganizer es la aplicación encargada del calendario y los horarios dentro de Kontact. Permite manejar tareas y eventos, establecer alarmas, exportar a web o a otras máquinas, establecer horarios para grupos de personas y muchas otras actividades. Actualmente acepta los dos formatos dominantes para almacenamientos e intercambio de información de calendarios, *vCalendar* e *iCalendar*. En la figura 5.11 se muestra la ventana inicial de KOrganizer, resaltando diferentes secciones. En la sección 1 se muestra un calendario en el cual podemos escoger que día es el que queremos revisar. En la sección 2 se muestran los detalles de un evento que hemos seleccionado. En la sección 3 podemos seleccionar cuáles son los calendarios que deseamos ver. Finalmente en 4 se muestra una vista para poder revisar eventos de distintas formas. Veamos ahora cómo hacer algunas de las tareas más comunes dentro de KOrganizer.

Figura 5.11 KOrganizer vacío



Eventos, tareas pendientes y el diario

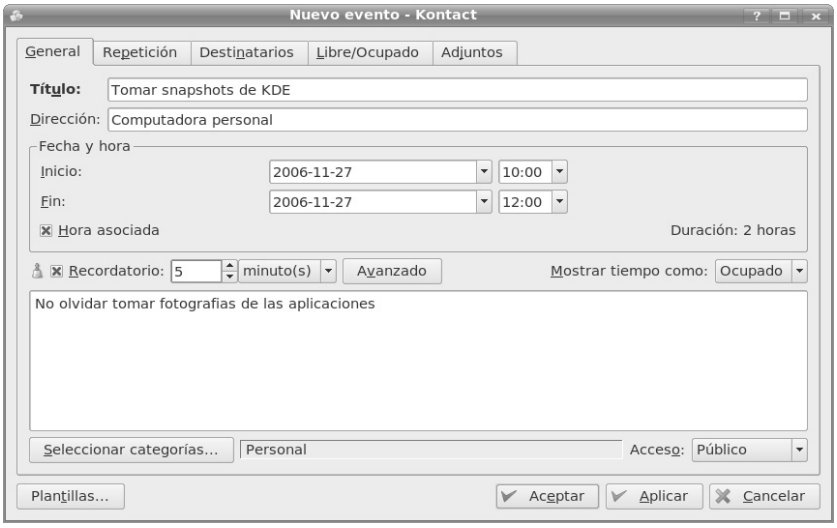
Dentro de Kontact un *evento* es una actividad de la que se quiere llevar un registro. Si solamente quieres recordar algo que tienes que hacer, pero dicha actividad no tiene asociada una fecha o un evento o alguna repetición, entonces es un *pendiente*. Un evento entonces es algo como un cumpleaños, una cita de negocios o alguna celebración. Éstas pueden repetirse una o varias veces y con base a distintas reglas. Además de eventos y tareas pendientes, también puedes asociar un texto a cada uno de los días; para eso utilizas el componente *Diario*. Veamos ahora cómo puedes agregar un evento.

Agregando un evento

Para agregar un evento es necesario seguir los siguientes pasos:

1. Seleccionar *Nuevo evento* dentro del menú *Acciones* o hacer doble click en el día y hora deseados. Aparecerá una ventana como la que se muestra en la figura 5.12.
2. Seleccionar un título y un lugar para el evento.
3. Seleccionar la hora de inicio y final del evento.
4. Opcionalmente, seleccionar la casilla *Recordatorio* para que cuando falte cierto tiempo KOrganizer te recuerde acerca del evento.

Figura 5.12 Creando una nuevo evento



Además de la pestaña *General* tienes otras pestañas que permiten hacer ajustes a los eventos que creas.

Repetición

En esta parte es posible especificar cuáles son las reglas mediante las cuales se debe repetir esta tarea. Aquí existen varias opciones. Lo primero es establecer cada cuánto quieres que se repita el evento. Esto puede ser cada cierto número de días o cada semana, mes o año. El siguiente paso es especificar un rango de fechas entre las cuales debe llevarse a cabo la repetición. Finalmente puedes agregar ciertos días que sean excepciones a la regla. En la figura 5.13 puedes ver un evento que se repetirá cada dos semanas, los días lunes, jueves y sábado. Se repetirá ocho veces, con excepción del sábado 2 de diciembre.

Figura 5.13 Ajustando la repetición de un evento

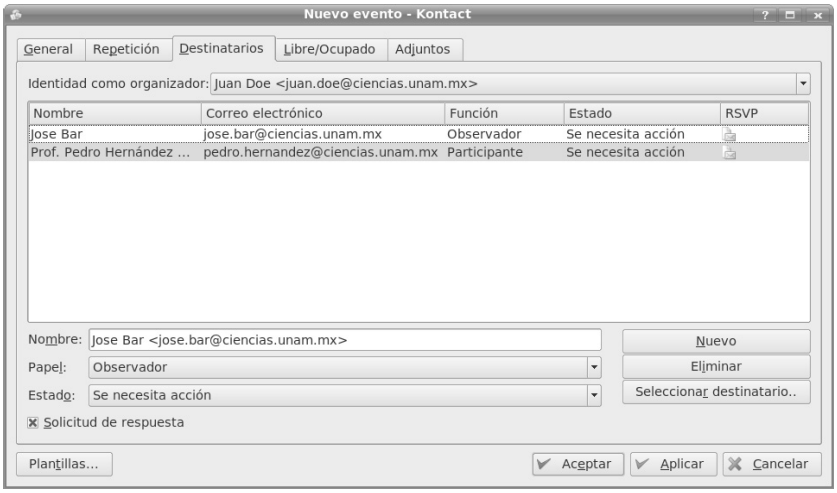
Destinatarios

En esta pestaña podemos crear una lista de personas que van a asistir a dicha reunión. Podemos indicar cuál es el rol que van a desempeñar y cual es su estado actual. Como se muestra en la figura 5.14 se pueden agregar las direcciones directamente con el teclado, o podemos utilizar algunos de los contactos almacenados en KAddressBook. KOrganizer se encargará de notificarles (mediante correo o una aplicación groupware)[4] que han sido

[4]La forma en la que deseas que se notifique a un destinatario puedes seleccionarla en la configuración de KOrganizer

invitados a dicho evento y puede además pedir que confirmen su asistencia al mismo.

Figura 5.14 Agregando participantes en un evento



Libre/Ocupado

Para saber si puedes o no invitar a alguien al evento es necesario saber si están disponibles en esa fecha a esa hora. Si los invitados publican esa información, puedes revisar sus horarios en esta pestaña. En la figura 5.15 sección 1 puedes revisar cada uno de los invitados a la reunión, puesto que aparece una línea por cada uno de los participantes. En la sección 2 se muestra una gráfica en cada línea que indica el horario de un participante. Con base en esta información puedes ajustar el horario de la reunión o si presionas *Zoom hasta ajustar* puedes permitirle a KOrganizer hacerlo por ti.

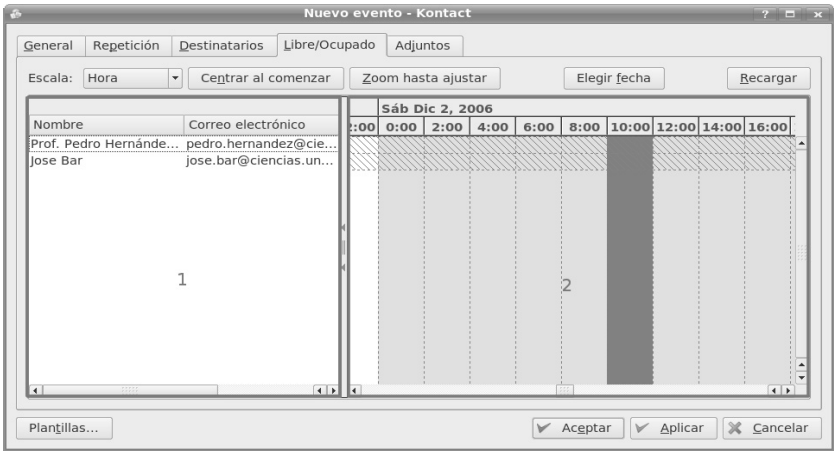
Adjuntos

En esta pestaña se pueden colocar archivos adjuntos a la reunión para que estén disponibles para consulta por parte de los participantes. Es necesario colocar el URI del archivo, para que sólo eso se envíe; **no** se envía el archivo completo.

Vistas

KOrganizer puede presentar la información de diferentes formas, restringiendo qué es lo que se desea ver. La ventana principal está dividida en dos secciones. En la sección de la izquierda se muestran los componentes para navegar las fechas, ver pendientes y ver elementos; en la parte derecha de la ventana se muestra la vista seleccionada. Entre los distintos tipos de vistas que presenta KOrganizer se encuentran los siguientes:

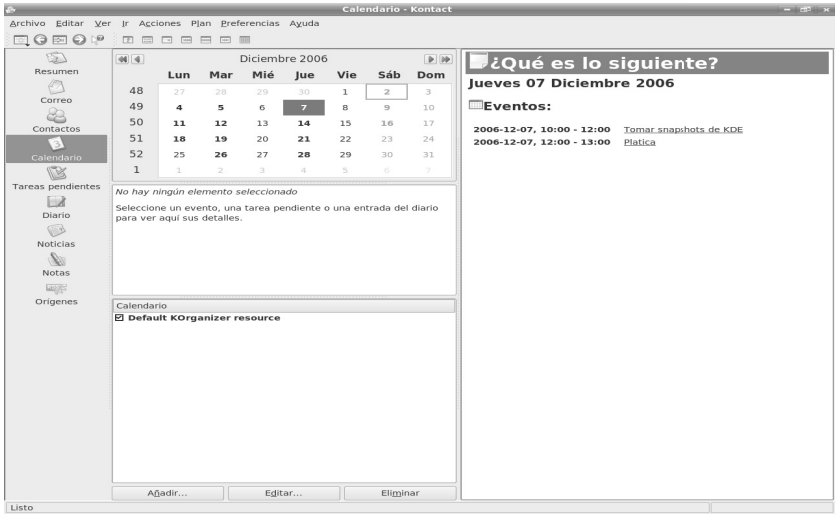
Figura 5.15 Ajustando la hora de la reunión



¿Qué es lo siguiente? Despliega eventos y pendientes de una forma que facilita la lectura.

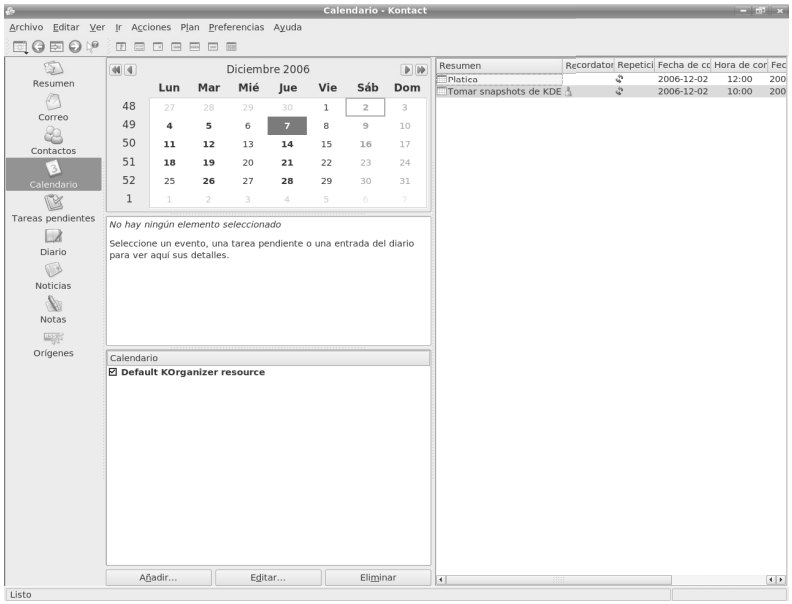
Se muestra en la figura 5.16.

Figura 5.16 Vista *¿Qué es lo siguiente?*



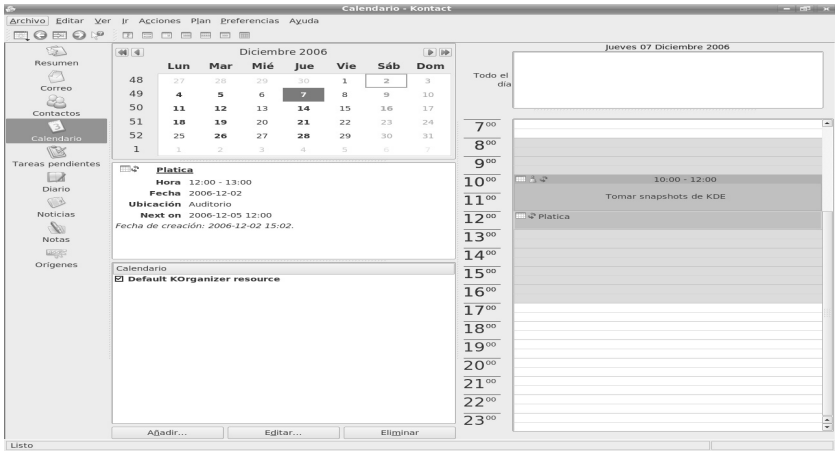
Lista. Muestra todos los pendientes y eventos en una lista. Se muestra en la figura 5.17.

Figura 5.17 Vista *Lista*



Día. Muestra un día, presentando las actividades que hay que realizar sólo durante ese día. Se muestra en la figura 5.18.

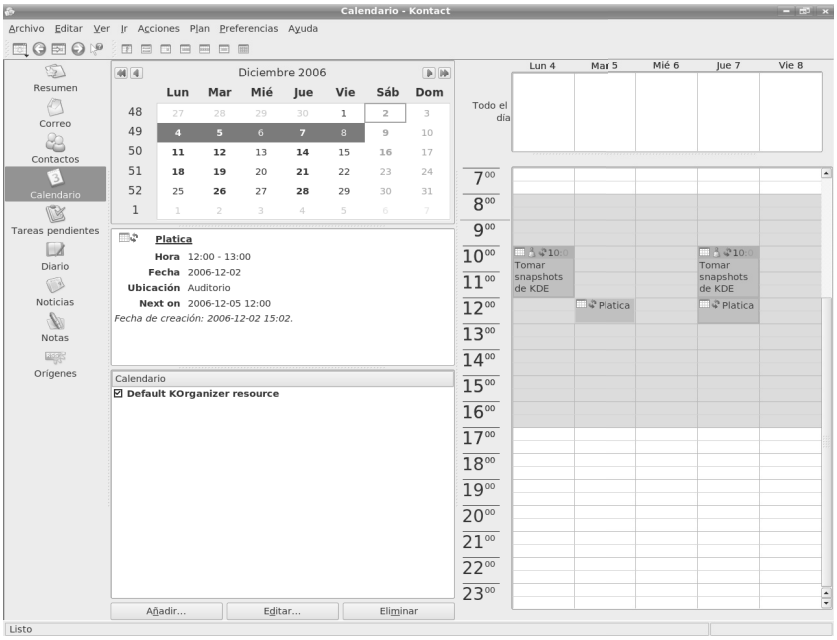
Figura 5.18 Vista *Día*



Semana laboral. Similar a la vista de día, pero muestra una semana de lunes a viernes. Se

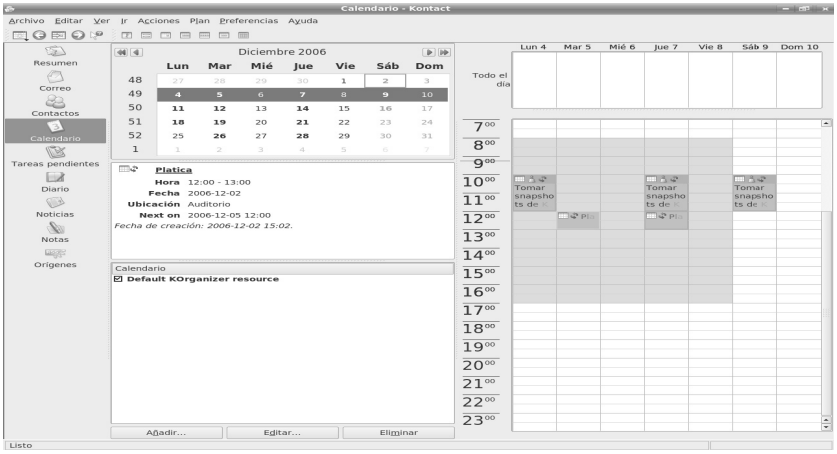
muestra en la figura 5.19.

Figura 5.19 Vista *Semana laboral*



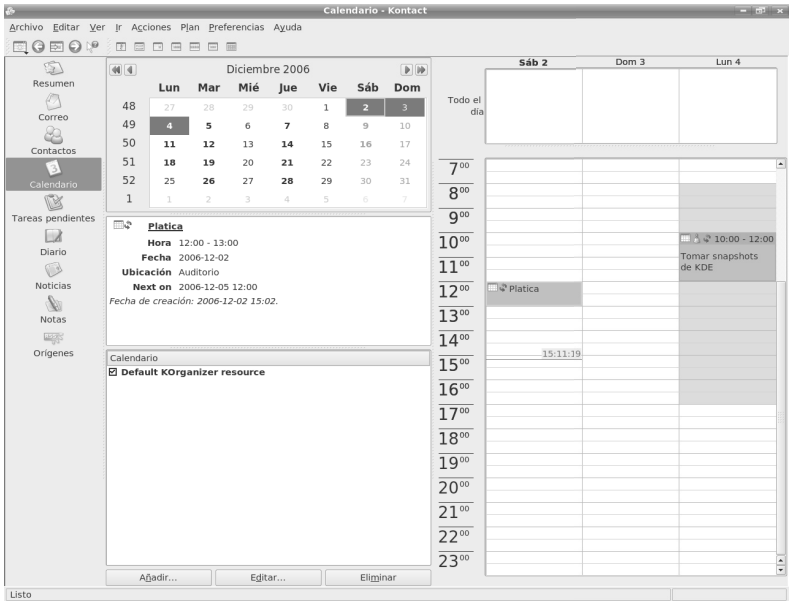
Semana. Igual que semana laboral, pero muestra todos los días de la semana. Se muestra en la figura 5.20.

Figura 5.20 Vista *Semana*



Tres días siguientes. Muestra las actividades de los siguientes días. Por omisión muestra los siguientes tres, pero eso puede modificarse en el elemento *Configurar calendario* dentro del menú *Preferencias*. Se muestra en la figura 5.21.

Figura 5.21 Vista *Tres días siguientes*



Mes. Muestra todas las actividades para el mes actual. Se muestra en la figura 5.22.

Agregando una entrada al diario

Si seleccionas el componente diario, Kontact te presentará la vista correspondiente en la que puedes agregar una nueva entrada o editar alguna ya existente. Esta vista se muestra en la figura 5.23.

5.15. Emacs

En esta sección te mostraremos como puedes realizar la mayoría de las actividades mostradas con anterioridad en este capítulo, como por ejemplo, leer y contestar correo electrónico, manejar tus contactos, etc.

La mayoría de los programas que presentaremos no son parte estándar de Emacs, por lo que debes instalarlos por separado. Muchos de éstos se conocen como subsistemas de Emacs.

Figura 5.22 Vista *Mes*

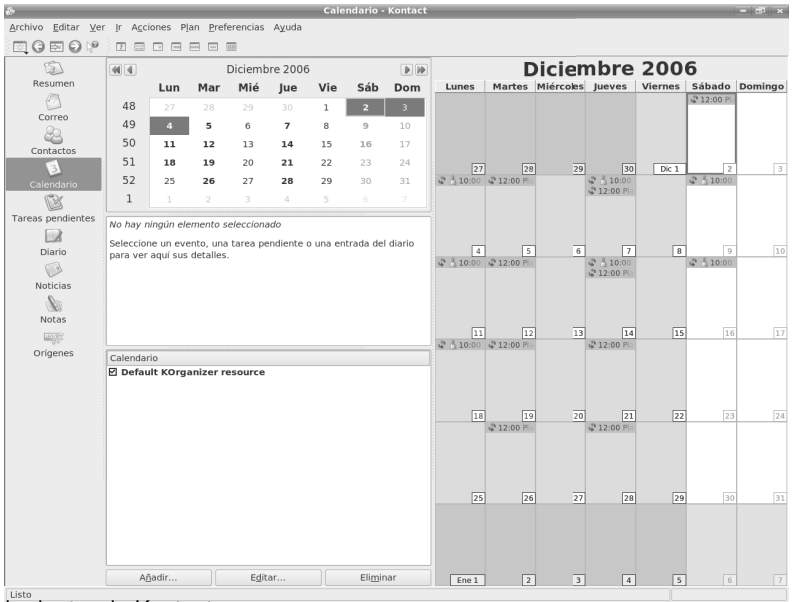
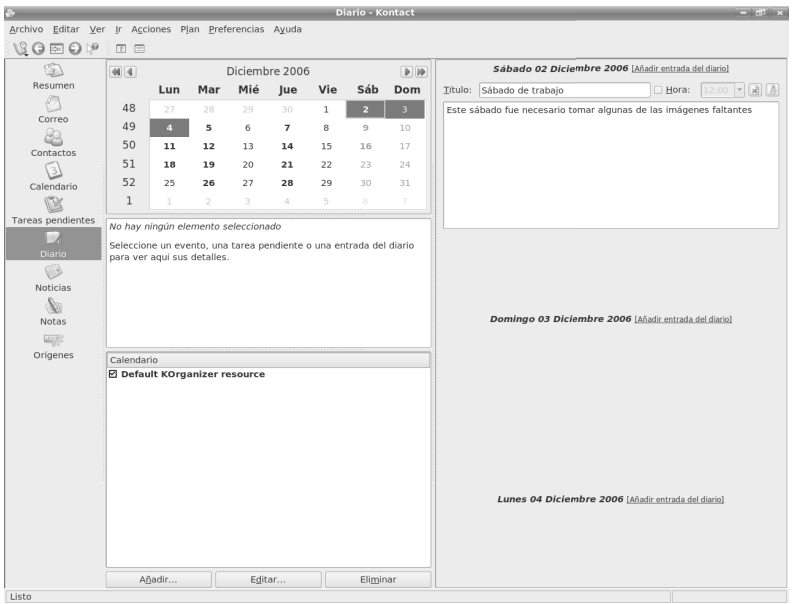


Figura 5.23 El diario dentro de Kontact



5.15.1. VM

5.1 Rmail y Gnus

Además de VM, Emacs cuenta con varios subsistemas más para leer y enviar correo electrónico. Los más importantes son Rmail y Gnus. El primero, Rmail, es un lector de correo muy sencillo, mientras que el segundo, Gnus, nació como un lector de noticias (USENET) al que le agregaron opciones de correo.
Gnus es un sistema tan grande y completo como VM y una excelente alternativa. Desafortunadamente ha caído un poco en desuso, por lo que no lo revisaremos.

$ smart install vm

VM (siglas de *View Mail*, en inglés) es un subsistema de Emacs que te permite manejar tu correo electrónico desde Emacs. Como la mayoría de los lectores de correo, VM te permite leer correos, contestarlos, reenviarlos y otras acciones más. Sin embargo, también tiene algunas funciones avanzadas como creación de resúmenes (*digest*), el reenvío de mensajes y la organización de los mismos de acuerdo a ciertos criterios.

Para iniciar VM basta que teclees el comando M-x vm y aunque esto funcionará y te mostrará la cara de VM, no te lo recomendamos hasta concluir la siguiente sección, donde te enseñaremos a configurar VM para leer tu correo.

Iniciando con VM

VM es probablemente uno de los subsistemas más grandes que cubriremos en este libro – probablemente con la excepción de AUCTEX en la sección 6.12 – tanto así que por la cantidad de variables que puedes modificar, VM utiliza un archivo de configuración distinto del ∼/.emacs, que se llama ∼/.vm.

Después de ejecutar vm, si hay mensajes en tu bandeja de entrada VM selecciona el primero de los mensajes no leídos y te muestra una *vista previa*. Estas vistas previas son la manera de VM de mostrarte parte del mensaje y permitirte decidir si quieres verlo completo o no. Si deseas verlo completo, con Space (vm-scroll-forward) puedes hacerlo. Nota que este comando hace dos cosas distintas, primero te muestra el inicio del mensaje y, si éste es más largo que la ventana actual, ejecuciones subsecuentes del comando (tecla Space) paginan el contenido del mensaje.

Si no deseas ver el mensaje que VM muestra, con n (vm-next-message) puedes cambiarte al siguiente mensaje, si es que hay otro.

Puedes guardar el mensaje que estás viendo con s (vm-save-message) o borrarlo tecleando d (vm-delete-message); sin embargo, el mensaje no se borra inmediatamente, sino que VM pone una marca en él para borrarlo más tarde. Esto es importante, porque

si te arrepientes acerca de borrar el mensaje, puedes seleccionarlo y presionar u (vm-undelete-message) y rescatas el mensaje.

La acción de eliminar para siempre los mensajes borrados se conoce como *expulsión* y se logra tecleando # # # (vm-expunge-folder).

Puedes utilizar h (vm-summarize) para que VM te muestre una ventana con un resumen del contenido de la carpeta actual. En esta lista aparecen los mensajes presentados por número de mensaje, con el autor, fecha, tamaño (en líneas y bytes) y el asunto del mensaje. Además, a la izquierda de cada resumen, aparecen una serie de letras que te indican, entre otros, si el mensaje es nuevo, no leído o marcado para ser borrado.

Una vez que terminas de leer y manejar tu correo, debes guardar la carpeta actual para que la próxima vez que entres a VM, éste sepa cuáles ya fueron leídos, cuáles ya contestaste, etc. La tecla S (vm-save-folder) salva la carpeta. Puedes optar por terminar la visita de esta carpeta con q (vm-quit), que salva la carpeta antes de terminar su revisión.

En cualquier momento durante tu sesión de VM puedes utilizar el comando g (vm-get-new-mail) para revisar si arribaron nuevos mensajes para esta carpeta. Si hay nuevos mensajes, este comando los moverá de la bandeja de entrada (en inglés *spool*) a la carpeta actual.

Tu carpeta de correo base

Por omisión VM intentará asociar tu buzón local con tu carpeta de correo base, por lo que si este comportamiento no es adecuado, y en la mayoría de los casos no lo será, debes indicarle dónde vive tu correo. Por ejemplo, nuestro viejo amigo *Juan Pérez* tiene lo siguiente en su ∼/.vm para indicarle a VM de dónde debe obtener el correo nuevo:

```
1  (setq vm-spool-files
2        '(("~/INBOX"
3           "imap-ssl:galadriel.fciencias.unam.mx:993:
4            inbox:login:Juan.Perez:*"
5           "~/INBOX.CRASH"))
6        )
```

Como puedes apreciar, la variable vm-spool-files recibe una lista de listas. Esto significa varias cosas. Primero, que VM puede manejar varias bandejas de correo electrónico distintas, sin importar si éstas son bandejas locales o remotas. Segundo y menos obvio, que puedes asociar en una misma carpeta correo proveniente de diversas fuentes. La sintaxis para especificar cada una de las fuentes de correo, es decir cada una de las listas internas de esta variable, es utilizar tres cadenas que representen, en este orden, una carpeta, una bandeja de entrada de correo y una carpeta de recuperación.

VM tiene por omisión una carpeta de entrada para correo llamada ∼/INBOX y, como puedes ver en el ejemplo arriba (línea 2), es la carpeta que utilizó Juan para configurar su correo. La carpeta de recuperación (en este caso el valor por omisión es ∼/INBOX.CRASH) la utiliza VM para almacenar inicialmente todos los mensajes extraídos de la bandeja de

entrada de correo; después procesa todos los mensajes almacenados en la carpeta de recuperación y los mueve a la carpeta de entrada.

Gracias a esto último, VM ofrece un excelente soporte para errores, ya que si algo sale mal durante el proceso de tu correo o se interrumpe la ejecución de Emacs por cualquier motivo, tu correo no se pierde y cuando ejecutes VM nuevamente, éste buscará en la bandeja de entrada y la carpeta de recuperación, y procesará los mensajes que encuentre y los moverá a tu carpeta de inicio.

Finalmente, en las líneas 3 y 4 del ejemplo de configuración arriba[5], Juan le indica a VM que para acceder a su correo tiene que utilizar estos parámetros:

Protocolo: El protocolo que soporta el servidor de correo donde Juan tiene su cuenta es IMAP y requiere cifrado con SSL. VM además soporta los protocolos de correo local y POP3.

Servidor: El servidor es galadriel.fciencias.unam.mx.

Puerto: 993, que es el puerto estándar para IMAP sobre SSL. El puerto usual para IMAP es 143, para POP3 el 110 y para POP3 sobre SSL el 995.

Bandeja de entrada: La bandeja de entrada en el servidor es inbox.

Mecanismo de autenticación: El mecanismo para que Juan se autentique con el servidor que utiliza en este caso es login, pero VM también soporta preauth y cram-md5, que son mecanismos comunes en muchos servidores IMAP sobre SSL.

Dirección de correo: Juan además le indica a VM que su cuenta de correo en el servidor tiene el identificador *Juan.Perez*.

Contraseña: Finalmente, por razones de seguridad, Juan le indica a VM que la contraseña para acceder al servidor deberá solicitarla (utilizando el mini-buffer) cada vez que él quiera acceder a su carpeta de inicio.
Una vez que VM recibe la contraseña y logra autenticarse exitosamente con el servidor, almacena la contraseña por el resto de la sesión, por lo que Juan solamente debe teclear su contraseña una vez durante su sesión actual de Emacs.

Selección de mensajes

Para poder leer, borrar o realizar cualquier acción con un mensaje, primero debes seleccionarlo o, en terminología VM, convertirlo en el *mensaje actual*. La mayoría de los comandos de VM surten efecto sobre el mensaje actual.

[5] En realidad estas dos líneas (3 y 4) deben aparecer como una sola, pero la dividimos en dos por razones de legibilidad.

Estos son los principales comandos para seleccionar mensajes en VM:

Tabla 5.5 Emacs: comandos de selección de VM

Enlace	Comando/Descripción
n	(vm-next-message) se mueve hacia adelante en la carpeta actual. Salta mensajes marcados como borrados.
p	(vm-prev-message) se mueve hacia atrás en la carpeta actual. Salta mensajes marcados como borrados.
Enter	(vm-goto-message) te pedirá un número de mensaje, N, y saltará a ese mensaje.
Tab	(vm-goto-message-last-seen) te llevará al último mensaje revisado.
N (P)	(vm-{next|prev}-message-no-skip) selecciona el mensaje siguiente (anterior), sin importar si está marcado como leído o no.
M-n (M-p)	(vm-{next(prev)}-unread-message) se mueve hacia adelante (atrás) hasta encontrar el siguiente (anterior) mensaje no leído.
M-s	(vm-isearch-forward) funciona como la búsqueda incremental de Emacs, excepto que cuando termina la búsqueda VM te muestra el mensaje en el que terminó el punto.

Si deseas que los comandos básicos n y p se salten también los mensajes leídos, debes asignar t a la variable vm-skip-read-messages; y si además deseas que los mensajes en la carpeta VM los recorra como en una lista circular, es decir que al llegar al último y seleccionar *el siguiente* (con n), VM te lleve al primer mensaje, también debes asignar t a la variable vm-circular-folders.

Una vez seleccionado un mensaje VM te lo mostrará. Como ya mencionamos anteriormente, esto es un trabajo de dos pasos: vista previa y mensaje completo.

La vista previa te muestra una pequeña parte del mensaje para que puedas decidir si quieres o no verlo completo. Con Space accedes al mensaje completo y puedes paginarlo.

Soporte en VM para MIME

VM tiene soporte para los dos mecanismos de codificación MIME (*Multipurpose Internet Mail Extensions*) que son *Quoted-Printable* y *BASE64*. En general, MIME asegura que la transmisión de los mensajes se realiza sin errores y define ciertos tipos de contenidos válidos.

Existen variables en VM que te permiten controlar si deseas o no ver ciertos contenidos MIME. Por ejemplo, vm-auto-displayed-mime-content-types, cuyo valor por omisión es ("textimagemultipart") y que incluye algunos de los valores comunes para mensajes de correo codificados con MIME.

Enviando mensajes

Cuando envías mensajes desde VM utilizas el modo mayor Mail de Emacs, más algunas extensiones con los siguientes enlaces:

Tabla 5.6 Emacs: comandos para manipular la composición de correo en VM

Enlace	Comando/Descripción
C-c C-y	(vm-yank-message) copia un mensaje de la carpeta padre al buffer de composición. Lee el número de mensaje deseado del minibuffer. Por omisión, a cada línea del mensaje copiado se le agrega el valor de la variable vm-included-text-prefix, cuyo valor por omisión es " > ".
C-c C-a	(vm-mime-attach-file) anexa un archivo a la composición. Cuando envíes el mensaje, VM insertará el archivo y lo codificará utilizando MIME.
C-c C-m	(vm-mime-attach-message) anexa un mensaje de correo a la composición. Te preguntará el número del mensaje que anexará y cuando envíes tu correo, VM añadirá el mensaje como un digerido MIME.
C-c C-b	(vm-mime-attach-buffer) añade un buffer de Emacs a la composición.
C-c C-e	(vm-mime-encode-composition) codifica el mensaje elaborado utilizando MIME, pero no lo envía. Esto es útil si quieres firmar el mensaje con PGP antes de enviarlo. Después de esto puedes enviar el mensaje con C-c C-c.
C-c C-p	(vm-preview-composition) Te muestra una vista previa del mensaje actual en elaboración. En la vista previa todos los comandos típicos de VM están disponibles y con q continúas la composición.

La forma más simple de iniciar un buffer de composición es con el comando m (vm-mail).

En caso de que el mensaje sea rechazado y regresado por cualquier motivo, puedes re-

enviar el mensaje utilizando M-r (vm-resend-bounced-message). Este mensaje tendrá un nuevo encabezado *Resent-To* donde debes poner la dirección correcta del receptor que fue rechazado.

Los comandos más interesantes de Mail y que son heredados por VM son:

Tabla 5.7 Emacs: comandos de Mail

Enlace	Comando/Descripción
C-c C-s	(mail-send) envía el mensaje y mantiene el buffer de composición seleccionado.
C-c C-c	(mail-send-and-exit) envía el mensaje y selecciona otro buffer. Este es el comando más común para concluir la composición de mensajes y enviarlos.
C-c C-f C-t	(mail-to) mueve el punto al encabezado *To*, creando uno si no existe.
C-c C-f C-s	(mail-subject) mueve el punto al encabezado *Subject*, creando uno si no existe.
C-c C-f C-c	(mail-cc) mueve el punto al encabezado *Cc*, creando uno si no existe.

Iniciar una nueva composición de correo es tan usual como contestar a un mensaje recibido y VM hace muy sencillo lograrlo. Cuando contestas un mensaje desde VM, éste se encarga de llenar los encabezados de destinatario (*To*) y asunto (*Subject*), anexando *Re:* al asunto del mensaje, que es la forma estándar de indicar que es una respuesta.

VM también puede ayudarte a citar material del mensaje al que contestas y lo identifica anexando el valor de la variable vm-included-text-prefix.
Los comandos para contestar son:

Tabla 5.8 Emacs: comandos para contestar correo en VM

Enlace	Comando/Descripción
r	(vm-reply) contesta al autor del mensaje actual.
R	(vm-reply-include-text) contesta al autor del mensaje actual y cita el cuerpo del mensaje.
f	(vm-followup) contesta a todos los receptores del mensaje actual.
F	(vm-followup-include-text) contesta a todos los receptores del mensaje actual y cita el cuerpo del mensaje.

VM ofrece además tres comandos para traspasar o enviar mensajes que recibes a otros

receptores. Éstos son:

Tabla 5.9 Emacs: comandos para adelantar mensajes de VM

Enlace	Comando/Descripción
z	(vm-forward-message) este comando te sitúa en un buffer de composición, similar al comando m, pero el mensaje original aparece en el buffer.
@	(vm-send-digest) funciona como z, excepto que todos los mensajes en el folder actual se concatenan en un digerido y puestos en el cuerpo del buffer de composición.
B	(vm-resend-message) si deseas traspasar un mensaje recibido, pero que no aparezca *encapsulado* (que es lo que hace z) entonces debes utilizar este comando y VM iniciará un buffer de composición con una réplica del mensaje, pero agregará un encabezado *Resent-To* que debes llenar con la dirección deseada.

Marcas de mensajes

La mayoría de los comandos mencionados para enviar, contestar y traspasar mensajes reciben argumentos numéricos prefijos (con C-u N, donde N es un número) y pueden trabajar sobre mensajes *marcados*. VM provee marcas de propósito general que pueden aplicarse a cualquiera o a todos los mensajes dentro de un folder.

Para marcar el mensaje actual teclea M M (vm-mark-message). Si le pasas un argumento numérico N, marcará también los siguientes $N - 1$ mensajes siguientes; con un argumento negativo, los N-1 previos. Un asterisco (*) aparecerá a la derecha de los números de mensaje marcados.

Para quitar la marca a un mensaje puedes utilizar M U (vm-unmark-message). M m marca todos los mensajes en la carpeta actual y M u les quita la marca.

Configuración de correo saliente

VM intentará enviar los mensajes que compones utilizando el método definido por Mail de Emacs, esto es, por el valor de send-mail-function, el cual por omisión intentará utilizar el programa sendmail local.

Dependiendo de tu servicio y configuración de acceso a la red, puede ser que no cuentes con un programa sendmail o con un equipo capaz de enviar directamente correo electrónico. En estos casos es común que tu servicio de correo incluya un servidor de SMTP, que es el protocolo de internet para transmisión de mensajes electrónicos. Si es así, te conviene

utilizar otro programa para enviar mensajes llamado smtpmail y poner algo similar a lo que nuestro amigo Juan puso en su ∼/.vm:

```
(setq vm-mail-header-from "Juan Perez <Juan.Perez@ciencias.unam.mx>"
      send-mail-function 'smtpmail-send-it
      smtpmail-smtp-server "galadriel.fciencias.unam.mx"
      smtpmail-local-domain "ciencias.unam.mx"
      smtpmail-sendto-domain "ciencias.unam.mx"
      )
```

5.15.2. BBDB

$ smart install bbdb

BBDB es una base de datos para manejar contactos en Emacs. BBDB viene de *Insidious Big Brother Database* y ofrece las siguientes características:

- Integración con distintos lectores de correo:
  - Despliega de manera sencilla la entrada correspondiente al remitente del mensaje actual.
  - Permite la creación automática de entradas basado en el contenido del mensaje actual.
  - Permite agregar de manera automática datos a campos de la entrada que corresponde al remitente del mensaje.
- Muestra una lista de registros que casan con una expresión regular.

Para activar BBDB debes poner lo siguiente en tu ∼/.emacs:

```
(require 'bbdb)
(bbdb-initialize)
```

Con lo anterior tendrás una base funcional que te permite realizar búsquedas y manipular registros. Sin embargo, no está asociada con ninguno de los lectores de correo. Para utilizarlo con VM (ver la Sección 5.15.1) y poder enviar mensajes aprovechado tu base de contactos, debes utilizar algo como esto:

```
(require 'bbdb)
(bbdb-initialize 'vm)
```

Además, para utilizarlo desde VM debes agregar lo siguiente a tu ∼/.vm.

```
(bbdb-insinuate-vm)
```

Esta línea agrega enlaces de teclas a VM y lo configura para que le notifique a BBDB cuando se carguen nuevos mensajes. También agrega el enlace M–TAB al modo *Mail* con el cuál puedes completar direcciones que salen de tu base de contactos en BBDB.

Utilizando BBDB

Ahora que hemos instalado y activado BBDB, podemos utilizarla para agregar, revisar o editar nuestros contactos. Por omisión, el archivo donde se guardan los contactos se llama

~/.bbdb y puedes cambiarlo asignando un nuevo valor a la variable bbdb-file.

La base de datos de BBDB consiste de un conjunto de registros y cada registro corresponde a una persona. Cada registro tiene varios campos y estos campos pueden tener distintos tipos; en la tabla 5.10 te mostramos los tipos predefinidos.

Tabla 5.10 Emacs: tipos válidos para campos BBDB

Tipo	Descripción
name	El nombre de la persona, o vacío si corresponde a una organización.
company	El nombre de la organización de la persona.
AKA	Lista de otros nombres o apodos para esta persona.
net	Lista de las direcciones electrónicas de la persona.
adress	Lista de direcciones postales (físicas).
phone	Lista de teléfonos de la persona.
notes	Comentarios.

Cuando ejecutas BBDB con el comando M-x bbdb te pide una expresión regular y te muestra todos los registros que casan con esta expresión. La búsqueda la realiza sobre todos los campos de la base, aunque puedes realizar búsquedas más limitadas utilizando otros comandos, como bbdb-name para buscar únicamente en el campo de nombre o bbdb-net para el campo de direcciones electrónicas.

Una vez que BBDB te muestra algunos registros de tu base, estás en el modo especial *BBDB* y tienes varios comandos a tu disposición para moverte, editar, agregar o borrar registros.

Tabla 5.11 Emacs: comandos de BBDB

Enlace	Comando/Descripción
e	(bbdb-edit-current-field) edita el campo en la línea actual.
;	(bbdb-edit-notes) abreviatura para editar el campo de comentarios.
d C-k	(bbdb-delete-current-field-or-record) borra el campo en la línea actual. Si la línea actual es la primera del registro, BBDB te preguntará si deseas borrar el registro completo.
C-o	(bbdb-insert-new-field) inserta un nuevo campo en el registro actual.
n	(bbdb-next-record) se mueve al siguiente registro.
p	(bbdb-prev-record) se mueve al registro anterior.

Continúa en la siguiente página

Continúa de la página anterior

Enlace	Comando/Descripción
m	(bbdb-send-mail) inicia la composición de un mensaje de correo electrónico dirigido a la persona.
s	(bbdb-save-db) salva la base actual al disco.
r	(bbdb-refile-record) integra el contenido del registro actual con otro registro en la base. Esto es útil porque es usual tener varios registros con información distinta, como, por ejemplo, distintos correos electrónicos de la misma persona.
c	(bbdb-create) crea un nuevo registro en la base de datos con la información que ingresas en el mini-buffer.
q	(bbdb-bury-buffer) esconde el buffer *BBDB*. Nota que este comando no mata el buffer.
a	(bbdb-add-or-remove-mail-alias) añade un nuevo alias de correo para persona.

Para crear tu primer registro, ejecuta M-x bbdb-create (o simplemente c dentro del buffer *BBDB*). Inmediatamente después de ejecutar este comando, BBDB te preguntará, utilizando el mini-buffer para ello, los siguientes datos: *Name*, *Organization*, *Net*, *Address* (dirección postal en varias líneas), *Phone* y finalmente *Notes*. Nota que a excepción de los primeros tres, puedes teclear Enter y BBDB dejará en blanco (y no desplegará) ese campo.

En general, los comandos que acabamos de mencionar en la tabla 5.10, no los utilizarás frecuentemente, porque BBDB ofrece una interfaz más eficaz para manejar tus contactos. Además, con las líneas que agregamos al ∼/.emacs arriba, la mayoría de las modificaciones a tu base de contactos se realizan de manera automática cuando redactas nuevos correos electrónicos o lees correo dirigido a ti.

De hecho, notarás que cuando entras a VM y seleccionas un correo, en el fondo de la pantalla BBDB te mostrará el registro de tu base de contactos con la información del remitente. Si no existe ya un registro para éste, BBDB creará uno por ti. De hecho, los registros de la base cuentan con un nuevo campo llamado mail-alias que te permite asignar nombres especiales, o simplemente alias, a ciertos contactos para enviar mensajes.

Por ejemplo, si agregamos (con el enlace a) el campo mail-alias al registro de Elisa Viso que creamos arriba, podemos teclear *mami* y después, durante la edición de un mensaje electrónico, puedes utilizar el alias en el campo To:.

5.15.3. w3m

En sus inicios, W3 era el navegador más popular para Emacs, pero era sumamente lento. Por otro lado, w3m era un paginador con capacidades para navegar el web, desarrollado por Akinori Ito. Así que se desarrolló esta interfaz para utilizar w3m desde Emacs y el resultado es un navegador eficiente.

$ smart install w3m-el

Para iniciar w3m debes poner la siguiente línea en tu ∼/.emacs:

```
(require 'w3m-load)
```

Al igual que VM, w3m prefiere tener su propio archivo de configuración, en este caso ∼/.emacs-w3m, que es el valor por omisión de la variable w3m-init-file.

Uso básico de w3m

Existen tres formas básicas para iniciar w3m, que se muestran en la tabla 5.12.

Tabla 5.12 Emacs: iniciando w3m

Comando	Descripción
w3m	Arranca emacs-w3m y despliega la página de inicio. Esta página está especificada por la variable w3m-home-page.
w3m-find-file	Te solicitará el nombre de un archivo local en el mini-buffer y lo desplegará en emacs-w3m.
w3m-browse-url	Te solicitará un URL en el mini-buffer y lo desplegará en emacs-w3m. Este comando es usualmente invocado desde otros programas.

Moverte en un buffer de emacs-w3m no es difícil, pues la mayoría de las teclas de Emacs funcionan de la misma forma. Por ejemplo, C-n, C-v y C-s (que usualmente sirven para moverte una línea hacia abajo, una página o buscar una palabra) también son operaciones válidas en los buffers de emacs-w3m.

Para seguir una liga utiliza la tecla Enter. Debes mover el punto a la liga, lo cual es sencillo pues las ligas se destacan del texto normal ya sea que estén subrayadas o tengan un color distinto. Aquí está una lista de enlaces para navegar dentro de un buffer emacs-w3m:

Tabla 5.13 Emacs: siguiendo ligas con w3m

Enlace	Comando/Descripción
Enter	(w3m-view-this-url) despliega la página señalada con el punto. El comportamiento exacto de este comando depende de las propiedades de la liga en cuestión.
g	(w3m-goto-url) te solicitará un URL en el mini-buffer y hará que emacs-w3m despliegue la página.
G	(w3m-goto-url-new-session) te solicitará un URL en el mini-buffer y desplegará la página en una nueva sesión.
c	(w3m-print-current-url)despliega el URL de la página mostrada en el área de eco y la pone en el anillo de la muerte.
u	(w3m-print-this-url) despliega, en el área de eco, la página objetivo del URL marcado con el punto.
R	(w3m-reload-this-page) vuelve a cargar la página. Esto es particularmente útil si estás leyendo las noticias o el diario de una persona, que pudieron ser actualizados desde el momento en que fue cargada la página.

Probablemente la posibilidad de moverte dentro de una página, como lo puedes hacer en cualquier buffer de Emacs, te tiene fascinado. Y, aunque no lo creas, aún hay más y resulta que emacs-w3m agrega una gran cantidad de enlaces para lograr que navegues a velocidades nunca antes vistas.

Las teclas fueron seleccionadas de manera tal que avanzar la página se logra con secuencias muy cortas. Más aún, dado que no puedes escribir en cualquier parte de una página, muchas de las teclas no ejecutan más el comando self-insert-command.

Tabla 5.14 Emacs: navegando con w3m

Enlace	Comando/Descripción
Space	(w3m-scroll-up-or-next-url) recorre la página hacia abajo. Si utilizas VM ya debes estar acostumbrado a este comportamiento, pues así funciona la lectura de mensajes en VM.
delete	(w3m-scroll-down-or-previous-url) recorre la página hacia arriba.
>	(w3m-scroll-left) recorre la página hacia la izquierda.
<	(w3m-scroll-right) recorre la página hacia la derecha.
.	(w3m-shift-left) se mueve (una posición) hacia la izquierda.
,	(w3m-shift-right) se mueve (una posición) hacia la derecha.
M-l	(w3m-horizontal-recenter) recorre la página horizontalmente hasta que la posición actual quede en el centro.

El modo mayor de emacs-w3m, w3m-mode, define además una serie de comandos para moverse entre distintos objetos, tales como ligas, formas o imágenes.

Tabla 5.15 Emacs: puntos de interés en la página *ala* w3m

Enlace	Comando/Descripción
Tab	(w3m-next-anchor) mueve el punto a la siguiente liga (o ancla, en el idioma de emacs-w3m).
M-Tab	(w3m-previous-anchor) mueve el punto al ancla anterior.
]	(w3m-next-form) mueve el punto a la siguiente forma.
[	(w3m-previous-form) mueve el punto a la forma previa.
}	(w3m-next-image) mueve el punto a la siguiente imagen.
{	(w3m-previous-image) mueve el punto a la imagen previa.

Por ejemplo, si estás viendo la página http://www.google.com, (sí, la del afamado buscador), puedes realizar los siguientes pasos:], Enter, (teclea lo que sea que quieras buscar en google) y finalmente C-c C-c para localizar lo deado.

Ahora que ya sabes moverte dentro de una página, resta revisar los mecanismos que ofrece emacs-w3m para movernos entre páginas. Dado que estas páginas son buffers de Emacs, bien podríamos utilizar los mecanismos ya provistos por Emacs, tales como C-c C-b. Sin embargo, esto sería muy lento; por ello contamos con las siguientes formas de hacerlo:

Tabla 5.16 Emacs: movimiento entre páginas con w3m

Enlace	Comando/Descripción
B	(w3m-view-preview-page) se mueve una página atrás en la historia. Con un argumento numérico se mueve hacia atrás N páginas.
N	(w3m-view-previous-page) se mueve hacia adelante una página en la historia. Por supuesto, esto funciona si utilizaste B
H	(w3m-gohome) se mueve a la página de inicio. Puedes cambiar la página de inicio cambiando el valor de la variable w3m-home-page.
^	(w3m-view-parent-page) intentará moverse al directorio padre de la página desplegada. Por ejemplo, intentará moverse a *http://foo/* cuando *http://foo/bar/* sea la página actual. Esto tiene sentido porque es común que busques algo en el sitio que no se encuentra ahí, la página ya no existe u otros problemas similares.
Space	(w3m-scroll-up-or-next-url) cuando el punto se encuentra al final de buffer y no puedes avanzar hacia adelante, entonces la tecla Space te llevará a la siguiente página donde la *siguiente página* es aquélla definida por el encabezado especial *next* en la página actual. Este comando no tiene relación alguna con la historia. Por ejemplo, si continuamos con la búsqueda que realizamos en google arriba, notarás que al llegar al final de la primera página de resultados y teclear Space, te lleva a la segunda página de resultados.
Del	(w3m-scroll-down-or-previous-url) cuando el punto se encuentra al inicio del buffer y no puedes moverte hacia arriba, puedes teclear Del y esto te llevará a la página previa. Al igual que Space, esto funciona si la página contiene el encabezado especial *previous*.

Regresa a tu vida ordinaria

Una vez que inicias este recorrido por los subsistemas de Emacs, como emacs-w3m, es fácil perderse y olvidar que existe un mundo fuera de Emacs, que tienes una vida, un trabajo, responsabilidades. Probablemente iniciaste tu sesión pensando terminar ese programa de tarea, aquella poesía que completará tu obra maestra o tal vez las formas que te permitirán defender mejor a tu cliente en corte. Sin embargo, a la mitad de tu trabajo, tenías la necesidad de buscar información en el web y pudiste hacer eso desde Emacs: increíble.

Sin embargo, es momento de continuar tu trabajo y dejar emacs-w3m, para lo cual tienes dos opciones. Teclea q si piensas regresar a emacs-w3m más tarde o Q si no piensas regresar en un buen rato.

La primera opción, q, ejecuta el comando w3m-close-window que *esconde* la ventana de emacs-w3m y selecciona otro buffer, mientras que la segunda, Q, ejecuta w3m-quit, que salva la historia de navegación al disco y termina la ejecución de emacs-w3m.

Imágenes y más

Por omisión, emacs-w3m no mostrará las imágenes de las páginas; sin embargo, la mayoría de encarnaciones de Emacs modernas soportan mostrar imágenes en buffers. Con el comando T puedes cambiar la opción para mostrar o no imágenes en emacs-w3m.

Si deseas que emacs-w3m despliegue siempre las imágenes, debes asignar t a la variable w3m-default-display-inline-images.

Además de esto, emacs-w3m te permite agrandar o achicar las imágenes, guardarlas al disco o verlas con un visor externo. Aquí están los comandos más relevantes:

Tabla 5.17 Emacs: manejo de imágenes en w3m

Enlace	Comando/Descripción
T	(w3m-toggle-inline-images) cambia entre mostrar o no imágenes en los buffers de emacs-w3m.
I	(w3m-view-image) utiliza un visor externo para ver la imagen actual.
M-i	(w3m-save-image) salva la imagen actual a un archivo.
M-[	(w3m-zoom-out-image) agranda la imagen actual.
M-]	(w3m-zoom-in-image) achica la imagen actual.

Favoritos

Emacs-w3m también soporta listas de ligas favoritas o *bookmarks* y, para no variar, te ofrece varias formas de indicarle qué ligas quieres conservar y cómo quieres referirte a ellas. Así, tenemos las opciones que se listan en la tabla 5.18 de la siguiente página.

Una vez que tienes ligas favoritas almacenadas, puedes acceder a ellas con el comando v (w3m-bookmark-view) desde cualquier buffer de emacs-w3m. Otra posibilidad es visitar el URL especial: about://bookmark/. En esta página verás tus ligas favoritas organizadas por sección.

Emacs-w3m tiene muchas más acarácterísticas que, por cuestiones de espacio, no podemos explicar aquí. Por ejemplo puedes interactuar con tu buscador de web favorito con una sola tecla, S (w3m-search) o puedes acumular varios URLs y abrirlos todos con un solo movimiento. Pero todas estas características las dejaremos como ejercicio al lector.

Tabla 5.18 Emacs: ligas favoritas en w3m

Enlace	Comando/Descripción
a	(w3m-bookmark-add-current-url) te permite integrar la página actual a tu lista de favoritos. Si lo llamas con un argumento prefijo, entonces te pedirá el URL a utilizar. Además, utilizará el mini-buffer para preguntarte si quieres editar el título y el URL.
M-a	(w3m-bookmark-add-this-url) añade el URL marcado con el punto a los favoritos.

5.15.4. ERC

5.2 ZenIRC

ZenIRC es otra alternativa popular para acceder a servidores de IRC. Al igual que ERC, ZenIRC no es parte estándar de Emacs y puedes encontrarlo en la dirección http://www.zenirc.org/.

$ smart install erc

ERC es un cliente de IRC[6] para Emacs. ERC es poderoso, modular y extensible. Por omisión ofrece las siguientes características:

- Control de inundación (detecta y detiene ataques electrónicos).
- Fechas.
- Puede conectarte a ciertos canales de manera automática.
- Convierte en botones (ligas) a los URLs, nombres de usuarios, etc.
- Rompe líneas muy largas para facilitar su lectura.
- Completa nombres de usuario y comandos.
- Mantiene una historia de tus entradas.
- Te informa de actividades en el canal utilizando la línea de modo.

Utilizando ERC

Una vez instalado ERC, que como ya dijimos no es parte estándar de Emacs, debes agregar lo siguiente a tu ∼/.emacs:

[6] IRC son siglas del protocolo de Internet *Internet Relay Chat* (RFC 1459), que especifica una red internacional para conectarse y charlar en tiempo real.

```
(require 'erc)
(require 'erc-spelling)
```

Y para iniciar el programa debes utilizar el comando M-x erc-select, quien te preguntará por el nombre del servidor al que debe conectarse. Una vez en un buffer de ERC tienes los siguientes comandos a tu disposición:

Tabla 5.19 Emacs: comandos de ERC

Enlace	Comando/Descripción
C-a	(erc-bol) te lleva al inicio de la línea o al final del *prompt*.
Enter	(erc-send-current-line) envía la línea actual.
Tab	(erc-complete-word) si estás en un prompt, completa la palabra actual. En cualquier otro caso, se mueve a la siguiente liga o botón.
M-Tab	(ispell-complete-word) completa la palabra actual utilizando ispell.
C-c C-b	(erc-iswitchb) te pregunta por un buffer de ERC al que te quieres cambiar.
C-c C-c	(erc-toggle-interpret-controls) activa/desactiva el intérprete de secuencias como comandos en los mensajes.
C-c C-d	(erc-input-action) de manera interactiva, ingresa una acción de usuario y la envía al IRC.
C-c Tab	(erc-invite-only-mode) convierte el canal en uno *por invitación* (+i).
C-c C-j	(erc-join-channel) te permite ingresar a un canal. Si el punto está al inicio del nombre de un canal, utiliza ese nombre.
C-c C-n	(erc-channel-names) ejecuta */names #channel* en el canal actual.
C-c C-p	(erc-part-from-channel) este comando sirve para salir del canal actual.
C-c C-q	(erc-quit-server) te desconecta del servidor después de preguntarte la razón.
C-c C-t	(erc-set-topic) te pregunta el nuevo tema del canal.
C-c C-u	(erc-kill-input) elimina la línea actual utilizando erc-bol seguido de kill-line.

Módulos

ERC viene con una manera poco estándar de agregar funcionalidad que involucra la utilización de módulos. Hay módulos de muchos tipos y para muy diversas funciones; para activarlos debes asignar los deseados a la variable erc-modules y a continuación ejecutar erc-update-modules.
Aquí está una lista de los módulos disponibles:

Tabla 5.20 Emacs: módulos para extender ERC

Módulo	Descripción
autoaway	Asigna el estado de ausente (*away*) de manera automática.
autojoin	Te registra a ciertos canales de manera automática.
bbdb	Integración con la base de contactos BBDB (ver la sección 5.15.2).
button	Convierte en botones a los URL, nombres y otros textos interesantes.
fill	Rompe líneas muy largas.
log	Guarda el contenido de los buffers en bitácoras.
match	Resalta amigos, tontos y otras palabras clave.
notify	Te avisa cuando cambia el estado en línea de ciertos usuarios.
pcomplete	Completa nombres de usuarios y comandos.
readonly	Convierte las líneas mostradas en sólo de lectura.
replace	Reemplaza texto en mensajes.
ring	Activa la historia de la entrada.
scrolltobottom	Mueve la vista al final del buffer.
smiley	Convierte los emoticones en iconos.
spell	Revisa tu ortografía.
stamp	Añade la hora a tus mensajes.
track	Rastrea toda actividad en el canal utilizando la línea de modo.
truncate	Trunca el tamaño de los buffers a cierto tamaño.
unmorse	Traduce el código morse en los mensajes.

Y listo; ahora puedes iniciarte en el arte de perder el tiempo en línea. Hay muchos servidores de IRC en el mundo; los hay de muchos tipos y con todo tipo de temas para conversar. Nosotros te recomendamos irc.freenode.net.

5.15.5. TRAMP

TRAMP es un paquete para editar archivos remotos en Emacs y son siglas de la frase en inglés *Transparent Remote (file) Access, Multiple Protocol*.

TRAMP se distribuye con versiones recientes de Emacs y, una vez cargado, te permitirá el acceso a un sistema de archivos remoto para editar archivos, control de versiones y dired (ver la sección 3.6.3).

El acceso al sistema remoto se puede lograr con distintos programas como rsh, rlogin, telnet o con cualquiera que tenga un método similar de acceso. TRAMP soporta conexiones utilizando ssh y es el mecanismo recomendado.

Para hacer su trabajo, TRAMP requiere tener acceso al sistema remoto y hacer una copia local para trabajar. La transferencia se puede realizar en una variedad de formas.

Los mecanismos más eficientes para transferir archivos grandes son rcp, scp y rsync. Los otros mecanismos requieren de herramientas adicionales para codificar y transferir los archivos en línea, tales como mimencode o uuencode.
Para activar TRAMP debes agregar lo siguiente a tu ∼/.emacs:

```
(require 'tramp)
```

Utilizando TRAMP

Aunque hasta el momento todo lo que hemos dicho de TRAMP suena complicado, te sorprenderá lo sencillo que es utilizarlo. Imagina que deseas abrir el archivo UNARCHIVO en la máquina remota MAQUINA; entonces necesitas visitar el archivo, sí, con el comando C-x C-f (find-file) y le das la siguiente ruta: /MAQUINA:UNARCHIVO. El método de acceso por omisión es ssh; si deseas cambiarlo debes asignar el nuevo valor a la variable tramp-default-method, por ejemplo con la línea que aparece a continuación:

```
(setq tramp-default-method "scp")
```

Aquí están algunos ejemplos de las rutas para distintos archivos en sitios remotos:

- /moria.fciencias.unam.mx:.emacs, para editar tu ∼/.emacs en tu directorio raíz en la máquina moria.fciencias.unam.mx.
- /moria.fciencias.unam.mx:∼/.emacs, para editar el mismo archivo, ya que ∼ se expande a tu directorio raíz.
- /ada.fciencias.unam.mx:/etc/issue, para editar el archivo /etc/issue en el servidor ada.fciencias.unam.mx.

Lo anterior supone que tu nombre de usuario en la máquina local es idéntico al de la remota. Si éste no es el caso, puedes indicarle a TRAMP el nombre en el servidor remoto en la misma línea: /USUARIO@MAQUINA:MIARCHIVO.

También puedes especificar el método de conexión para abrir cierto archivo, como en: /ssh:juan@galadriel.fciencias.unam.mx:/home/juan/miarchivo.txt, que se conectará utilizando ssh al servidor galadriel.fciencias.unam.mx como el usuario juan.

5.15.6. Diccionario

Además de revisar la ortografía de tus documentos (ver la sección 4.3.17), puedes requerir un diccionario. Aquí te mostramos cóímo integrar y utilizar un cliente para un servicio de diccionarios en línea, conocido como *DICT* o *RFC 2229*. Los servidores de diccionarios, aunque existen para muchos idiomas, manejan exclusivamente inglés.

$ smart install dictionary-el

La extensión que nos ocupa se llama *Dictionary* y ofrece una excelente interfaz, amigable y navegable, para acceder a definiciones de palabras en uno o varios servidores. Sobre todo en computación, ¿cuántas veces te encuentras con palabras para las que desearías tener un diccionario a mano? Con este cliente y Emacs no tienes un diccionario, sino muchos. Para activar este subsistema de Emacs debes agregar lo siguiente a tu ∼/.emacs:

```
(load "dictionary-init")
```

Además te recomendamos agregar también las siguientes dos líneas que agregan enlaces de teclas para dos de los comandos más intersantes de *dictionary*:

```
(global-set-key [(control c) ?s] 'dictionary-search)
(global-set-key [(control c) ?m] 'dictionary-match-words)
```

Utilizando *dictionary*

Puedes ejecutar el comando dictionary o utilizar alguno de los enlaces de tecla que definimos arriba, C-c s o C-c m, para acceder al diccionario.. Todos estos comandos te llevarán a un nuevo buffer, con el modo mayor Dictionary.

Este buffer de dictionary está dividido en dos partes. La primera es la sección de botones que te permiten navegar las definiciones, buscar nuevamente, seleccionar diccionarios o salir del diccionario. La siguiente es la sección de definiciones. Una vez que dictionary localiza una o varias definiciones solicitadas, las muestra en esta sección.

Notarás que en las definiciones aparecen palabras resaltadas con otro color. Éstas son referencias cruzadas y funcionan como hipertexto, es decir, son ligas que puedes seguir y dictionary te llevará a su definición.

La forma más usual para interactuar con dictionary es con el comando dictionary-search que te pide en el mini-buffer una palabra a buscar, pero utiliza como propuesta inicial la palabra más cercana al punto. Por la frecuencia de su uso te recomendamos asociar este comando al enlace C-c s, porque así puedes acceder a él rápidamente.

Dentro del buffer de dictionary tienes a tu disposición los siguientes comandos:

Tabla 5.21 Emacs: comandos de *dictionary*

Enlace	Comando/Descripción
q	(dictionary-close) cierra el buffer de dictionary.
h	(dictionary-help) despliega la ayuda en línea de dictionary.
s	(dictionary-search) te pregunta por una nueva palabra para buscarla en los diccionarios.
d	(dictionary-lookup-definition) busca la definición de la palabra cercana al punto.
n o Tab	(dictionary-next-link) posiciona el punto en la siguiente liga.
p o Shift-Tab	(dictionary-prev-link) posiciona el punto en la liga anterior.
m	(dictionary-match-words) pregunta por un patrón y lista todas las palabras que casan con él.
Enter	Visita la liga.

LaTeX 6

6.1. Introducción

Una de las aplicaciones más importantes de la computadora hoy en día es la preparación de texto, sea éste para notas, una carta, un libro, reporte, artículo, tarea, etc.

Un procesador de texto es un paquete al que le tecleas texto, con algunas "observaciones" y el paquete procede a formatear el texto. Entre las operaciones que deseamos haga con el texto están las siguientes:

1. Armar los renglones de tal manera que quede el texto bien repartido.
2. Armar páginas de tal manera que se vayan numerando y que tengan todas el mismo tamaño.
3. Facilitar el cambio de tipo de letra y proveer distintos tipos de letras como itálicas, negritas, etc..
4. Facilitar la edición de tablas de distintos tipos.
5. Facilitar la construcción de ecuaciones matemáticas.
6. Facilitar la inserción de figuras en el texto.

Muchos de los procesadores de texto pretenden hacer mucho trabajo por ti. Por ejemplo, al escribir una carta proveen maneras directas y fáciles de formar la carta, pidiendo únicamente el destinatario, el remitente, la firma, etc..

Hay fundamentalmente dos tipos de procesadores de texto, aquellos que van mostrando el resultado de la edición en la misma pantalla, conforme se va escribiendo – como es el caso de Word – y aquellos que funcionan con comandos de control y que ejecutan un

programa para poder ver cómo queda el texto ya formateado. LaTeX es de este último tipo. Llamemos a los primeros de tipo *intérprete*, porque van *interpretando* conforme van recibiendo el texto, y a los segundos de tipo *compilador*, porque primero reciben todo el texto y después lo transforman a las páginas correspondientes.

Cada uno de estos tipos de procesadores tiene sus ventajas y sus desventajas. La principal ventaja del intérprete es que es más fácil de usar y detectas inmediatamente cuando algo no es como lo quieres. La principal desventaja es que supone demasiados parámetros respecto a cómo quieres el texto, y toma muchas decisiones que son difíciles de deshacer. La principal ventaja de los compiladores es que te da un manejo más preciso del formato que quieres, permitiendo de esta manera una edición más rápida. La principal desventaja es que requiere un nivel de abstracción mayor, ya que como no estás *viendo*[1] el resultado de lo que haces, y tienes que compilarlo para verlo, trabajas más "a ciegas".
Se eligió LaTeX para trabajar acá por varias razones, entre ellas:

(a) Es un paquete de distribución gratuita, disponible en prácticamente todos los sistemas Unix y que está ampliamente documentado.
(b) La documentación viene con LaTeX, por lo que está accesible.
(c) Cuenta con un gran número de gente trabajando continuamente en el proyecto, por lo que hay muchos paquetes adicionales para casi cualquier cosa que se te pueda ocurrir, como graficación, música, animación, entre otros.
(d) Es el más usado en el medio de Ciencias de la Computación y Matemáticas para la elaboración de trabajos.

En suma, aun cuando no logres ver más que una pequeña parte, la estructura de LaTeX es tal que el resto de los paquetes los podrás aprender a manejar por su cuenta, conforme los vayas requiriendo.

6.2. Componentes de un texto

Muchos se refieren a los procesadores de texto como procesadores de palabra, reconociendo con esto que el componente básico de un texto son las palabras. Sin embargo, es en la forma de agrupar palabras donde realmente se formatea texto. Veamos algunos componentes importantes de un texto:

palabras	oraciones	párrafos
subsubsecciones	subsecciones	secciones
capítulos	partes	documento

A partir de las subsubsecciones puedes poner títulos y subtítulos.

[1] Esto, en realidad, ya no es del todo cierto, pues LaTeX cuenta ya con intérpretes que permiten ir viendo cómo queda el texto conforme lo vas escribiendo.

6.3. Ambientes para formato

Por el tipo de procesador que es LaTeX, debes indicarle en qué tipo de ambiente quieres elaborar el documento. Los ambientes definen los márgenes que quieres (arriba, abajo, derecho, izquierdo), qué componentes se vale que tenga el documento (capítulos, secciones, partes), el tamaño del renglón, la separación entre párrafos, la sangría. LaTeX cuenta, entre otros, con las siguientes clases de documentos:

Tabla 6.1 LaTeX: ambientes

Ambiente	Nombre	Descripción
Reporte	report	Se utiliza para textos menos formales que un libro o un artículo. No tiene capítulos, sino que agrupa a partir de secciones.
Artículo	article	Es, tal vez, el ambiente que más seguido utilizarán para la entrega de tareas, elaboración de pequeños manuales, etc. Permite escribir en dos columnas fácilmente.
Libro	book	Da toda la infraestructura para la publicación de un libro, con partes, capítulos, secciones, etc. Permite preparar las páginas para imprimir algo por los dos lados.
Acetatos	seminar	Supone un tamaño de letra de cuatro veces el normal, con lo que permite armar acetatos.
Cartas	letter	Permite elaborar cartas que acomodan de manera fácil el remitente, la firma, etc.

Cada una de las clases de documento supone un distinto ambiente. El ambiente es lo primero que le debes dar al compilador de LaTeX .
Un archivo fuente de texto para LaTeX consiste de:

- Una especificación de ambiente

```
\documentclass{report}
```

- El preámbulo, donde especificas algunos modificadores para este ambiente, como pueden ser paquetes adicionales que quieras usar.

```
\usepackage[spanish]{babel}
\usepackage{ucs}
\usepackage[utf8x]{inputenc}
\usepackage{amsmath}
```

  También en el preámbulo puedes cambiar de manera explícita algunos de los parámetros de la hoja.

```
\setcounter{chapter}{5}
```

```
\addtolength{voffset}{2cm}
```

O, en general, establecer propiedades de los objetos con que vas a trabajar o incluir definiciones:

```
\input{definiciones}
\includeonly{segundo}
```

- El documento propiamente dicho que empieza con

  ```
  \begin{document}
  ```

  y termina con

  ```
  \end{document}
  ```

  y entre estas dos marcas puede haber texto;

  ```
  Éste es un documento que trae conceptos básicos
  ```

  comandos de LaTeX;

  ```
  \begin{center}
      ...
  \end{center}
  ```

  o, en general, comandos que insertan en ese lugar texto de otros archivos:

  ```
  \include{primero}
  \include{segundo}
  ```

Actividad 6.1 *En esta primera práctica de* LaTeX *escribirás las observaciones en un archivo de texto simple. Para ello, utilizando Emacs, deberás abrir un archivo nuevo donde vayas escribiendo.*

Actividad 6.2 *Carga en tu Emacs el archivo* template1.tex *que se encuentra en el subdirectorio indicado en la página 3.1.*

Actividad 6.3 *Compila el archivo fuente. Para hacerlo, vas a llevar a cabo el siguiente diálogo en la ventana de comandos. Irás anotando línea por línea la interacción. Deberá estar subrayado lo que Emacs escribe y sin subrayar las respuestas.*

C–c C–c

para invocar al compilador de LaTeX.

Save file "/home/usuario/template1.tex"?(y/n)

pregunta si se desea actualizar la copia del archivo en el disco. La respuesta deberá ser sí (y) para que LaTeX *trabaje con la última versión.*

Command: (Default LaTeX)

Invoca al compilador de LaTeX para que trabaje sobre el archivo fuente. Se oprime la tecla Enter. *En este momento, el compilador mandará un mensaje:*

Type teclaC–c teclaC–l to display results of compilation

Si tecleas en este momento C–c C–l, *el compilador de LaTeX irá mostrando, conforme va revisando el archivo, el resultado de la compilación. Al terminar, indicará si hubo o no errores. Si no los hubo, aparecerá un mensaje de que terminó bien:*

LaTeX *succesfully compiled (3) pages*

donde dice, entre paréntesis, el número de hojas que pudo armar con el texto dado. También aparece el mensaje en la otra mitad de la pantalla que dirá que LaTeX terminó o que tuvo errores.

Para ver cómo quedaron las páginas compiladas, vuelve a teclear C–c C–c *y el mensaje que aparecerá es si deseas ver (View) cómo quedó. Si oprimes la tecla* Enter *se abrirá una ventana nueva que mostrará el resultado de la compilación, utilizando para ello el paquete xdvi. El archivo generado tiene el mismo nombre, pero su extensión es dvi.*

Si es que hubo algún error, LaTeX te pedirá que oprimas C–c ` *(aunque el mensaje pide, en realidad, que teclee nada más* C–c *y el acento grave ( `), en algunos casos y que tienen que ver conque tienes Emacs para que ponga acentos para el español, deberás repetir esta última tecla, o bien teclearla en combinación con la tecla* Alt *de la parte derecha del teclado). En seguida, LaTeX te llevará a donde se encuentra el error en el texto.*

6.3.1. Posibles errores

En este primer ejercicio realmente tendrás pocos errores que reportar. Generalmente serán aquéllos ocasionados por errores de dedo. Nóta que LaTeX tiene varios tipos de comandos:

i. Aquellos que están entre \begin{...} y \end{...}, donde lo que va entre llaves se repite. Veamos algunos ejemplos:

```
\begin{document}
   ....
\end{document}
```

```
\begin{compactenum}
\item ....
\item ....
\end{compactenum}

\begin{center}
    ...
\end{center}
```

A estos comandos les llamamos "de ambiente" porque delimitan a un texto para que se les apliquen determinadas reglas.

ii. También tenemos comandos que marcan el inicio y final de un cierto tipo de datos; por ejemplo, carácteres matemáticos. Para esto, cuando quieres que la fórmula vaya insertada entre el texto, puedes usar cualquiera de los dos ambientes que siguen.

```
 $  .... $
\(  .... \)
```

y cuando quieres que aparezca en un párrafo separado, tienes para ello cualquiera de los dos ambientes que siguen.

```
$$  ....   $$
\[  ....   \]
```

iii. Aquellos que marcan lo que sigue, y que aparecen únicamente como comandos de control. A estos les llamamos comandos de estado, porque cambian el estado del ambiente:

Ejemplos:

```
\bf
\large
```

Estos comandos funcionan "hasta nuevo aviso". Puedes delimitar también su rango de acción si colocas entre llaves al comando y a todo a lo que quieres que se aplique:

```
{\large Esta es una prueba}
```

y entonces el comando se referirá únicamente a lo que está en las llaves más internas.

iv. Aquellos comandos que actúan sobre sus parámetros:

```
\section{Primera}
\setcounter{section}{4}
```

v. Y por último, aquellos comandos que simplemente se usan como abreviaturas convenientes o para denotar símbolos especiales. *Siempre* empiezan con \

```
\cdot
\uparrow
```

En el primero, segundo y tercer tipo de comandos es usual que se te olvide cerrar un ambiente, y entonces el compilador reportará que se encontró el final del documento antes de lo esperado, o algún mensaje donde indique que lo que sigue no tiene sentido donde está.

Es frecuente que tengas errores al teclear y LaTeX no reconozca el comando. En esos casos señalará exactamente el lugar donde no está entendiendo.

En el último tipo de comandos, además del primer tipo de error, como pudiera ser el de teclear mal el nombre del comando, puedes también darle menos parámetros de los que espera, o de distintos tipos. Por ejemplo, el comando

```
\setcounter{section}{4}
```

espera una número entero en su segundo parámetro, por lo que si le dieras otra cadena protestaría.

Muchas veces es necesario tener en el documento texto que no quieres que aparezca en la versión final. También pudiera suceder que tengas problemas con un comando particular y quieras ver qué pasa si lo quitas, pero sin quitarlo realmente del archivo. Lo que conviene en esos casos es usar comentarios.

LaTeX tiene dos tipos de comentarios. El primero, y el más sencillo de usar, es el que simplemente comenta el resto de una línea. Para ello se usa el carácter `%` (porciento), y desde el punto en el que aparece hasta el fin de línea (en la pantalla, aunque la línea se haya "doblado") el compilador de LaTeX no va a considerar eso para compilar y no aparecerá en el trabajo final.

La segunda manera es introduciendo el ambiente de comentario mediante

```
\begin{comment}
    ...
\end{comment}
```

y todo lo que quede en este ambiente quedará excluido de la compilación de LaTeX. Para que puedas usar esta forma de comentario, debes incluir el paquete comment en el preámbulo de tu documento:

```
\usepackage{comment}
```

Esta manera de incluir comentarios también puede ser muy útil si estás trabajando con un archivo de texto muy grande y deseas compilar por pedazos.

6.4. Formato general de un documento

LaTeX procesa los espacios en blanco de manera especial. Por ejemplo, entre dos palabras un espacio en blanco es lo mismo que 25 o que un cambio de renglón; similarmente, un renglón en blanco es equivalente a diez. En general, no respeta de manera estricta el formato que trae el documento fuente, pues su trabajo consiste, precisamente, en acomodar

adecuadamente el texto de acuerdo a los comandos que recibe. Distingue una palabra de otra por al menos un espacio en blanco, y un párrafo de otro por al menos un renglón en blanco. La forma como aparece el texto fuente, en ausencia de separaciones de palabras o párrafos, no tiene nada que ver con el formato final, ni en el tamaño del renglón, ni en la manera en que silabea.

LaTeX se encarga de generar espacios entre los párrafos cuando encuentra dos cambios de renglón seguidos (al menos un renglón en blanco). También, cada vez que empieza un capítulo, sección, subsección, etc. genera espacios verticales en blanco.

LaTeX ofrece la posibilidad de manipular directamente los espacios que se dejan entre párrafos o el cambio de renglón. En los comandos que siguen, <num> representa a un entero positivo o negativo y <mdda> representa una unidad de medida, que puede ser pt, in, cm, mm, ex, em o cualquier variable de longitud – ex representa la altura de una x mientras que em representa el ancho de la letra m – . Los comandos con que cuentas para manipular los espacios se muestran en la tabla 6.2.

Tabla 6.2 LaTeX: comandos para modificar espacios

Comando	Descripción
\␣	Indica que se deje incondicionalmente un espacio en blanco; puede ser necesario cuando LaTeX genera algún código y suprime los blancos que lo separan.
~	Produce también un espacio en blanco, pero que el compilador no puede usar para cambiar de línea.
\\	Cambia de renglón y el siguiente renglón empieza en el margen izquierdo.
\\[< num><mdda>] \\∗[< num><mdda>]	Da un cambio de renglón, pero salta, además de al renglón siguiente, tanto como se le indique. Si el número es negativo, retrocede en la hoja. El ∗ a continuación de \\\\ es para forzar a que se dé el espacio; de otra manera, LaTeX decide si aplica el espacio o no.
\vspace{<num><mdda>} \vspace∗{<num><mdda>}	Da un espacio vertical del tamaño especificado. Si el número es negativo, retrocede en la página. Nuevamente, el ∗ es para forzar a LaTeX a que recorra el espacio especificado.
\smallskip \medskip \bigskip	Permiten el desplazamiento vertical del texto en un cuarto del tamaño de la línea base (\\baselineskip), media línea base y una completa. Es muy útil porque son saltos relativos al tamaño de letra que estés usando.

Continúa en la siguiente página

Tabla 6.2 LaTeX: comandos para modificar espacios *Continúa de la página anterior*

Comando	Descripción
\noindent	Hace que un párrafo o renglón empiece en el margen izquierdo de la hoja, en lugar de hacerlo con sangría.
\indent	Obliga a dejar un espacio horizontal igual al de la sangría de párrafos.
\hspace{<num><mdda>} \hspace*{<num><mdda>}	Deja un espacio horizontal dado por el número. Si el número es negativo, retrocede en la línea. LaTeX puede decidir que el espacio no es adecuado en ese punto y no obedecer. Nuevamente, el * fuerza a LaTeX a obedecer la instrucción.

Tanto \vspace como \hspace pudieran ser inhibidos por LaTeX si es que el compilador considera que no se vería bien. Por ejemplo, si el desplazamiento vertical toca a principio de una página, decide no hacerlo.

Todo documento, menos las cartas, deben tener un título, autor(es) y la fecha en que fue impreso. Algunas veces se agrega una nota de agradecimiento. En general, a esto se le llama el título. En el preámbulo (antes de que empiece el documento propiamente dicho) debes colocar los datos para que LaTeX pueda armar el título que se desea. Los comandos a LaTeX son:

```
\title{Ejercicios de Latex}
```

Este es un comando con un parámetro que le indica a LaTeX que el título del documento es "Ejercicios de Latex". Nota que el argumento debe ir entre llaves.

```
\author{El Chapulín Colorado \and Pepe Grillo}
```

Si se trata de más de un autor, los nombres de los autores irán separados por la palabra \and.

```
\date{El día más bello del planeta\\
      \today}
```

En la fecha puedes poner lo que quieras, aunque se espera una fecha. Las dos diagonales inversas \\ sirven para cambiar de renglón, y las puedes utilizar en casi cualquier momento. El comando \today sirve para que LaTeX inserte la fecha de hoy.

```
\thanks{Gracias a los voluntarios de este taller}
```

Sirve para agregar, al pie de la página, algún comentario que se desee. Este comando deberá aparecer una vez iniciado el documento propiamente dicho.

Como ya mencionamos, dependiendo de la clase de documento LaTeX se comportará de manera diferente al compilar el texto fuente (la especificación de \documentclass).

Actividad 6.4 *Agrega encabezado al documento. Ponte tú como autor, junto con tus mejores amigos, el título que a tí más te parezca y algún comentario respecto a la fecha. Agrega*

un párrafo de agradecimiento tan grande como quieras. Recuerda nada más que debe ir después de \begin{document}. Compila y ve el efecto.

Actividad 6.5 *El objetivo ahora es ver el efecto que tienen las distintas clases de documento sobre lo que se imprime. En todas las acciones que siguen, haz el cambio, compila y ve el resultado. Anota cuál es el efecto del cambio. Convierte en comentarios (ya sea con* % *o con* \begin{comment} ... \end{comment} para conseguir que compile bien. Comenta tus observaciones y qué es lo que tuviste que "quitar" para que compilara bien. Cada vez que cambies a un nuevo tipo de documento, quita los comentarios que hayas puesto.

Actividad 6.6 *Cambia el tipo de documento a* article. *Vuelve a compilar y reporta lo que observas, en cuanto al formato de la hoja.*

Actividad 6.7 *Haz lo mismo que en la actividad anterior, pero con la clase* book.

6.4.1. Cartas

Para escribir cartas, todo indica que el formato que le tiene que dar LaTeX es muy distinto al de los documentos que hemos visto hasta ahora. Puedes meter en un mismo documento varias cartas, ya que se espera que el remitente y la firma sean los mismos para todas las cartas incluidas en el documento. Por ello, una vez iniciado el documento, debes especificar estos dos parámetros. Veamos primero el remitente:

Remitente: Se refiere a lo que normalmente aparece en el tope de la carta y que es la dirección de aquél quien firma la carta:

```
\address{Dra. Genoveva Bienvista\\
         Cubículo 003. Edif. de Matemáticas\\
         Facultad de Ciencias, UNAM}
```

Puede consistir de uno o más renglones. Si deseas más de un renglón en el remitente, separas los renglones con \\.

Firma: Para la firma se especifica de manera similar al remitente:

```
\signature{ Dra. Genoveva Bienvista\\
            Profesor Titular B}
```

Una vez que tienes definidos estos parámetros, se procede a escribir cada una de las cartas. Se delimita cada una de éstas por

```
\begin{letter}
  . . .
\end{letter}
```

Dentro de cada carta colocarás los parámetros que se refieren a cada una de ellas. Veamos cómo se hace:

Destinatario: A continuación de \begin{letter} colocarás entre llaves los datos del destinatario:

```
\begin{letter}{Distinguido estudiante de Ciencias
               de la Computación\\
               Primer Ingreso\\
               Generación 2007}
```

Con esto, aparecerá el texto inmediatamente antes de la carta.

Introducción: Llamamos introducción a la frase con la que se saluda o inicia la carta. También se relaciona únicamente con la carta en cuestión:

```
\opening{Muy querido estudiante:}
```

y este renglón aparecerá al inicio de la carta.

Despedida: Esta es la frase que aparecerá antes de que aparezca la firma y "jalará" a la firma a continuación. Aparecerá dondequiera que coloques el comando, por lo que lo deberá colocar inmediatamente antes del fin de la carta (\end{letter}).

```
\closing{Quedo de Uds. atentamente,}
```

Copias: Muchas veces quieres poner al pie de la carta la lista de las personas a las que pretendes entregarle una copia de la carta. Esto se logra con el comando:

```
\cc{Mis mejores amigos\\
    Mis estimados colegas\\
    Primer ingreso en general}
```

También este comando, como el de despedida, aparecerá en donde lo coloques, por lo que lo deberás colocar también al final de la carta y después de la despedida.

Actividad 6.8 *Ahora cambia a la clase* letter y modifica el texto fuente para que parezca, en efecto, una carta (o edita un archivo propio para ello). Observa los resultados.

6.5. Tipos y tamaños de letras

Si revisas estas notas, hechas con LaTeX, verás que existen una gran cantidad de tipos de letras y tamaños que puedes usar dentro de tu documento. Para cambiar el tamaño de las letras cuentas con comandos de estado, por lo que el que uno de ellos aparezca se referirá a todo el ambiente en el que están. Es conveniente, por ello, escribirlos entre llaves:

```
{\large este es .......}
```

Veamos una lista de tamaños de letras con sus nombres en la tabla 6.3.

Tabla 6.3 Tamaños disponibles para letras

Nombre	Descripción del tamaño
tiny	esta es una muestra diminuta
scriptsize	un poco menos pequeña
footnotesize	creciendo un poco
small	esta es una muestra pequeña
normalsize	este es el tamaño normal
large	una muestra agrandada un poco
Large	Agrandamos la muestra un poco más
LARGE	Muy grande
huge	Mucho muy grande
Huge	¡Súper extra grande!
HUGE	¡Súper dúper extra grande!

Como se ve, tienes a tu disposición siete tamaños distintos de letras (aunque en realidad no hay casi diferencia entre huge, Huge y Huge. En cualquier lugar donde puede aparecer texto, éste puede aparecer agrandado o empequeñecido. Sin embargo, cuando se está usando ambiente matemático estos comandos no tendrán efecto alguno dentro del ambiente y pueden causar un mensaje de aviso.

Actividad 6.9 *Vuelve a cargar el archivo* CalcProp.tex *para que vuelvas a tener la versión original.*

Actividad 6.10 *Localiza en el documento fuente aquellos renglones que están precedidos por la palabra* \item. *A cada uno de ellos, ponle uno de los tamaños de letra listados anteriormente, encerrando el comando y el texto correspondiente entre llaves. Compila y ve el resultado.*

Actividad 6.11 *Quita las llaves que delimitan el texto. Compila y ve el resultado.*

Actividad 6.12 *Haz que la primera letra de cada párrafo empiece en cada uno de los tamaños especificados. Compila y ve el resultado.*

Veamos ahora cuáles son los tipos de letra que provee LaTeX (se les denomina *fonts*). Vienen tanto como comandos de cambio de estado como con un argumento, que es el texto que se va a cambiar.

Tabla 6.4 LaTeX: comandos para cambios de tipo de letra

Decl.	Abr.	Comando	Descripción
\mdseries		\textmd {...}	Medium Series, se refiere a tipo normal
\bfseries	\bf	\textbf {...}	**Boldface Series, se refiere a negritas**
\rmfamily	\rm	\textrm {...}	Roman Family, un tipo de letra
\sffamily	\sf	\textsf {...}	Sans Serif Family, letras simples
\ttfamily	\tt	\texttt {...}	`Typewriter Family, letra como de máquina de escribir`
\upshape		\textup {...}	Letras sin inclinación
\itshape	\it	\textit {...}	*Texto en itálicas*
\slshape	\sl	\textsl {...}	*Texto ligeramente inclinado*
\scshape	\sc	\textsc {...}	TEXTO EN MAYÚSCULAS ESCALADAS
\normalfont		\textnormal {...}	Texto normal

Todas las abreviaturas funcionan como cambios de estado, por lo que si deseas que únicamente se apliquen a un pedazo de texto, encerrarás el comando junto con el texto entre llaves, la que abre precediendo al comando.

Puedes utilizar otro tipo de texto, conocido como *enfatizado*, que simplemente implica un cambio de tipo de letra, que depende de la instalación de LaTeX. Su abreviatura, es \em y su comando es \emph{...}. Funcionan de manera similar al resto de los cambios de tipo de letra. Tiene la propiedad de que si está en modo enfatizado y se encuentra un comando, cambia a modo normal y viceversa:

```
{\em Veamos cómo funciona este comando cuando tenemos
{\em uno dentro de otro}}
```

Veamos cómo funciona este comando cuando tenemos uno dentro de otro

El último comando, \textnormal, lo usas para "escapar" a algún tipo que hayas dado para todo un párrafo o una sección:

```
\textbf{Quieres que aparezca en negritas, excepto
\textnormal{este texto}}
```

Quieres que aparezca en negritas, excepto este texto

Actividad 6.13 *Modifica el archivo* CalcProp.ltx *para que las primeras dos palabras de cada párrafo sean con un tipo de letra distinto y un tamaño distinto. Usa todos los tipos listados, aunque repitas.*

Actividad 6.14 *Compila y reporta tus resultados.*

A veces puedes combinar dos o más tipos y tamaños de letras. Por ejemplo, puedes querer tener negritas de itálicas:

Negritas itálicas

Este tipo de combinaciones las consigues utilizando los comandos que corresponden a la declaración, cuyo uso es de comando de estado:

```
{\bfseries\itshape Negritas itálicas}} comparadas
con \textit{Itálicas}
```

que produce

Negritas itálicas comparadas con *itálicas*

Si se utilizan los dos comandos de abreviatura

```
{\it\bf Itálicas negritas} comparadas con \textbf{negritas}
```

únicamente se verificará el último (más interno) cambio de estado que se haya solicitado:

Itálicas negritas comparadas con **negritas**

Lo mismo sucederá con cualquier combinación que se desee de dos tipos de letras: deberás combinar usando una como cambio de estado y la otra como comando con argumento – aunque la eficacia de esto depende del orden – o las dos en modo de declaración.

6.6. Listas de enunciados o párrafos

Un párrafo es un conjunto de oraciones que empiezan y terminan con doble Enter. En el documento fuente puedes observar un renglón en blanco para finalizar cada párrafo. El aspecto en el documento formateado dependerá de los parámetros que tenga LaTeX para compilar ese texto – aunque se pueden modificar, no lo haremos por el momento–.

LaTeX provee mecanismos para presentar listas de párrafos, ya sean numerados, etiquetados o con una marca separándolos entre sí. En general, tienen el siguiente formato:

```
\begin{<tipo-de-lista>}
\item <párrafo>
\item <párrafo>
\item . . .
\end{<tipo-de-lista>}
```

Cada párrafo que desees marcar (numerar) deberá ir precedido del comando \item. Entre un \item y el siguiente, o entre \item y el final de la lista puede haber más de un párrafo.

Los *<tipo de lista>* que vas a manejar por lo pronto son los siguientes:

```
\begin{enumerate}
      <enumeración de párrafos>
 \end{enumerate}

\begin{itemize}
    <listado de párrafos>
\end{itemize}

\begin{description}
     <descripción de párrafos>
\end{description}
```

Cada uno de éstos puede aparecer anidado en el otro, o anidado en sí mismo. Veamos primero las numeraciones.

6.6.1. Numeraciones

Empezaremos por el tema de incisos numerados. Veamos un anidamiento de numeración:

```
\begin{enumerate}
\item Este es el primer párrafo en el primer nivel
  \begin{enumerate}
  \item Queremos ver qué pasa al anidar en el segundo nivel
    \begin{enumerate}
    \item Anidamos una vez más para el tercer nivel
    \item Otro inciso para que se vea qué pasa en el tercer nivel
      \begin{enumerate}
      \item Numeración en el cuarto nivel
      \end{enumerate}
    \end{enumerate}
  \item Otro inciso en el segundo nivel de numeración
  \end{enumerate}
\item Otro inciso en el primer nivel de numeración
\end{enumerate}
```

que produce el texto:

1. Este es el primer párrafo en el primer nivel

   a) Queremos ver qué pasa al anidar en el segundo nivel

      1) Anidamos una vez más para el tercer nivel
      2) Otro inciso para que se vea qué pasa en el tercer nivel

         a′ Numeración en el cuarto nivel

   b) Otro inciso en el segundo nivel de numeración

2. Otro inciso en el primer nivel de numeración

Como puedes observar, cada nivel lleva su propio contador y usa sus propios métodos de numeración. Los valores por omisión, como acabas de ver, son los números arábigos seguidos de un punto para el primer nivel; las letras minúsculas seguidas de un paréntesis para el segundo nivel; nuevamente números arábigos, pero seguidos de un paréntesis para el tercer nivel; y nuevamente letras mayúsculas, pero seguidas de un apóstrofo para el cuarto nivel. LaTeX admite únicamente cuatro niveles de anidamiento para las numeraciones. Las numeraciones también tienen definidas espacio entre los distintos niveles y sangrías para los mismos. Puedes definir qué tipo de numeración utilice. La manera más fácil es incluyendo el paquete para numeraciones en el preámbulo de tu documento fuente, \usepackage{enumerate} y entonces podrás marcar qué tipo de numeración deseas, colocando el que correspondería al primero de la lista:

```
\begin{enumerate}[(a)]
\item Es un paquete de distribución gratuita, disponible
      en prácticamente todos los sistemas Unix, y que está
      ampliamente documentado.
\item La documentación viene con \textnormal{\LaTeX{}},
      por lo que está accesible.
\end{enumerate}
```

Con esta opción, produces la numeración con las etiquetas que deseas:

(a) Es un paquete de distribución gratuita, disponible en prácticamente todos los sistemas Unix, y que está ampliamente documentado.

(b) La documentación viene con LaTeX, por lo que está accesible.

Mientras que

```
\begin{enumerate}[I.]
\item Es un paquete de distribución gratuita, disponible en
      prácticamente todos los sistemas Unix, y que está
      ampliamente documentado.
\item La documentación viene con \textnormal{\LaTeX{}},
      por lo que está accesible.
\end{enumerate}
```

produce

I. Es un paquete de distribución gratuita, disponible en prácticamente todos los sistemas Unix, y que está ampliamente documentado.

II. La documentación viene con LaTeX, por lo que está accesible.

Además del tipo de número a usar, indicas también una cierta manera de editarlo. Por ejemplo, en el caso anterior, cuando usas minúsculas las colocas entre paréntesis, mientras que cuando usas números romanos en mayúscula, al número lo haces seguir por un punto.

Las maneras que tienes de numerar presentan al tipo de letra que va a ir, precedido y/o seguido de algún separador, como `):,-`:

- Arábigos. El valor por omisión para el primer nivel. `[1.]`
- Letras minúsculas. `[a-]`
- Letras mayúsculas. `[A:]`
- Números romanos en minúscula. `[i.]`
- Números romanos en mayúscula. `[I.]`

Si aparece entre los corchetes algo que no sea identificado como el primer número de cierto tipo, LaTeX simplemente repetirá esa cadena frente a cada uno de los párrafos marcados con \item. Si deseas, por ejemplo, agregar un texto antes del número, como Nota 1, lo que debes hacer es encerrar entre llaves lo que no corresponde al número.

```
\begi
\item
\item
\end{
```

Nota 1. Primera nota.

Nota 2. Segunda nota.

Cabe aclarar que la sustitución que hagas para el tipo de numeración afecta únicamente al nivel en el que aparece. Los niveles anidados seguirán la convención por omisión descrita arriba.

6.6.2. Listas marcadas

Cuando simplemente quieres marcar a cada párrafo con algún carácter, como un guión, un punto o algo similar, puedes utilizar las listas de párrafos:

```
\begin{itemize}
\item . . .
\item . . .
 . . .
\end{itemize}
```

También estas listas las puedes anidar entre sí hasta cuatro niveles de profundidad, cambiando el carácter que marca a los distintos párrafos. Observa el siguiente ejemplo, con el código de LaTeX a la izquierda y el resultado a la derecha.

```
\begin{itemize}
\item Primero del primer nivel
  \begin{itemize}
  \item Primero del segundo nivel
    \begin{itemize}
    \item Primero del tercer nivel
      \begin{itemize}
      \item Primero del cuarto nivel
      \item Segundo del cuarto nivel
      \end{itemize}
    \item Segundo del tercer nivel
    \end{itemize}
  \item Segundo del segundo nivel
  \end{itemize}
\item Segundo del primer nivel
\end{itemize}
```

- Primero del primer nivel
  - Primero del segundo nivel
    - Primero del tercer nivel
      - Primero del cuarto nivel
      - Segundo del cuarto nivel
    - Segundo del tercer nivel
  - Segundo del segundo nivel
- Segundo del primer nivel

Al igual que en las numeraciones, no puedes tener más de cuatro anidamientos del mismo tipo de listas.

Puedes modificar el carácter que quieras que use como marca en la lista, colocando el carácter deseado entre corchetes a continuación de \item en el párrafo seleccionado. El código se encuentra a la izquierda y el resultado a la derecha.

```
\begin{itemize}
\\item[-] Primero del primer nivel
  \begin{itemize}
  \item[+] Primero del segundo nivel
    \begin{itemize}
    \item[:] Primero del tercer nivel
      \begin{itemize}
      \item[>] Primero del cuarto nivel
      \item Segundo del cuarto nivel
      \end{itemize}
    \item Segundo del tercer nivel
    \end{itemize}
  \item Segundo del segundo nivel
  \end{itemize}
\item Segundo del primer nivel
\end{itemize}
```

- Primero del primer nivel
  - + Primero del segundo nivel
    - : Primero del tercer nivel
      - > Primero del cuarto nivel
      - ◇ Segundo del cuarto nivel
    - ○ Segundo del tercer nivel
  - • Segundo del segundo nivel
- ■ Segundo del primer nivel

Como puedes observar, a diferencia de las numeraciones, únicamente cambia el párrafo de la lista que hayas marcado. El resto de las marcas se mantiene igual. Hay manera de cambiar la marca para todos los elementos del nivel de anidamiento, de la misma manera que se hizo con las numeraciones.

6.6.3. Descripciones

El formato para las descripciones es:

```
\begin{description}
    . . .
    ⟨párrafos-marcados⟩
    . . .
\end{description}
```

donde cada *párrafo marcado* lleva inmediatamente después de \item, entre corchetes, el título que quieres aparezca para la descripción. Este título aparecerá con otro tipo de letra que el normal, para destacarlo. El resto del párrafo se alineará dejando una pequeña sangría. Sigue un ejemplo en el siguiente código.

```
\begin{description}
\item[enumerate:] Cada nivel se va numerando manteniendo un
  contador para tal efecto.
  \begin{description}
  \item[segundo nivel] Es una prueba para ver si tiene sentido
    meter otro nivel más en descripciones.
    \begin{description}
    \item[tercer nivel] Otra pruebita más del asunto
      \begin{description}
      \item[cuarto nivel] Y aún hay más
        \begin{description}
        \item[quinto nivel] A ver si en este tipo de listas
          sí se vale
        \end{description}
      \end{description}
    \end{description}
  \end{description}
\item[itemize:] Cada párrafo se marca con un determinado carácter,
  dependiendo del nivel de anidamiento.
\item[description:] Sirve para poner explicaciones o definiciones.
\end{description}
```

que produce el siguiente texto:

enumerate: Cada nivel se va numerando manteniendo un contador para tal efecto.

- **segundo nivel** Es una prueba para ver si tiene sentido meter otro nivel más en descripciones.
  - **tercer nivel** Otra pruebita más del asunto
    - **cuarto nivel** Y aún hay más
      - **quinto nivel** A ver si en este tipo de listas sí se vale

itemize: Cada párrafo se marca con un determinado carácter, dependiendo del nivel de anidamiento.

description: Sirve para poner explicaciones o definiciones.

Es importante que notes que en el caso de las descripciones puedes anidar tantas veces como lo desees.

Vamos a crear un archivo nuevo para escribir algunas preguntas y respuestas respecto a las listas de párrafos que te acabamos de presentar. Para ello:

Actividad 6.15 *Crea un archivo con clase de documento* article *y que use los mismos paquetes que usa* CalcProp.tex.

Actividad 6.16 *Ponle como título tu nombre y el número de actividad.*

Actividad 6.17 *A continuación, escribe en el archivo el nombre de cada paquete que usas y una breve descripción de para qué lo usas (los nombres de los paquetes te deben dar una idea).*

Ahora que ya tienes más comandos que teclear, veremos algunas tareas que Emacs te facilita:

1. Aparear bien los ambientes en

   ```
   \begin{...}
     ...
   \end{...}+
   ```

   Esto se consigue tecleando C-c C-e y en la línea de comando de Emacs aparecerá

   ```
   Environment type (default ...)
   ```

   Generalmente, el valor por omisión es el del último ambiente elegido o itemize, si es la primera vez en la sesión que lo solicitas. Si tecleas Enter, procederá a crear la pareja \begin{..} \end{..}, dejando el cursor entre ellos para que procedas a escribir. Si el ambiente que elegiste es alguna lista, coloca el primer \item. Si la lista elegida es description, pregunta por la primera etiqueta.
   Este comando de Emacs va guardando los ambientes que le vas solicitando. Puedes recorrer la lista de comandos usados con la flecha vertical. Cuando es la primera vez que vas a usar un ambiente dado, simplemente tecleas el nombre del ambiente (por ejemplo, enumerate, description) para que lo meta a la lista de ambientes "disponibles".
2. Cerrar algún ambiente que esté abierto: C-c]. Con este comando, Emacs coloca el \end{..} del \begin{..} más cercano que no haya sido cerrado.

Actividad 6.18 *Haz una lista de invitados a una boda, donde separes los invitados de los padres del novio, los padres de la novia, los invitados del novio y los invitados de la novia. Esta lista deberá aparecer en el archivo que acabas de crear. La lista deberá facilitar sacar el total de invitados a la boda.*

6.6.4. Listas más compactas

Como puedes observar de los ejemplos que mostramos, muchas veces el espacio entre los incisos en cada tipo de lista es excesivo. LaTeX cuenta con un paquete de LaTeX que te permite acortar estos espacios, el paquete paralist. Este paquete, además de acortar los espacios verticales, asume las funciones del paquete enumerate permitiendo modificar las etiquetas de las distintas listas. Para poder hacer esto último, pasas como opciones al cargar el paquete newenum y newitem de la siguiente manera:

```
\usepackage[newenum,newitem]{paralist}
```

Si usas este paquete, es conveniente ya no cargar el paquete enumerate. Proporciona los siguientes ambientes:

```
\begin{compactenum}
  ...
\end{compactenum}

\begin{compactitem}
    ...
\end{compactitem}

\begin{compactdesc}
    ...
\end{compactdesc}
```

Estos tres ambientes resultan muy cómodos para compactar un poco el texto que se está produciendo. En este libro es más común que usemos estos ambientes que los originales de LaTeX. paralist tiene además listas "corridas" en un único renglón, pero no las revisaremos en este momento.

6.7. Tablas

En general, LaTeX provee comandos para definir tablas de los tipos más variados. Veremos acá únicamente las más sencillas, aunque no por eso poco poderosas.

Una tabla es un conjunto de renglones alineados por columnas. Por ello, para que LaTeX vaya acomodando las columnas le debes dar el número de columnas y el formato de de cada una de ellas. El formato general de una tabla de texto es el siguiente:

```
\begin{tabular}[<pos>]{<cols>}
   <renglones>
\end{tabular}
```

Es necesario aclarar que una tabla puede aparecer en cualquier posición donde aparezca una palabra. Por ello, es conveniente en la inmensa mayoría de los casos preceder y suceder a la tabla con un renglón en blanco.

El argumento que va entre corchetes, *<pos>* se refiere al caso en que desees que la tabla aparezca a continuación de una palabra, si la queremos alineada "desde arriba" ([t]), desde abajo ([b]) o centrada [c], que es el valor por omisión. Este parámetro no tendrá sentido si se deja, como acabamos de mencionar, un renglón en blanco antes y después de la tabla.

El argumento *<cols>* se refiere al formato que deberán tener las columnas de la tabla, y que será aplicado a cada uno de los renglones. Debes especificar el formato para cada una de las columnas que deseas tener en la tabla. También especificas si se debe colocar

algo entre las columnas. De esto, *<cols>* corresponde a una lista de especificadores, que se refieren a la sucesión de columnas e intercolumnas:

Tabla 6.5 Modificadores para la definición de columnas

Especificación	Descripción
l	Especifica que el contenido de esta columna se colocará pegado a la izquierda, y ocupará tanto espacio como requiera para quedar contenido en un renglón.
r	Similar al anterior, pero pegado a la derecha.
c	Similar al anterior, pero centrado.
p{*<ancho>*}	Da un ancho fijo para la columna. LaTeX acomodará el texto como si fuera un párrafo, ocupando tantos renglones como sea necesario.
m{*<ancho>*}	Igual que p, excepto que si hay columnas que dan un número distinto de líneas para el mismo renglón, centra verticalmente el contenido de las columna.
Para estas dos opciones, el *{<ancho>}* deberá estar dado en: in pulgadas cm centímetros mm milímetros pt puntos em ancho de carácter Algún nombre de longitud	
|	Una línea vertical que separa columnas.
||	Dos líneas verticales separando las columnas.

En los primeros tres casos, el tamaño final de la columna será el máximo de todos los renglones de la tabla. Si es que va a haber mucho texto en cada columna, te recomendamos usar la cuarta o quinta descripción, para que el renglón se "doble" y se conforme un párrafo.

Los renglones se componen de material que quieres en cada columna. Cada vez que deseas cambiar de columna, debes insertar un carácter `&` y cada vez que quieras cambiar de renglón deberás agregar al final del renglón \\.

Puedes colocar líneas entre cada dos renglones con el comando \hline. Cuando no es el primero de los renglones, debe venir a continuación de un cambio de renglón. Veamos un ejemplo sencillo a continuación.

```
\begin{tabular}[t]{cccc}
   letras y símbolos\\
   palabras & oraciones
   & párrafos  & subsecciones\\
   secciones  & capítulos
   & partes  & documento
\end{tabular}
```

Se tiene una tabla con cuatro columnas y dos renglones y se usa & para *separar* las columnas entre sí. Se utilizan \\ para *separar* los renglones. La tabla queda de la siguiente manera:

letras
pa
se

Es importante que notes que lo que corresponde a un renglón formado no forzosamente tiene que aparecer en el mismo renglón en el código. Sólo en el caso en que la especificación de la columna sea p o m, el introducir un renglón en blanco será interpretado como cambio de párrafo, que lo realiza dentro del mismo renglón de la tabla y en la misma columna.

Actividad 6.19 *Crea un archivo* Tablas.tex *y copia el preámbulo de* CalcProp.tex*. Copia la definición de la tabla anterior, compila y ve el resultado.*

Actividad 6.20 *Modifica la tabla anterior para que quede de la siguiente manera:*

letras y símbolos	palabras	oraciones
párrafos	subsecciones	secciones
capítulos	partes	documento

don-

de la primera columna está justificada a la izquierda, la segunda centrada y la tercera a la derecha. La tabla tiene tres columnas en lugar de cuatro.

6.7.1. Multicolumnas

En la tabla anterior, el texto letras y símbolos ocupa mucho más espacio que el resto de los textos en esa columna. Por ello, se toma la decisión de que este texto ocupe dos columnas. Si lo separas "a pie" queda mal distribuido, pues interviene el espacio entre columnas:

letra
palab
seccio

Para que se centre en dos columnas puedes utilizar el comando

```
\multicolumn{<núm>}{<cols>}{<texto>}
```

donde {<núm>} es el número de columnas de la tabla original que va a ocupar; {#<cols>} es la manera de acomodar esa columna, con sus posibles carácteres entre columnas; y {<texto>} es lo que queremos que aparezca en esa nueva columna. Un \multicolumn puede aparecer en cualquier lugar donde pueda aparecer el contenido de una columna, al inicio de un renglón o inmediatamente a continuación de un &. Debes tener cuidado de que el número de columnas que se coloca en {<núm>} no exceda el número de columnas que quedan disponibles en el renglón, a partir de la columna donde se está colocando el comando. La tabla queda como sigue:

```
\begin{tabular}[t]{cccc}
   \multicolumn{2}{c}{letras y símbolos}\\
    palabras
    & oraciones  & párrafos  & subsecciones\\
    secciones  & capítulos  & partes  & documento
\end{tabular}
```

y se forma como sigue:

letras y símbolos			
palabras	oraciones	párrafos	subsecciones
secciones	capítulos	partes	documento

Puedes usar el comando de \multicolumn también para modificar exclusivamente el contenido de una columna en un determinado renglón. Por ejemplo, si deseas que para esa columna en ese renglón en lugar de estar centrado esté justificado a la derecha, o que no tenga separadores entre columnas. Puedes hacer esto para lograr, sin juntar columnas, casi el mismo efecto que lograste uniendo dos columnas:

```
\begin{tabular}[t]{cccc}
  \multicolumn{1}{r}{letras y}&
  \multicolumn{1}{l}{símbolos}\\
  palabras
  & oraciones  & párrafos  & subsecciones\\
  secciones  & capítulos  & partes  & documento
\end{tabular}
```

El texto formado queda como se muestra a continuación. Nota que, ahora sí, desapareció el espacio de más entre "y" y "símbolos".

letras y	símbolos		
palabras	oraciones	párrafos	subsecciones
secciones	capítulos	partes	documento

También puedes aprovechar las multicolumnas para poner títulos a la tabla o pies de página:

Componentes de un documento de LaTeX

letras y símbolos			
palabras	oraciones	párrafos	subsecciones
secciones	capítulos	partes	documento

Se puede numerar a partir de subsecciones

En general puedes agregarle modificadores a las columnas para que el texto *de esa columna* tenga un tamaño o tipo particular. Por ejemplo, para el título de la tabla agregas al principio del texto el modificador \bf y para el último renglón cambias el tamaño del carácter a \footnotesize. Estos cambios se aplican únicamente a esa columna y en ese renglón. Asimismo, puedes saltar líneas en blanco entre renglones, simplemente agregando cambios de línea (\\) al final del renglón.

Actividad 6.21 *Repite la tabla que construiste y modifícala para agregarle el título y el último renglón.*

6.7.2. Separación de renglones y columnas

Como mencionamos al decir qué puede ir en {<cols>}, dijimos que puedes tener una o dos líneas verticales que separan a las columnas. También puedes tener líneas horizontales que separen a los renglones. Veamos la siguiente tabla que incluye separadores de columna y de renglón.

Ambiente	Nombre	Descripción
Reporte	report	Se utiliza para cosas menos formales que un libro o un artículo. No tiene capítulos, sino que agrupa a partir de secciones.
Artículo	article	Es, tal vez, el ambiente que más seguido utilizarás para la entrega de tareas, elaboración de pequeños manuales, etc. Permite escribir en dos columnas fácilmente.

que fue producida por el siguiente código:

```
\begin{tabular}[t]{|l|l|p{8cm}|}
   \hline
   Ambiente& Nombre& Descripción\\
   \hline
   Reporte&report&Se utiliza para cosas menos formales
                que un libro o un artículo. No
                tiene capítulos, sino que agrupa
                a partir de secciones.\\
   \hline
   Artículo&article&Es, tal vez, el ambiente que más seguido
                utilizarás para la entrega
                de tareas, elaboración de pequeños
                manuales, etc. Permite escribir en dos
                columnas fácilmente.\\
   \hline
\end{tabular}
```

Observa que antes de cada \hline, excepto el primero, debe haber un cambio de renglón. En cambio, después del \hline no es necesario un cambio de renglón. Como ya vimos, puedes modificar el tipo y/o tamaño de cualquiera de los contenidos de una columna en un determinado renglón. Por ejemplo, se acostumbra que los títulos vayan en negritas. También deseas líneas dobles horizontales entre los títulos y el contenido de la tabla. Juguemos un poco con eso. Para no ocupar tanto espacio nada más mostraremos el renglón o renglones que estemos modificando. Lo primero que haremos es separar a los encabezados de columnas con líneas dobles y ponerlos en negritas, de la siguiente manera:

```
\hline\hline
\bf Ambiente& \Large\textbf{Nombre}&\sc Descripción\\
\hline\hline
```

Integrados estos cambios, la tabla se ve como sigue:

Ambiente	Nombre	DESCRIPCIÓN
Reporte	report	Se utiliza para cosas menos formales que un libro o un artículo. No tiene capítulos, sino que agrupa a partir de secciones.
Artículo	article	Es, tal vez, el ambiente que más seguido utilizarán para la entrega de tareas, elaboración de pequeños manuales, etc. Permite escribir en dos columnas fácilmente.

Supongamos que le agregas dos renglones más, pero que por el momento no tienes decidido qué escribir en la descripción:

```
Acetatos&seminar \\\ hline
Cartas&letter \\\ hline
```

Y veamos cómo se ve la tabla:

Ambiente	Nombre	DESCRIPCIÓN
Reporte	report	Se utiliza para cosas menos formales que un libro o un artículo. No tiene capítulos, sino que agrupa a partir de secciones.
Artículo	article	Es, tal vez, el ambiente que más seguido utilizarás para la entrega de tareas, elaboración de pequeños manuales, etc. Te permite escribir en dos columnas fácilmente.
Acetatos	seminar	
Cartas	letter	

Nota que falta la línea vertical al final. Esto es porque no especificaste nada para la tercera columna. Cuando tienes líneas verticales para separar columnas, debes "completar" todas las columnas en cada renglón. Los renglones agregados debieron tener la siguiente forma:

```
Acetatos &seminar &\\\ hline
Cartas & letter &\\\ hline
```

Veamos cómo queda la tabla, que se muestra a continuación.

Ambiente	Nombre	DESCRIPCIÓN
Reporte	report	Se utiliza para cosas menos formales que un libro o un artículo. No tiene capítulos, sino que agrupa a partir de secciones.
Artículo	article	Es, tal vez, el ambiente que más seguido utilizarás para la entrega de tareas, elaboración de pequeños manuales, etc. Te permite escribir en dos columnas fácilmente.
Acetatos	seminar	
Cartas	letter	

Supongamos ahora que quieres agregarle a la tabla un título, que quede dentro de la tabla en el primer renglón. Para eso puedes utilizar nuevamente el comando \multicolumn. Debes tener cuidado si es que las columnas originales tienen separación por columnas, porque debes especificar explícitamente las nuevas separaciones. Quieres el título centrado, pero de ancho mayor que el resto de los renglones. Para lograrlo pones un renglón en blanco antes y después. Veamos cómo funciona en la tabla que estamos usando como ejemplo:

```
\multicolumn{3}{|c|}{}\\
\multicolumn{3}{|c|}{\Large\textbf{Clases de documentos
    en \textnormal{\LaTeX{}} }}\\
\multicolumn{3}{|c|}{}\\\hline
```

Clases de documentos en LaTeX		
Ambiente	**Nombre**	DESCRIPCIÓN
Reporte	report	Se utiliza para cosas menos formales que un libro o un artículo. No tiene capítulos, sino que agrupa a partir de secciones.
Artículo	article	Es, tal vez, el ambiente que más seguido utilizarán para la entrega de tareas, elaboración de pequeños manuales, etc. Permite escribir en dos columnas fácilmente.

De los ejemplos anteriores, es claro que cuando colocas carácteres entre las columnas, dejar renglones en blanco no es trivial.

Al igual que cuando no tienes líneas entre las columnas, puedes utilizar \multicolumn para quitar las líneas de una determinada columna en un cierto renglón. La manera como se asocian las separaciones de columnas son:

– La primera columna tiene asociadas la línea a su izquierda y a su derecha.

– A partir de la segunda columna, cada columna debe redefinir su línea derecha.

Actividad 6.22 *Codifica la siguiente tabla, que es una nueva versión de la tabla anterior. Ten cuidado con las líneas que separan las columnas. Observa que puedes trabajar a partir de la tabla ya codificada.*

Tema:	***Clases de documentos en*** LaTeX	
Ambiente	***Nombre***	DESCRIPCIÓN
Reporte	*report*	*Se utiliza para cosas menos formales que un libro o un artículo. No tiene capítulos, sino que agrupa a partir de secciones.*
Artículo	*article*	*Es, tal vez, el ambiente que más seguido utilizarás para la entrega de tareas, elaboración de pequeños manuales, etc. Te permite escribir en dos columnas fácilmente.*
*** La lista no es exhaustiva*		

6.7.3. Líneas "incompletas" verticales y horizontales

Veamos la tabla que se encuentra a continuación a la izquierda. Entre el segundo y tercer renglón no quieres una línea completa, sino únicamente una que cubra la segunda y tercera columna. Esto se consigue con el comando `\cline{`*ci*`-`*cf*`}`, donde *ci* es el número de la columna inicial y *cf* es el número de la columna final. A su lado se muestra la codificación de la tabla, donde puedes ver en la cuarta línea de la codificación el uso del comando `\cline{2-3}`, que indica que la línea parcial empieza en la columna 2 y termina en la 3.

Verificador de Paridad			
	Sigma		
Q	0	1	F
PAR	PAR	NON	1
NON	NON	PAR	0

```
\begin{tabular}{|c|c|c||c|}
  \hline
  \multicolumn{4}{|c|}{\bf Verificador\ de\
    Paridad}\\ \hline\hline
  &\multicolumn{2}{c||}{\bf Sigma}
  &\\\cline{2-3}
  \bf Q & 0 & 1 & F\\\hline
  PAR & PAR & NON & 1\\\hline
  NON & NON & PAR & 0\\\hline
\end{tabular}
```

Tienes también el comando \vline que introduce una línea de separación de columna,

que cubre toda la columna. Se codificó como se muestra a la derecha de la tabla formada.

Esta es		una prueba
Para mostrar	la división	artificial

```
\begin{tabular}{|c|l|}
   \hline
   Esta es & una prueba\\
   \hline
   Para mostrar \vline\  la
    división & artificial\\ \
   \hline
\end{tabular}
```

Como se puede ver, en la cuarta línea del código aparece, a mitad de la columna, el comando \vline. Va seguido de la secuencia \ que obliga a dejar un espacio en blanco entre el comando y lo que sigue. También hay que recordar que el formato del código no influye en el de la tabla, por lo que puedes partir renglones arbitrariamente en el código.

6.7.4. Espacio entre los renglones

Para terminar con esta sección hablaremos un poco de cómo "estirar" el espacio entre renglones de una tabla, forzando a que las separaciones de columna se mantengan. Esto es deseable cuando quieres que haya más separación entre renglones que la normal, pero no tanto espacio como lo da un renglón en blanco. Puedes modificar la distancia dada por el cambio del renglón colocando un argumento a continuación de \\ con el formato $[< num >< medida >]$. Veamos un ejemplo, en el que agregamos 4 y 10 puntos al cambio de renglón. El código se encuentra a la derecha y la tabla formada a la izquierda.

Esta es		una prueba
Para mostrar	la división	artificial

```
\begin{tabular}{|c|l|}
   \hline
   Esta es & una prueba\\[4pt]
   \hline
   Para mostrar \vline\  la
     división &
     artificial\\[10pt]
   \hline
\end{tabular}
```

Observa que las líneas verticales que separan a las columnas se extienden para cubrir toda la columna.

$< num >$ puede ser negativo también, forzando a que haya un "retroceso". El código se encuentra a la derecha de la tabla formada:

Esta es		una prueba
Para mostrar	la división	artificial
Acá mostramos	el efecto de “subir” la raya	

```
\begin{tabular}{|c|l|}
   \hline
   Esta es & una prueba\\[6pt]
   \hline
   Para mostrar \vline\  la
    división
   & artificial\\[10pt]
   \hline
   \multicolumn{2}{|l|}{Acá
      mostramos \vline\
      el efecto de ``subir''
      la raya}\\[-6pt]
   \hline
\end{tabular}
```

Actividad 6.23 *Codifica la siguiente tabla:*

Bonetería Científica, S. A.			
	Precio		
Año	*bajo*	*alto*	*Comentarios*
1971	*97-245*		*Mal año.*
72	*245-245*		*Poco comercio debido a un invierno crudo.*
73	*245-2001*		*Ningún gnus fue buen gnus este año.*

6.8. Fórmulas matemáticas y símbolos especiales

Hasta ahora has escrito texto que si bien te permite escribir, por ejemplo, una novela, no te sirve de mucho para cuando quieres escribir texto un poco más especializado. En estas notas han aparecido muchos carácteres especiales. En general, deseas poder escribir carácteres especiales y, en particular, fórmulas matemáticas.

Para algunos carácteres especiales, como pudieran ser las vocales acentuadas del español, las diéresis en la u o la tilde en la n, hemos logrado que Emacs se comporte como máquina de escribir y haga que el apóstrofo (’), la tilde (~) y las comillas (") se comporten lo que se conoce como símbolos “sordos”, esto es, que “no escuchen” hasta que se teclee el siguiente carácter – o bien está configurado el teclado para que la combinación de tecla con Alt produzca los carácteres acentuados –. Sin embargo, pudiera ser que no estuviera

Emacs habilitado de esta manera. En general, LaTeX provee mecanismos para que se impriman una gran cantidad de símbolos que no aparecen en el inglés, pero que se usan en idiomas como el francés, el español y el alemán. Casi todos empiezan con una diagonal inversa \ y siguen con el código especial dado a ese carácter. LaTeX los llama *símbolos*. Veamos a continuación una lista de los más usados:

Tabla 6.6 Acentos

`\`{o}`	ò	`\~{o}`	õ	`\v{o}`	ǒ	`\c{o}`	o̧	`\'{o}`	ó
`\={o}`	ō	`\H{o}`	ő	`\d{o}`	ọ	`\^{o}`	ô	`\.{o}`	ȯ
`\t{oo}`	o͡o	`\b{o}`	o̲	`\"{o}`	ö	`\u{o}`	ŏ	`\'{\i}`	í

En todas estas combinaciones puede aparecer cualquier carácter alfabético. En el caso de que lo que siga a la \ sea un símbolo especial, no es necesario poner al argumento entre llaves: `\~a` ã y `\^d` d̂.

Los símbolos que siguen no tienen argumentos, sino que se sustituyen tal cual se especifican.

Tabla 6.7 Símbolos no ingleses

\ae	æ	\oe	œ	\aa	å	\AA	Å	\AE	Æ
\OE	Œ	\O	Ø	\l	ł	\L	Ł	?`	¿
!`	¡	\o	ø	\ss	ß				

Se tienen algunos símbolos de puntuación que resultan útiles. Se incluyen acá también los símbolos que tienen algún significado especial en LaTeX.

Tabla 6.8 Símbolos de puntuación y especiales

\dag	†	\S	§	\copyright	©			\P	¶
\ddag	‡	\pounds	£	`\#`	#			`\$`	$
`\%`	%	`\&`	&	`\_` \&	_	&&verb+{ }+		{ }	

Finalmente, cuando se trate de que un símbolo aparezca “tal cual”, sin que sea interpretado por LaTeX (por ejemplo, la diagonal inversa \) simplemente usas el comando \verb+*texto*+ y todo el texto que aparece entre los símbolos “+” no será interpretado y aparecerá en la página tal cual lo tecleaste[2].

[2]Usando este comando es como se han escrito algunos de los comandos en todo este texto.

Actividad 6.24 *Crea un archivo nuevo copiando el preámbulo que hemos visto hasta ahora.*

Actividad 6.25 *Construye una tabla de cinco columnas en las que veas el resultado de aplicar seis de los tipos de acentos a las cinco vocales.*

6.8.1. Fórmulas matemáticas

El tipo de texto que hemos escrito hasta ahora es lo que LaTeX denomina texto LR (*left to right*) y que consiste fundamentalmente de párrafos que se acomodan en la hoja como vienen. Cuando quieres escribir fórmulas matemáticas, éstas contienen símbolos que obligan al compilador a realizar un formato también vertical, no nada más en el renglón correspondiente. Para lograrlo deberás introducir un "ambiente" matemático. Tienes dos tipos de ambientes matemáticos:

i. En la línea, donde lo que buscas es poder intercalar una fórmula o un carácter ($\alpha_i = 2 \cdot \beta^3$). Esto se logra, de cualquiera de las siguientes maneras:

Se escribe:	**Se forma:**
$\alpha = 2 \cdot \beta^3$	$\alpha_i = 2 \cdot \beta^3$
\(\ alpha_i=2\cdot\beta^3\)	$\alpha_i = 2 \cdot \beta^3$
\begin{math}\alpha_i=2\cdot\beta^3\end{math}	$\alpha_i = 2 \cdot \beta^3$

ii. En un "recuadro" que contiene a la fórmula matemática. Introduces también un ambiente que dejará espacio en blanco entre el párrafo anterior y el siguiente. Dentro de ese párrafo puedes escribir las fórmulas. Para mostrar esta modalidad, se mostrará del lado izquierdo el texto fuente y del derecho el texto formado:

```
\[\alpha=2\cdot\beta^3\]
```

$$\alpha_i = 2 \cdot \beta^3$$

```
\begin{displaymath}
    \alpha_i=2\cdot\beta^3
\end{displaymath}
```

$$\alpha_i = 2 \cdot \beta^3$$

La diferencia fundamental entre el modo de despliegue (párrafo) y el de línea es el tamaño de los términos. Mientras que en el modo de línea se ajusta el tamaño a la altura normal de una línea, en el modo de párrafo ocupa tanto espacio vertical como requiera. Compara estos dos modos en el siguiente esquema.

\(2\ frac{n}{n+1}\) $2\frac{n}{n+1}$ \[2\ frac{n}{n+1}\] $$2\frac{n}{n+1}$$

Cuando estás en modo matemático, ya sea de línea o de recuadro, los espacios en blanco únicamente tendrán sentido para separar comandos. No aparecerán en las fórmulas que escribas como tales, a menos que utilices un blanco precedido por la diagonal inversa. El texto que escribas se formará con un tipo parecido al de las itálicas, y si escribes varias palabras, aparecerán sin los espacios. Compara el modo matemático con el tipo itálico en los siguientes renglones.

\(Esta\ es\ una\ prueba\)	$Esta\ es\ una\ prueba$
{\ it Esta es una prueba}	*Esta es una prueba*

Nota que el lugar que ocupa cada carácter es ligeramente mayor en el modo matemático.

Revisemos ahora cuáles son los componentes principales de las fórmulas matemáticas.

Exponentes

Una de las acciones más comunes que quieres realizar es la de poner exponentes. Esto se logra fácilmente si donde quieres deseas un exponente colocas el símbolo ^ seguido de un sólo símbolo, para que ese símbolo aparezca como exponente; o bien seguido de varios símbolos, todos ellos entre llaves. No se permiten secuencias de más de una combinación de ^ con el exponente. Siguen algunos ejemplos:

$a^m b^n c^{n+m}$	$a^m b^n c^{n+m}$
x^{y^2}	x^{y^2}
x^{2y}	x^{2y}
x^2y	$x^2 y$

En el segundo ejemplo nos vimos forzados a encerrar el segundo exponente entre llaves para que no se infringiera la regla de que hubiera más de uno seguido. De esta manera, el exponente de x es, simplemente, otra expresión con exponente.

Como puedes observar en todos estos ejemplos, las llaves sirven para indicar dónde empieza y termina el exponente. Es claro, por ejemplo en el último caso, que en ausencia de llaves LaTeX considerará únicamente al primer carácter que sigue a ^.

Actividad 6.26 *Codifica en* LaTeX *las siguientes fórmulas:*

$$3x^2 + 4xy + 2y^2 = 0$$

$$-b + (b^2 - 4ac)^{1/2}/(2a)$$

$$a^{n!} b^{m!} c^{(n+m)!}$$

$$(x - y)^2 (x^2 - y^2)(x + y)^2$$

Subíndices

Funcionan de la misma manera que los exponentes, excepto que se usa el símbolo _ de subrayado en lugar del ^. Las mismas reglas aplican para subíndices que para exponentes. Siguen algunos ejemplos:

`$a_m b_n c_{n+m}$`	$a_m b_n c_{n+m}$
`$x_{y_2}$`	x_{y_2}
`$x_{2y}$`	x_{2y}
`$x_2y$`	$x_2 y$

Actividad 6.27 *Codifica en* LaTeX *las siguientes expresiones:*

$$P_i log_2 P_i$$

$$(x_{i_1} - y_{i_1}) + (x_{i_2}^2 - y_{i_2}^2) + (x_{i_3}^3 - y_{i_3}^3)$$

$$(x_1^2 + y_1^2)^{1/2}$$

$$(X_{max} - X_{min})^2 / (Y_{max} - Y_{min})^2$$

6.9. Emacs y LaTeX

En este momento ya conoces una buena cantidad de herramientas LaTeX, has incursionado incluso en matemáticas en LaTeX. Éste es un buen momento para que revises qué puede hacer Emacs por ti. Mejor dicho, qué puede hacer AUCTeX, el modo mayor para editar LaTeX en Emacs por ti.

En la sección 6.12.1, en la página 238, explicamos con detalle qué comandos de AUCTeX ejecutan las secuencias C-c C-e y otras que hemos mencionado en este capítulo. También agregamos información sobre el modo para ingresar símbolos matemáticos de manera rápida y un buen número de detalles más.

AUCTeX, desgraciadamente, no es parte estándar de todas las distribuciones de Emacs. Por ello, en la sección 6.12.1 te explicamos cómo obtener, instalar y configurar AUCTeX.

Letras griegas

Para que se imprima una letra griega haces preceder el nombre de la misma (en inglés) por la diagonal inversa. En la tabla 6.9 se muestran las letras griegas disponibles:

Tabla 6.9 Símbolos griegos

Minúsculas

\alpha	α	\theta	θ	o	o	\tau	τ
\beta	β	\vartheta	ϑ	\pi	π	\upsilon	υ
\gamma	γ	\iota	ι	\varpi	ϖ	\phi	ϕ
\delta	δ	\kappa	κ	\rho	ρ	\varphi	φ
\epsilon	ϵ	\lambda	λ	\varrho	ϱ	\chi	χ
\varepsilon	ε	\mu	μ	\sigma	σ	\psi	ψ
\zeta	ζ	\nu	ν	\varsigma	ς	\omega	ω
\eta	η	\xi	ξ				

Mayúsculas

\Gamma	Γ	\Lambda	Λ	\Sigma	Σ	\Psi	Ψ
\Delta	Δ	\Xi	Ξ	\Upsilon	Υ	\Omega	Ω
\Theta	Θ	\Pi	Π	\Phi	Φ		

Puedes ver que no para todas las letras griegas hay la versión en mayúscula. Si no aparece en esta lista es porque no la hay. También existe, para algunas de ellas, la versión para variable.

Actividad 6.28 *Haz una tabla que contenga únicamente las letras griegas para las cuales hay mayúsculas, y aquéllas para las cuales hay versión en* var. *En la primera columna de la tabla deberá aparecer el "nombre" de la letra en minúscula; en la segunda columna deberá imprimirse el símbolo; en la tercera columna, si es que hay, el símbolo para la mayúscula; y en la cuarta columna, si es que hay, el símbolo para la variable.*

Símbolos con flechas

En matemáticas, y en computación en general, se usan muy frecuentemente distintos tipos de flechas. Presentamos algunos en la tabla 6.10.

Tabla 6.10 Símbolos con flechas

\leftarrow	$\leftarrow$	\longleftarrow	$\longleftarrow$	\Leftarrow	$\Leftarrow$
\Longleftarrow	$\Longleftarrow$	\rightarrow	$\rightarrow$	\longrightarrow	$\longrightarrow$
\Rightarrow	$\Rightarrow$	\Longrightarrow	$\Longrightarrow$	\ leftrightarrow	$\leftrightarrow$
\ longleftrightarrow	$\longleftrightarrow$	\Leftrightarrow	$\Leftrightarrow$	\Longleftrightarrow	$\Longleftrightarrow$
\uparrow	$\uparrow$	\Uparrow	$\Uparrow$	\downarrow	$\downarrow$
\Downarrow	$\Downarrow$	\updownarrow	$\updownarrow$	\Updownarrow	$\Updownarrow$
\mapsto	$\mapsto$	\longmapsto	$\longmapsto$	\hookleftarrow	$\hookleftarrow$
\hookrightarrow	$\hookrightarrow$	\leftharpoonup	$\leftharpoonup$	\rightharpoonup	$\rightharpoonup$
\leftharpoondown	$\leftharpoondown$	\rightharpoondown	$\rightharpoondown$	\rightleftharpoons	$\rightleftharpoons$
\leadsto	$\leadsto$	\nearrow	$\nearrow$	\searrow	$\searrow$
\swarrow	$\swarrow$	\nwarrow	$\nwarrow$		

Operadores

En la tabla 6.11 te mostramos símbolos comúnmente usados en matemáticas como operadores y símbolos relacionales.

Tabla 6.11 Operadores y símbolos relacionales

\pm	$\pm$	\cap	$\cap$	\diamond	$\diamond$
\oplus	$\oplus$	\mp	$\mp$	\cup	$\cup$
\bigtriangleup	$\bigtriangleup$	\ominus	$\ominus$		
\times	$\times$	\uplus	$\uplus$	\bigtriangledown	$\bigtriangledown$
\otimes	$\otimes$	\div	$\div$	\sqcap	$\sqcap$
\ triangleleft	$\triangleleft$	\oslash	$\oslash$	\ast	$\ast$
\sqcup	$\sqcup$	\ triangleright	$\triangleright$	\odot	$\odot$
\star	$\star$	\vee	$\vee$	\lhd	$\lhd$
\ bigcirc	$\bigcirc$	\ circ	$\circ$	\wedge	$\wedge$
\rhd	$\rhd$	\dagger	$\dagger$	\ bullet	$\bullet$
\setminus	$\setminus$	\unlhd	$\unlhd$	\ddagger	$\ddagger$
\cdot	$\cdot$	\wr	$\wr$	\unrhd	$\unrhd$
\amalg	$\amalg$	\leq	$\leq$	\geq	$\geq$
\equiv	$\equiv$	\models	$\models$	\prec	$\prec$
\succ	$\succ$	\sim	$\sim$	\perp	$\perp$
\preceq	$\preceq$	\succeq	$\succeq$	\simeq	$\simeq$

Continúa en la siguiente página

Tabla 6.11 Operadores y símbolos relacionales *Continúa de la página anterior*

\mid	$\mid$	\ ll	$\ll$	\gg	$\gg$
\asymp	$\asymp$	\ parallel	$\parallel$	\subset	$\subset$
\supset	$\supset$	\approx	$\approx$	\bowtie	$\bowtie$
\subseteq	$\subseteq$	\supseteq	$\supseteq$	\cong	$\cong$
\Join	$\bowtie$	\sqsubset	$\sqsubset$	\sqsupset	$\sqsupset$
\neq	$\neq$	\smile	$\smile$	\sqsubseteq	$\sqsubseteq$
\sqsupseteq	$\sqsupseteq$	\doteq	$\doteq$	\frown	$\frown$
\in	$\in$	\ni	$\ni$	\notin	$\notin$
\propto	$\propto$	\vdash	$\vdash$	\dashv	$\dashv$

Actividad 6.29 *Codifica en* LaTeX *las siguientes fórmulas:*

$$f_X(y_i)\Delta_i log[f_X(y_i)\Delta_i]$$

$$\parallel X - Y_i^{(k)} \parallel^2 f_X dX$$

$$m_k(t_{ij}) = i(\alpha_k t_{ij} + \Delta_k)$$

Actividad 6.30 *Codifica en* LaTeX *las siguientes fórmulas:*

$$p \wedge 1 \equiv \neg p \Rightarrow q$$

$$(p \wedge q) \Leftrightarrow \neg(\neg p \vee \neg q)$$

$$\parallel X - Y_j \parallel^2 \leq \parallel X - Y_i \parallel^2, Y_i \in C$$

$$d(x_i, y_i) = x_i \oplus y_j$$

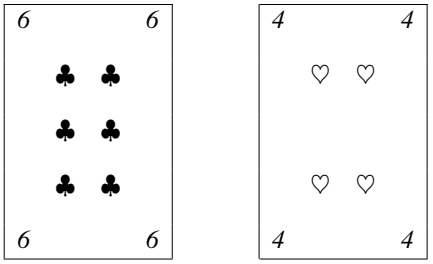
Símbolos varios

Notarás que a pesar de que hemos dado ya muchísimos símbolos matemáticos, todavía nos faltan algunos que ya conoces como el "para todo". En la tabla 6.12 ponemos algunos símbolos más que te van a ser de utilidad.

Tabla 6.12 Símbolos varios

\aleph	$\aleph$	\prime	$\prime$	\ forall	$\forall$	\ infty	∞
\hbar	$\hbar$	\emptyset	$\emptyset$	\exists	$\exists$	\Box	$\Box$
\imath	$\imath$	\nabla	∇	\neg	$\neg$	\Diamond	$\Diamond$
\jmath	$\jmath$	\surd	$\surd$	\ flat	$\flat$	\ triangle	$\triangle$
\ ell	ℓ	\top	$\top$	\natural	$\natural$	\ clubsuit	$\clubsuit$
\wp	$\wp$	\bot	$\bot$	\sharp	$\sharp$	\diamondsuit	$\diamondsuit$
\Re	$\Re$	\ |	$\|$	\backslash	$\backslash$	\ heartsuit	$\heartsuit$
\Im	$\Im$	\angle	$\angle$	\ partial	∂	\spadesuit	$\spadesuit$
\mho	$\mho$						

Actividad 6.31 *Codifica en* LaTeX *las siguientes figuras (pista: utiliza una tabla* tabular *para acomodar las figuras):*



Nombres de funciones comunes

LaTeX proporciona varios nombres de funciones comunes para que las escribas de manera especial, y no simplemente como cadenas. Éstas se presentan en la tabla 6.13.

Tabla 6.13 Nombres de funciones comunes

\arccos	$\arccos$	\cos	$\cos$	\csc	$\csc$	\exp	$\exp$
\ker	$\ker$	\limsup	$\limsup$	\min	$\min$	\sinh	$\sinh$
\arcsin	$\arcsin$	\cosh	$\cosh$	\deg	$\deg$	\gcd	$\gcd$
\lg	$\lg$	\ln	$\ln$	\Pr	$\Pr$	\sup	$\sup$
\arctan	$\arctan$	\cot	$\cot$	\det	$\det$	\hom	$\hom$
\lim	$\lim$	\log	$\log$	\sec	$\sec$	\tan	$\tan$
\arg	$\arg$	\coth	$\coth$	\dim	$\dim$	\ inf	$\inf$

Continúa en la siguiente página

Tabla 6.13 Nombres de funciones comunes *Continúa de la página anterior*

\liminf	$\liminf$	\max	$\max$	\sin	$\sin$	\tanh	$\tanh$
a \mod b	$a \mod b$			\pmod{b}	$\pmod{b}$		

Nota que como estamos trabajando en español (paquete babel) aquellas abreviaturas que lo requieren llevan acento.

Fracciones

En el modo matemático tienes dos maneras de escribir fracciones. La primera de ellas es con la diagonal, como en

```
$3/4$        3/4
$1/(n+1)$    1/(n+1)
```

También se puede usar el comando `\frac{numerador}{denominador}` que escribe la fracción de la siguiente forma:

```
$\frac{3}{4}$    \frac{3}{4}        \[\frac{3}{4}\]
```

$$\frac{3}{4}$$

```
$\frac{1}{n+1}$    \frac{1}{n+1}        \[\frac{1}{n+1}\]
```

$$\frac{1}{n+1}$$

Si \frac usas en el modo matemático en línea, LaTeX va a tratar de ajustar la fórmula al alto de un renglón, mientras que si usas el modo matemático de recuadro, ocupará tanto espacio vertical como requiera.

Actividad 6.32 *Codifica en LaTeX las siguientes fórmulas:*

$$z_n = x_n - y_n = x_n - \frac{x_n + x_{n-1}}{2} = \frac{x_n - x_{n-1}}{2}$$

$$f_X(x) \log \frac{2X_{\text{máx}}/M}{c'(x)} \Delta_i$$

Raíces

La raíz de una fórmula o expresión se logra con el comando \sqrt y poniéndole como argumento entre llaves a la expresión que quieres cubra la raíz.

```
\frac{x+y}{1+\sqrt{\frac{y}{z+1}}}
```

$$\frac{x+y}{1+\sqrt{\frac{y}{z+1}}}$$

Puedes también poner valor en el radical, agregando ese valor entre corchetes como primer argumento de \sqrt:

```
\frac{x+y}{1+\sqrt[n]{\frac{y}{z+1}}}
```

$$\frac{x+y}{1+\sqrt[n]{\frac{y}{z+1}}}$$

Actividad 6.33 *Codifica en LaTeX las siguientes fórmulas:*

$$\frac{-b \pm \sqrt[2]{b^2 \cdot a \cdot c}}{2 \cdot a}$$

$$\Sigma_3(\chi) = \sqrt{(\frac{1}{\kappa^2} - \frac{1}{(\kappa+1)^2})\chi^{-\kappa}}$$

Símbolos agrandables

Tenemos en matemáticas muchos símbolos, como el de la integral o las series, que deben tomar el tamaño dado por todo lo que está a su derecha. Como vimos en el ejercicio anterior, la raíz cuadrada, por ejemplo, crece al tamaño que toma su argumento. Esto lo hace automáticamente. Los símbolos en la tabla 6.14 son con los que cuenta LaTeX para el modo matemático. El tamaño aumenta únicamente cuando aparece en el modo en que acomoda a la fórmula en un párrafo, no en una línea.

Tabla 6.14 Símbolos agrandables

\sum	$\sum$	\bigcap	$\bigcap$	\bigodot	$\bigodot$
\prod	$\prod$	\bigcup	$\bigcup$	\bigotimes	$\bigotimes$
\coprod	$\coprod$	\bigsqcup	$\bigsqcup$	\bigoplus	$\bigoplus$
\int	$\int$	\bigvee	$\bigvee$	\biguplus	$\biguplus$
\oint	$\oint$	\bigwedge	$\bigwedge$		

Cuando pones subíndice o exponente a cualquiera de estos símbolos, LaTeX los acomodará de tal manera que se vean bien. Por ejemplo,

```
\[\sum_{-\infty}^\infty f_{X|Y}(x_i|y_j)\]
```

$$\sum_{-\infty}^{\infty} f_{X|Y}(x_i|y_j)$$

```
\[\int_0^n f(x)^2g(y)^2\]
```

$$\int_0^n f(x)^2 g(y)^2$$

Mientras que si los pones en modo de línea, aparecerán como sigue:

`$\sum_{-\infty}^\infty f_{X\|Y}(x_i\|y_j)$`	se forma	$\sum_{-\infty}^{\infty} f_{X\|Y}(x_i\|y_j)$
`$\int_0^n f(x)^2g(y)^2$`	se forma a	$\int_0^n f(x)^2 g(y)^2$.

Otra manera de agrandar algunos de los símbolos, sobre todo aquellos que forman parejas como los paréntesis, los corchetes o las llaves, es "aclarando" que vas a tener un delimitador izquierdo y uno derecho para un determinado pedazo de fórmula, usando los comandos `\left` y `\right`, y colocando inmediatamente a continuación el símbolo que se va a usar. En el ejemplo que dimos para las raíces, los paréntesis no crecieron al tamaño de la fórmula que parentizan. Para lograrlo haces lo siguiente:

```
\[\Sigma_3(\chi)=\sqrt{\left(\frac{1}{\kappa^2}-\frac{1}
{(\kappa+1)^2}\right)\chi^{-\kappa}}\]
```

$$\Sigma_3(\chi) = \sqrt{\left(\left(\frac{1}{\kappa^2} - \frac{1}{(\kappa+1)^2}\right)\chi^{-\kappa}\right)}$$

Puedes querer únicamente el símbolo de la izquierda o el de la derecha. En ese caso, como símbolo a agrandar usas un punto (.). Veamos un ejemplo:

```
\[P(x)=\left\{\frac{x^2+y^2}{\sqrt{(x^2-y^2)^2}}\right.\]
```

$$P(x) = \left\{ \frac{x^2+y^2}{\sqrt{(x^2-y^2)^2}} \right.$$

Actividad 6.34 *Codifica en* LaTeX *las siguientes fórmulas:*

$$\left(E_y \int_0^{t_\varepsilon} L_{x,y^k(s)}\varphi(\chi)ds\right)$$

$$y_n = \sum_{i=0}^{N} a_i x_{n-i} + \sum_{i=1}^{M} b_i y_{n-i}$$

$$D^{(k)} = \sum_{i=1}^{M} \int_{b_{i-1}^{(}k)}^{b_i^{(k)}} (x-y_i)^2 f_X(x)dx$$

Equivalente a `\left` y `\right` tenemos manera de pedir distintos tamaños de delimitadores con las siguientes parejas, listadas de menor a mayor. El comando se hace seguir por un único símbolo.

```
\bigl       \bigr
\Bigl       \Bigr
\biggl      \biggr
\Biggl      \Biggr
```

Por ejemplo, la siguiente fórmula

```
\[  \Biggl\{  \biggl[\Bigl(\bigl|prueba\bigr|\Bigr)\biggr]\Biggr\}\]
```

se forma de la siguiente manera:

$$\left\{\left[\left(|prueba|\right)\right]\right\}$$

La lista de los símbolos que puedes usar como delimitadores y que se agrandan se encuentra en la tabla 6.15.

Tabla 6.15 Delimitadores

$\uparrow$	\uparrow	$\Uparrow$	\Uparrow	$\downarrow$	\downarrow	$\Downarrow$	\Downarrow
$\{$	\{	$\}$	\}	$\updownarrow$	\updownarrow	$\Updownarrow$	\Updownarrow
$\lfloor$	\lfloor	$\rfloor$	\rfloor	$\lceil$	\lceil	$\rceil$	\rceil
$\langle$	\langle	$\rangle$	\rangle	/	/	$\backslash$	\backslash
$\vert$	|	$\Vert$	\\|	⎭	\rmoustache	⎧	\lmoustache
⎫	\rgroup	⎩	\lgroup	$\vert$	\arrowvert	$\Vert$	\Arrowvert
⎪	\bracevert						

Actividad 6.35 *Codifica en* LaTeX *la siguiente fórmula:*

$$\sum_{1<m\leq n}\left\lfloor\left(\sum_{1\leq k<m}\lfloor(m/k)/\lceil m/k\rceil\rfloor\right)^{-1}\right\rfloor.$$

Puntos suspensivos (elipses)

Cuando en modo matemático (o de texto) deseas poner puntos suspensivos, el teclearlo directamente en LaTeX como tres puntos seguidos no da el efecto deseado, ya que acerca mucho entre sí a los puntos. Por ello cuentas con dos maneras de poner puntos suspensivos:

```
\[ \text{valores para } x_1,\ldots,x_n\]
```

valores para $x_1, \ldots, x_n$

donde los puntos suspensivos se colocan alineados con la parte inferior de la línea (\ldots); o bien, alineados con la parte central de la línea (\cdots):

```
\[1+\frac{1}{x^2}+\frac{1}{x^4}+\cdots +\frac{1}{x^{2^n}}\]
```

$$1+\frac{1}{x^2}+\frac{1}{x^4}+\cdots+\frac{1}{x^{2^n}}$$

donde los puntos suspensivos se alinean con el operador para darle continuidad. También cuentas con puntos suspensivos verticales, \vdots:($\vdots$) y diagonales, \ddots ($\ddots$).

Actividad 6.36 *Codifica en* LaTeX *las siguientes fórmulas:*

$$F(\chi) = a_d\chi^d + a_{d-1}\chi^{d-1} + \cdots + a_1\chi^1 + a_0\chi^0$$

$$\Delta^n f(\chi) = c_d\left(\frac{\chi}{d-n}\right) + c_{d-1}\left(\frac{\chi}{d-1-n}\right) + \cdots + c_0\left(\frac{\chi}{-n}\right)$$

Cambios de tipos y tamaños de letras

En el modo para matemáticas no van a funcionar los comandos que cambian el tipo o tamaño de letra. Tampoco vas a poder utilizar las letras acentuadas o los símbolos que no son del idioma inglés. Para intercalar un símbolo de éstos en un ambiente matemático lo debes colocar como argumento de un comando \mbox. Veamos un ejemplo:

$\delta(q\underline{o}, a)$

```
\begin{displaymath}
  \delta(q \mbox{\b{o}}, a)
\end{displaymath}
```

En el modo matemático tampoco van a funcionar las negritas u otros cambios en el tipo de letra. El modo matemático tiene sus propios tipos, que se muestran en la tabla 6.16.

Tabla 6.16 Tipos de letras en modo matemático

`$\mathcal{ABCDEF...}$`	$\mathcal{ABCDEF...}$
`$\mathit{ it \acute{a}licas \cdot 2^{ ft }\Psi \log [\ psi ]}$`	$it\acute{a}licas \cdot 2^{ft}\Psi \log[\psi]$
`$\mathrm{romanas \cdot 2^{ft}\Psi \log [\ psi ]}$`	$\mathrm{romanas} \cdot 2^{\mathrm{ft}}\Psi \log[\psi]$

Continúa en la siguiente página

Tabla 6.16 Tipos de letras en modo matemático *Continúa de la página anterior*

`$\mathbf{negritas \cdot 2^{ft}\Psi \log[\psi]}$`	$\mathbf{negritas \cdot 2^{ft}\Psi \log[\psi]}$
`$\mathsf{sans\ serif \cdot 2^{ft}\Psi \log[\psi]}$`	$\mathsf{sans\ serif \cdot 2^{ft}\Psi \log[\psi]}$
`$\mathtt{m\acute{a}quina\ de\ escribir \cdot 2^{ft}\Psi \log[\psi]}$`	$\mathtt{máquina\ de\ escribir \cdot 2^{ft}\Psi \log[\psi]}$
`$\mathbb{BLACKBOARD CON BORDES DOBLES}$`	$\mathbb{BLACKBOARD\ CON\ BORDES\ DOBLES}$

Debes notar que todos estos tipos de letra mantienen al texto en modo matemático. También es importante repetir que para separar palabras debes teclear un espacio "\ ", pues si no el modo matemático no respeta espacios en blanco.

El primer y último tipo (\mathcal y mathbb únicamente los puedes usar con mayúsculas.

Cuando pides negritas en modo matemático con \mathbf hay símbolos que no se imprimen en negritas. Observa que en el cuarto ejemplo, los símbolos correspondientes a $\Psi \log[\psi]$ no salieron obscurecidos. Para que salgan obscurecidos debes usar el comando

```
\boldsymbol{ ... }
```

que logra poner en negritas también a los símbolos matemáticos. Mostramos ambas opciones juntas para que se note la diferencia:

```
\[\mathbf{negritas \cdot 2^{ft}\Psi \log[\psi]}\]
```

$$\mathbf{negritas \cdot 2^{ft}\Psi \log[\psi]}$$

```
\[\mathbf{negritas \cdot 2^{ft}\boldsymbol{\Psi \log[\psi]}}\]
```

$$\mathbf{negritas \cdot 2^{ft}\boldsymbol{\Psi \log[\psi]}}$$

Otra manera de lograr que todo el texto matemático se escriba en negritas es mediante el comando de cambio de estado \boldmath (\unboldmath), que a partir del momento en que aparece prende (apaga) las negritas en modo matemático. Veamos un ejemplo:

```
\hspace*{4em}\boldmath \(a+b\Box c\)
    \unboldmath\qquad \(a+b\Box c\)
```

$\boldsymbol{a+b\Box c}$ $\qquad$ $a+b\Box c$

Cabe hacer la aclaración que estos comandos los tienes que poner antes de que entres al modo matemático, para que tengan efecto.

Actividad 6.37 *Codifica en LaTeX la siguiente fórmula, cuidando el tipo de letra y las negritas:*

$$\text{Sea } x \neq y \quad \forall x \in \mathbb{N},\ \text{y } \mathbf{y \geq x}$$

Insertar texto en modo matemático

Es necesario muchas veces introducir en un recuadro o línea matemática texto que quieres aparezca en modo normal. Nota que esto no es lo mismo que hicimos cuando cambiamos el tipo de letra para los símbolos matemáticos. Para ello se puede usar el comando \text {...} poniendo entre llaves el texto que deseas aparezca. También puedes utilizar, en general, cualquiera de los comandos \text$xx$ que vimos al inicio de esta sección para modificar tipo de texto (donde xx es uno de bf, sf, tt, etc.).

Este comando preserva los espacios entre palabras:

```
\[x=\frac{-b\pm \sqrt{b^2-4ac}}{2a}\ \
    \text{Da la solución de ecuación cuadrática}\]
```

$$x=\frac{-b\pm \sqrt{b^2-4ac}}{2a}\ \text{Da la solución de ecuación cuadrática}$$

Tablas de fórmulas

Hay dos maneras de organizar varias fórmulas en una tabla. La primera de ellas es con un tabular y colocando en cada celda el modo matemático en línea ($ \(... $\)). La otra manera es declarando un ambiente de línea o recuadro y colocando allí un arreglo matemático, \begin{array }...\ end{array}. Es importante insistir en que un array sólo puede estar en un ambiente matemático. Las declaraciones de columnas y renglones las haces exactamente igual que con tabular, excepto que el contenido es interpretado como fórmulas. La excepción es cuando para describir la columna diste p{<num><mdda>}, pues en ese caso automáticamente esa columna quedará en modo texto.

Este mecanismo se utiliza también para imprimir matrices o, en general, tablas que en su mayoría van a contener fórmulas matemáticas.

```
\[\begin{array}{llcl}
  a_{11}&a_{12}&\ldots&a_{1m}\\
  a_{21}&a_{22}&\ldots&\ldots\\
  \cdots&\cdots&\ldots&\ldots\\
  a_{n1}&a_{n2}&\cdots
  &a_{nm}
\end{array}\]
```

$$\begin{array}{llcl}
  a_{11}&a_{12}&\ldots&a_{1m}\\
  a_{21}&a_{22}&\ldots&\ldots\\
  \cdots&\cdots&\ldots&\ldots\\
  a_{n1}&a_{n2}&\cdots
  &a_{nm}
\end{array}$$

Siempre que tienes una fórmula que ocupa más de un renglón, puedes ponerle algún carácter a la izquierda y otro a la derecha que crezcan tanto como la fórmula en cuestión. Esto lo haces marcando la fórmula con \left y \right a su izquierda y derecha respectivamente, y poniendo el símbolo que quieras. Por ejemplo, la matriz anterior podría quedar:

```
\[\left[\begin{array}{llcl}
    a_{11}&a_{12}&\ldots&a_{1m}\\
    a_{21}&a_{22}&\ldots&\ldots\\
    \cdots&\cdots&\ldots&\ldots\\
    a_{n1}&a_{n2}&\cdots&a_{nm}
\end{array}\right]\]
```

$$\left[\begin{array}{llcl} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & \ldots \\ \cdots & \cdots & \ldots & \ldots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{array}\right]$$

Si deseas omitir ya sea el símbolo de la izquierda o el de la derecha, pones un punto (.) en lugar del carácter. No tienen porqué ser carácteres correspondientes al inicio y al final. En general, se usan aquellos que lo son, pero no es forzoso:

```
\[\left\|\begin{array}{llcl}
    a_{11}&a_{12}&\ldots&a_{1m}\\
    a_{21}&a_{22}&\ldots&\ldots\\
    \cdots&\cdots&\ldots&\ldots\\
    a_{n1}&a_{n2}&\cdots&a_{nm}
\end{array}\right\}\]
```

$$\left\|\begin{array}{llcl} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & \ldots \\ \cdots & \cdots & \ldots & \ldots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{array}\right\}$$

También puedes tener un array en el modo línea,

```
$\begin{array}[t]{l}a+b\\a-b\end{array}$,
```

que queda $\begin{array}[t]{l}a+b\\a-b\end{array}$. Debes notar que usa tantos renglones como necesite, corrigiendo el siguiente renglón para que no se encime.

Las líneas entre columnas y entre los renglones se utilizan exactamente igual que en el caso de las tablas no numéricas (tabular). También habrás notado que la descripción de las columnas se hace también de la misma manera que en un tabular. También se pueden redefinir las columnas.

Actividad 6.38 *Codifica en* LaTeX *la siguiente tabla:*

$$\begin{array}{lcll} (a+b)^2 & = & a^2+2ab+b^2 & \text{Suma de cuadrados} \\ a^2-b^2 & = & (a+b)(a-b) & \text{Diferencia de cuadrados} \\ (a+b)^n & = & a^n+2a^{n-1}b+\cdots+b^n & \text{Expansión binomial} \end{array}$$

Matrices y combinaciones

Si bien puedes armar matrices usando arrays, LaTeX provee comandos especiales para ello. Por ejemplo, es muy común que quieras definir una función “por casos” como la que sigue:

$$P_{r-j} = \begin{cases} 0 & \text{si } r-j \text{ es impar,} \\ r!(-1)^{(r-j)/2} & \text{si } r-j \text{ es par.} \end{cases}$$

```
\[P_{r-j}=
  \begin{cases}
     0&\text{si } r-j \text es impar},\\
     r! (-1)^{(r-j)/2}
     &\text{si } r-j \text es par}.
  \end{cases}
\]
```

Si esto lo hicieras con un array, lo haría como sigue:

$$P_{r-j} = \left\{ \begin{array}{ll} 0 & \text{si } r-j \text{ es impar,} \\ r!(-1)^{(r-j)/2} & \text{si } r-j \text{ es par.} \end{array} \right.$$

```
\[P_{r-j}=\left\{
   \begin{array}{ll}
     0&\text{si } r-j \text es impar},\\
     r! (-1)^{(r-j)/2}
     &\text{si } r-j \text es par}.
   \end{array}
  \right.
\]
```

que puedes corroborar en la parte izquierda que forma exactamente la misma ecuación. Por ello, el ambiente de casos es, simplemente, un atajo para codificar este tipo de "tablas".

Otro atajo común es el que se usa para matrices. LaTeX define varios ambientes específicos para matrices, dependiendo de los delimitadores que quieras tener:

```
\[\begin{matrix}
    a&b\\
    c&d
  \end{matrix}\]
```

$$\begin{matrix} a & b \\ c & d \end{matrix}$$

```
\[\begin{pmatrix}
    e&f\\
    g&h
  \end{pmatrix}\]
```

$$\begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

```
\[\begin{bmatrix}
    i&j\\
    k&l
  \end{bmatrix}\]
```

$$\begin{bmatrix} i & j \\ k & l \end{bmatrix}$$

```
\[\begin{vmatrix}
    m&n\\
    o&p
  \end{vmatrix}
```

$$\begin{vmatrix} m & n \\ o & p \end{vmatrix}$$

```
\[\begin{Vmatrix}
    q&r\\
    s&t
  \end{Vmatrix}\]
```

$$\begin{Vmatrix} q & r \\ s & t \end{Vmatrix}$$

Como puedes ver, lo que te ahorras al usar estos ambientes es la especificación de los delimitadores y de las columnas de cada matriz.

Actividad 6.39 *Codifica cada una de las matrices anteriores usando* array *y delimitadores.*

Actividad 6.40 *Codifica la siguiente fórmula:*

$$\left(\begin{array}{c} n! \\ r!(n-r)! \end{array} \right)$$

6.9.1. Anotaciones en los símbolos

Es frecuente que quieras anotar a tus variables con distintos símbolos, que van desde prima hasta vectores. También pudieses hacerlo abajo de la variable o expresión. Veamos algunos ejemplos:

\vec{a}	$\vec{a}$
\underset{i=1}{\cup}	$\underset{i=1}{\cup}$

Como puedes notar, aun cuando estemos en modo de línea, acomoda el texto que le indiques donde le indiques. El primer parámetro LaTeX lo escribe con un tamaño menor y lo coloca encima (over) o debajo (under) del segundo argumento. El segundo argumento únicamente puede ser un símbolo. El primer argumento, como ya vimos, puede ser cualquier cosa. Para aquellos símbolos que son muy comunes, tenemos definidos comandos que colocan determinado carácter. Éstos se encuentran en la tabla 6.17.

Tabla 6.17 Acentos en modo matemático

$\hat{a}$	\hat{a}	$\acute{a}$	\acute{a}	$\bar{a}$	\bar{a}	$\dot{a}$	\dot{a}
$\check{a}$	\check{a}	$\grave{a}$	\grave{a}	$\vec{a}$	\vec{a}	$\ddot{a}$	\ddot{a}
$\breve{a}$	\breve{a}	$\tilde{a}$	\ tilde {a}				

Muchas veces quieres poner una flecha, raya, llave, encima de toda una expresión que tiene más de un símbolo. Para ello tenemos los siguientes símbolos que se extienden para cubrir a más de un símbolo:

```
\[ \overline{a+b+c} = \underline{a+b+c} \]
```

$$\overline{a+b+c} = \underline{a+b+c}$$

```
\[ \overbrace{a+b+c} = \underbrace{a+b+c} \]
```

$$\overbrace{a+b+c} = \underbrace{a+b+c}$$

```
\[ \overrightarrow{a+b+c} = \underrightarrow{a+b+c} \]
```

$$\overrightarrow{a+b+c} = \underrightarrow{a+b+c}$$

```
\[\overleftarrow{a+b+c}=\underleftarrow{a+b+c}\]
```

$$\overleftarrow{a+b+c}=\underleftarrow{a+b+c}$$

```
\[\overleftrightarrow{a+b+c}=\underleftrightarrow{a+b+c}\]
```

$$\overleftrightarrow{a+b+c}=\underleftrightarrow{a+b+c}$$

Por supuesto que puedes anidar todos estos comandos y combinarlos de la manera que desees.

Actividad 6.41 *Codifica en* LaTeX *la siguiente expresión:*

$$A \cap B = \begin{cases} A & \textit{si } A \subseteq B \\ B & \textit{si } B \subseteq A \\ \emptyset & \textit{si } A \textit{ y } B \textit{ son ajenos} \\ \overline{\bar{A} \cup \bar{B}} & \textit{por la ley de De Morgan} \end{cases}$$

6.9.2. Arreglos de fórmulas

Seguramente estás pensando en escribir fórmulas (o ecuaciones) en modo párrafo. Hasta ahora has utilizado para ello el ambiente displaymath. Sin embargo, si deseas numerar las ecuaciones o algo por el estilo, debes usar el ambiente equation.

```
\begin{equation}\label{eq:dia3-1}
  R=\frac{1}{M}\sum_{k=1}^M R_k
\end{equation}
```

$$R = \frac{1}{M}\sum_{k=1}^{M} R_k \tag{6.1}$$

Nota que el número de la ecuación aparece en el extremo derecho de la línea donde se encuentra la misma. Esto se puede cambiar, así como la forma de numeración[3]. El contador de ecuaciones se va incrementando progresivamente dentro del capítulo.

Si quieres colocar varias fórmulas alineadas, un array tiene el problema de que no importa si está en modo línea o párrafo, nos dará modo línea para cada renglón del arreglo.

[3] Consulta alguno de los manuales de LaTeX que se mencionan en la bibliografía si así lo deseas.

```
\[\begin{array}[t]{ll}
    x_1=\sum_{k=1}^n a_ky^k
     &\text{primer valor}\\[10pt]
    x_2=\sum_{k=1}^n b_ky^k
     &\text{segundo valor}
   \end{array}
\]
```

$$\begin{array}[t]{ll} x_1=\sum_{k=1}^n a_ky^k & \text{primer valor}\\[10pt] x_2=\sum_{k=1}^n b_ky^k & \text{segundo valor} \end{array}$$

Si quieres varias líneas, pero cada una de ellas en modo párrafo, tienes que usar el ambiente de arreglos de ecuaciones \eqnarray.

```
\begin{eqnarray}
  x_1=\sum_{k=1}^n a_ky^k
  &\text{Primer valor}\\
  x_2=\sum_{k=1}^n b_ky^k
  &\text{segundo valor}
\end{eqnarray}
```

$$x_1=\sum_{k=1}^n a_ky^k \quad \text{Primer valor} \tag{6.2}$$

$$x_2=\sum_{k=1}^n b_ky^k \quad \text{segundo valor} \tag{6.3}$$

Nota que el modo matemático se introduce junto con el ambiente eqnarray, lo mismo que sucede con equation. Si deseas eliminar el número de la(s) ecuación(es) usa los ambientes eqnarray$*$ y equation$*$ respectivamente:

```
\begin{eqnarray*}
  x_1=\sum_{k=1}^n a_ky^k
    &\text{Primer valor}\\
  x_2=\sum_{k=1}^n b_ky^k
    &\text{segundo valor}
\end{eqnarray*}
```

$$\begin{array}{ll} x_1=\sum_{k=1}^n a_ky^k & \text{Primer valor}\\ x_2=\sum_{k=1}^n b_ky^k & \text{segundo valor} \end{array}$$

Las ecuaciones que aparecen dentro de ambientes sin numeración no incrementan el contador de ecuaciones.

Sucede a veces que no cabe una ecuación en una sola línea, por lo que tienes que repartirla en varios renglones. Esto lo logras con el ambiente split dentro de un ambiente equation. Este comando es muy útil cuando, por ejemplo, quieres mostrar procesos por los que pasa una expresión. Nota que de esta manera se asigna un único número a la ecuación, en atención a que es una sola.

```
\begin{equation}
  \begin{split}
    \begin{bmatrix}
      \Theta_1 \\
      \Theta_2
    \end{bmatrix}
    &+\frac{1}{\sqrt{2}}
    \begin{bmatrix}
      1&1\\
      1&-1
    \end{bmatrix}
    \begin{bmatrix}
      \alpha \\
      \alpha
    \end{bmatrix}\\[10pt]
    &-\begin{bmatrix}
      \sqrt{2}\alpha \\
      0
    \end{bmatrix}
  \end{split}
\end{equation}
```

$$\begin{bmatrix}\Theta_1\\ \Theta_2\end{bmatrix} + \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\alpha\\ \alpha\end{bmatrix} - \begin{bmatrix}\sqrt{2}\alpha\\ 0\end{bmatrix} \tag{6.4}$$

Puedes conseguir un efecto similar, pero asignando un número a cada renglón, si usamos el arreglo de ecuaciones, como se muestra a continuación del ambiente split, que comentamos en la segunda versión. En los ambientes de matrices, ecuaciones y arreglos te ahorras especificar número de columnas y tipo de columnas, ya que estos ambientes los toman de los elementos que aparecen en las mismas.

```
\begin{eqnarray}
  %\begin{split}
  \begin{bmatrix}
    \Theta_1 \\
    \Theta_2
  \end{bmatrix}
  &+\frac{1}{\sqrt{2}}
  \begin{bmatrix}
    1&1\\
    1&-1
  \end{bmatrix}
  \begin{bmatrix}
    \alpha \\
    \alpha
  \end{bmatrix}\\[10pt]
  &-\begin{bmatrix}
    \sqrt{2}\alpha \\
    0
  \end{bmatrix}
  %\end{split}
\end{eqnarray}
```

$$\begin{bmatrix}\Theta_1\\ \Theta_2\end{bmatrix} + \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\alpha\\ \alpha\end{bmatrix} \tag{6.5}$$

$$- \begin{bmatrix}\sqrt{2}\alpha\\ 0\end{bmatrix} \tag{6.6}$$

Existen otros ambientes más para matemáticas. Si los quieres revisar, consulta alguno de los libros de LaTeX.

Espacios en fórmulas matemáticas

En el modo de matemáticas es muchas veces necesario acomodar de manera específica algunos símbolos, moviéndolos hacia la izquierda o hacia la derecha. En la tabla 6.18 vemos los distintos tipos de espacios que puedes tener. Están divididos entre positivos (hacia la derecha) y negativos (de retroceso).

Tabla 6.18 Comandos de espacio en modo matemático

Espacio positivo

Abr	Completo	Ejemplo		Normal
mu	\mspace{1mu}	x x	xx	
\,	\thinspace	x\,x	x x	xx
\:	\medspace	x\medspace x	x x	xx
\;	\thickspace	x\;x	x x	xx
	\enskip	x\enskip x	x x	xx
	\quad	x\quad x	x x	xx
	\qquad	x\qquad x	x x	xx

Espacio negativo

Abr	Completo	Ejemplo		Normal
\!	\negthinspace	x\!x	x x	xx
	\negmedspace	x\negmedspace x	xx	xx
	\negthickspace	x\negthickspace x	xx	xx

Es necesario aclarar que estos espacios se pueden utilizar también fuera del entorno matemático, en el modo de texto, excepto por \mspace{} que se usa únicamente en modo matemático.

Actividad 6.42 *Codifica en* LaTeX *la siguiente ecuación:*

$$G_{TC} = \frac{\frac{1}{N}\sum_{i=0}^{N-1}\sigma_i^2}{\left(\prod_{i=0}^{N-1}\sigma_i^2\right)^{\frac{1}{N}}} \tag{6.7}$$

Actividad 6.43 *Codifica en LaTeX la siguiente ecuación:*

$$s_n \neq s_{n-1} = s_{n-2} \quad M_1 = 0.4 \tag{6.8}$$
$$s_n \neq s_{n-1} \neq s_{n-2} \quad M_2 = 0.9 \tag{6.9}$$
$$s_n = s_{n-1} \neq s_{n-2} \quad M_3 = 1.5 \tag{6.10}$$
$$s_n = s_{n-1} = s_{n-2} \quad M_4 = 2.0 \tag{6.11}$$

Actividad 6.44 *Codifica en LaTeX la siguiente ecuación:*

$$\begin{aligned} \begin{bmatrix} x_{01} & x_{01} \\ x_{10} & x_{11} \end{bmatrix} &= \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} \theta_{00} & \theta_{01} \\ \theta_{10} & \theta_{11} \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} \theta_{00}+\theta_{01}+\theta 10+\theta_{11} & \theta_{00}-\theta_{01}+\theta 10-\theta_{11} \\ \theta_{00}+\theta_{01}-\theta 10-\theta_{11} & \theta_{00}-\theta_{01}-\theta 10+\theta_{11} \end{bmatrix} \\ &= \theta_{00}\alpha_{0,0}+\theta_{01}\alpha_{0,1}+\theta_{10}\alpha_{1,0}+\theta_{11}\alpha_{1,1} \end{aligned} \tag{6.12}$$

Marcos alrededor de expresiones

El comando \box{ ... } sirve para enmarcar una expresión. Veamos un ejemplo:

```
\[ \begin{pmatrix}
     \enskip\boxed{a}& b &c\\
     d&\boxed{e}&f\\
     g&h&\boxed{i}\enskip\
   \end{pmatrix}
\]
```

$$\begin{pmatrix} \boxed{a} & b & c \\ d & \boxed{e} & f \\ g & h & \boxed{i} \end{pmatrix}$$

Con esto damos por terminada la parte correspondiente a fórmulas matemáticas. Por supuesto que esto no agota el material relevante, pues por cuestiones de tiempo dejamos fuera muchísimas características y posibilidades que tiene LaTeX. Sin embargo, creemos que con esta introducción, podrás seguir adelante de manera independiente, de existir el interés en ti.

6.10. Imágenes y figuras

6.10.1. Tablas, figuras, etc.

Habrás notado a lo largo de este libro que aparecen figuras o tablas que se van numerando. Más aún, la numeración de las tablas, por ejemplo, es independiente de la de las

gráficas o, en general, figuras que queramos incluir. Esto se hace mediante lo que LaTeX llama *flotantes*. Les llama así porque le puedes indicar al compilador que las acomode no forzosamente donde aparecen, sino en la primera página en la que no ocasionen que se deje espacio en blanco innecesario. Por supuesto que también le puedes indicar que las coloque exactamente donde aparecen, sin importar el armado de las páginas que las preceden, o haciendo tú el armado a pie.
Las flotantes que vamos a revisar acá son únicamente dos:

- Figuras (figure)
- Tablas (table)

Ambas flotantes tienen los mismos argumentos, que tienen que ver fundamentalmente con dónde se desea que se les acomode; ambas proveen un espacio con la posibilidad de ponerles un título (\caption)[4]. Definen un ambiente donde los cambios de tipo de letra o tamaño serán efectivos únicamente dentro de ese ambiente. En general, el ambiente table se usa para material que se presenta en forma de tabla o lista, mientras que el de figure se usa para material gráfico. La sintaxis de estos ambientes es la siguiente:

Para figuras:

```
\begin{figure}[posicionador]
    Contenido
    \caption{Título de la figura}
\end{figure}
```

Para tablas (o cuadros):

```
\begin{table}[posicionador]
    Contenido
    \caption{Título de la tabla}
\end{table}
```

Si el *contenido de la figura o tabla* va antes de \caption, entonces el título aparecerá abajo de la figura. Si va después, esto es que \caption sea el primer renglón después de que empieza la figura o tabla, el título irá arriba. La posición de la figura o tabla puede ser:

Tabla 6.19 Posiciones posibles para los flotantes

letra	significado
t	*top:* Lo coloca en la parte superior de la primera página disponible.
b	*bottom:* Lo coloca en la parte inferior de la primera página donde quepa.
p	*page:* Lo coloca en una página de flotantes, la primera que pueda.
h	*here:* Lo coloca, si es que puede, en el punto donde aparece.
H	*Here:* Incondicionalmente donde aparece. Si es necesario, porque la figura no quepa en esa página, deja página en blanco y lo coloca en la siguiente página.

Continúa en la siguiente página

[4] También hay la posibilidad de marcarlas con una etiqueta para poder hacer referencias a ellas, pero dado lo poco del tiempo no entraremos en eso en esta ocasión.

Tabla 6.19 Posiciones posibles para los flotantes *Continúa de la página anterior*

letra	significado
!	*Try harder:* Intenta de manera más firme acomodar la figura o tabla donde se especifica con los posicionadores que siguen a !.

Para acomodar una figura o tabla procedes a dar tus preferencias, que son combinaciones de las primeras tres letras. LaTeX tratará de acomodar la figura, colocándola en la misma página o posponiéndola, procurando dejar poco espacio en blanco. Si no se especifica, el valor por omisión es tbp. LaTeX podría no acomodar las figuras como le indicas porque tiene parámetros que le indican, por ejemplo, cuántas figuras pueden ir en una misma página o qué porcentaje del total de la página pueden ocupar. No entraremos en estos detalles por falta de tiempo. En la mayoría de los casos, y sobre todo en un principio, puedes trabajar con los valores por omisión, excepto posiblemente para el posicionamiento. Te recomendamos, en general, poner H y si no se forma "bonito", proceder a acomodar la figura o tabla a pie. Veamos algunos ejemplos sencillos, con el código de LaTeX que los produce.

Tabla 6.20 Conversión de grados Farenheit a Centígrados

Fahr	Cent
32	0
40	4.5
80	27

```
\begin{table}[!h]
  \caption{Conversión de Farenheit
      a Centígrados}
  \begin{tabular}[t]{|r|r|}
    \hline
    \bf Fahr & \bf Cent\\\hline
    32 & 0\\
    40 & 4.5\\
    80 & 27\\\hline
  \end{tabular}
\end{table}
```

Puedes cambiar varios aspectos de esta tabla. Si, como ya dijimos, quieres que la acomode en la parte superior de una página, conseguirías que coloque primero el texto que dimos como código y después la tabla misma. Sin embargo, es poco recomendable dar una única posición como opción, porque entonces si LaTeX no puede acomodar a la tabla o figura en la siguiente página, lo más seguro es que ya no la coloque en ningún lugar y la perdamos. Observa cómo la tabla 6.21 que estás colocando inmediatamente después de estas líneas, queda acomodada de manera aparentemente "arbitraria" (está acomodada utilizando sólo la mitad del ancho de la página).

Tabla 6.21 Conversión de Farenheit a Centígrados

Fahr	Cent
32	0
40	4.5
80	27

La tabla 6.21 fue generada con el siguiente encabezado: \begin{table}[H]. Las líneas

```
\begin{center}
  ...
\end{center}
```

se pueden usar para centrar cualquier párrafo, línea, figura o tabla.

Cuando dejas que LaTeX acomode a la tabla o figura donde mejor pueda – en ausencia de H – deberá haber un renglón en blanco antes y después de la tabla, para evitar que se encimen o mezclen renglones que pusimos en párrafos separados.

En una flotante de tabla puede colocar uno o más array, cuidando nada más de declarar un ambiente matemático adentro.

Actividad 6.45 *Codifica en LaTeX la tabla 6.22 en la siguiente página. Pon el posicionamiento para que aparezca exactamente donde se tecleó.*

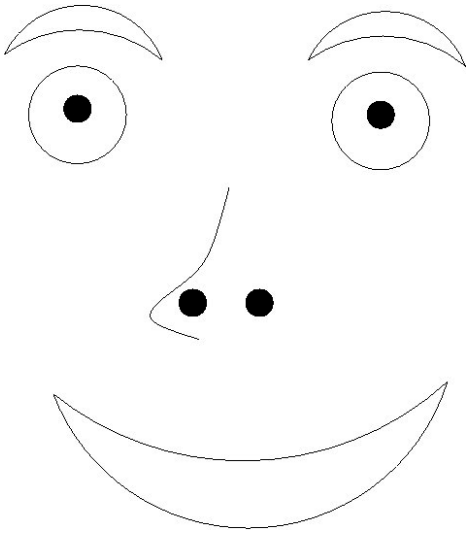
Tabla 6.22 Representaciones numéricas

Decimal	Binario	Octal	Hexadecimal
1	1	1	1
2	10	2	2
3	11	3	3
5	101	5	5
7	111	7	7
8	1000	10	8
15	1111	17	F
16	10000	20	10

Todo lo que dijimos de las tablas se aplica a las figuras. El contador de tablas es independiente del de figuras. LaTeX provee un mecanismo sencillo para incluir imágenes o figuras generadas desde fuera y que estén codificadas en postscript o imágenes .png. Esto se hace con el comando \includegraphics {...} que toma como argumento un archivo en el que se encuentre la imagen o figura que se desea incluir. En el archivo dummy.ps se encuentra una figura (hecha en xfig) que deseamos incorporar en nuestro documento. Para hacerlo, tenemos que incluir el uso del paquete graphics en el preámbulo del documento:

\usepackage{graphics}. Este paquete cuenta con los mecanismos para poder manipular la figura:

Figura 6.1 Gráfico introducido con `includegraphics`

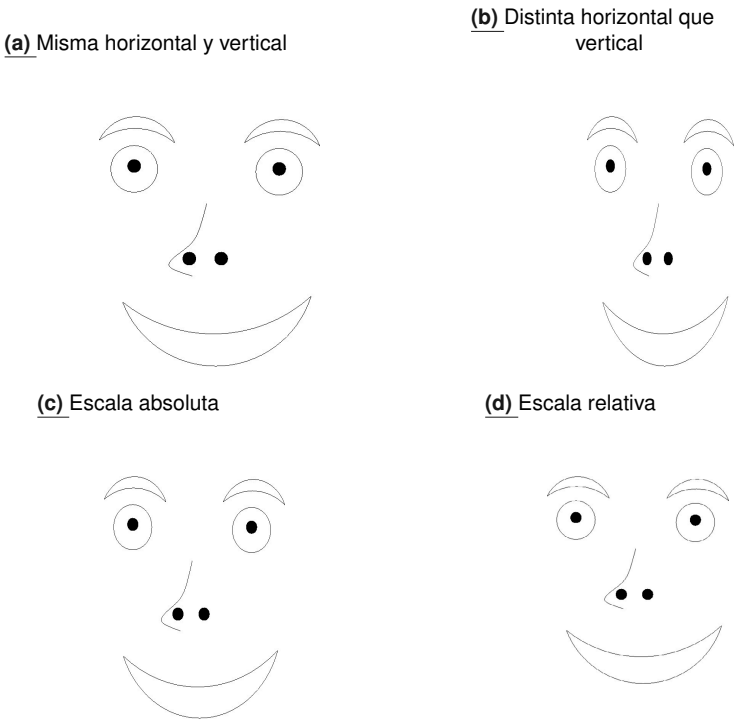


que fue incluida con el siguiente código:

```
\begin{figure}[H]
  \label{fig:1}
  \caption{Gráfico introducido con \texttt{includegraphics}}
  \includegraphics[width=.5\textwidth]{latex/dummy}
\end{figure}
```

El paquete graphics contiene además comandos que permiten girar y escalar una imagen, párrafo, tabla, símbolo, etc. Tienes dos maneras de escalar un objeto: de manera relativa, diciendo el factor de escala, como se muestra en la figura 6.3a (\scalebox{.25}); o indicando la proporción en la escala horizontal entre llaves y con la escala vertical entre corchetes: `\scalebox{.2}[.3]`, como en 6.3b (\scalebox{.25}[.5]). También puedes dar medidas absolutas, como en la figura 6.3c (\resizebox{8cm}{2cm}) o bien, utilizar una escala relativa, donde omitimos el factor vertical, y se asumirá el mismo que el horizontal, por ejemplo 6.3d (\resizebox{2cm}{!}). En la escala absoluta se deben dar las dos medidas. Cualquiera de ellas puede ser sustituida por !.

Figura 6.2 Escalando objetos con graphics.



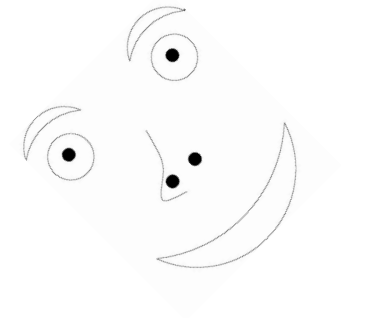
Otras dos posibilidades que tienes con objetos es la de rotarlos o "espejearlos". En la figura 6.3 puedes ver rotaciones de 45 grados 6.4a (\rotatebox{45}), de 180 6.4b (\rotatebox{180}) y una figura y su reflejo 6.4c (\reflectbox). En cada una de ellas se marca la línea base como una línea punteada, ya que algunas de las figuras (como la rotada) fueron "subidas" para que no quedara desproporcionada la figura.

Se puede observar también que estas figuras pueden aparecer mezcladas con el texto, como aquí  sin necesidad de incluir en una figura o tabla, donde lo único que se hizo fue insertar \reflectbox{\scalebox{.1}{\includegraphics{dummy.ps}}} entre las palabras "aquí" y "sin".

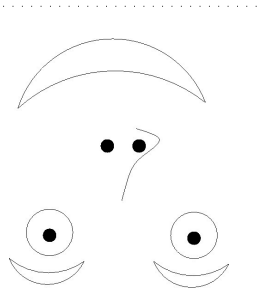
Es importante insistir en que estas operaciones de escalar, rotar y reflejar las puedes aplicar a cualquier objeto: texto, fórmulas matemáticas (siempre y cuando estén en modo matemático .. o \(.. \)), tablas completas, figuras, etc.

Figura 6.3 Rotaciones y reflejos con `graphics`

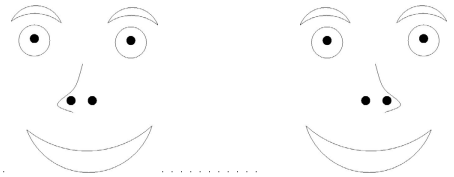
(a) Rotación de 45



(b) Rotación de 180



(c) Figura y su reflejo



Si por ejemplo quieres el reflejo de algo pero a lo largo del eje horizontal, lo puedes conseguir con el código que sigue

```
\scalebox{2}{reflejo horizontal\rotatebox{180}{%
                    \reflectbox{reflejo horizontal}}}
```

y que produce

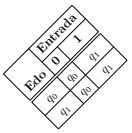
reflejo horizontal

Debes notar que lo que pasa al texto “abajo” de la línea base es la rotada, ya que toma la esquina inferior izquierda como punto de rotación.

Actividad 6.46 *Codifica en* LaTeX *lo siguiente:*

Tabla 6.23 Comparación de tamaños

Edo	Entrada	
	0	**1**
q_0	q_0	q_1
q_1	q_0	q_1



Actividad 6.47 *Codifica en LaTeX la siguiente expresión:*

$$\alpha = b^2 - 4ac \qquad \mathfrak{A} = \mathfrak{p}_{\mathfrak{Z}} - \sqrt{\mathfrak{a}\mathfrak{C}}$$

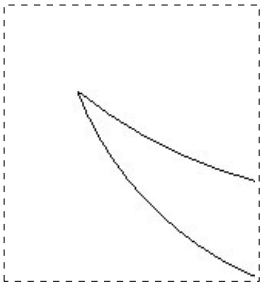
Actividad 6.48 *Importa a un archivo de LaTeX que use el paquete graphics alguna de las gráficas que elaboraste con xfig (la del árbol genealógico, por ejemplo). La debes tener guardada en tu cuenta, por lo que si no la exportaste, abre otra ventana y ejecuta xfig para exportarla en postscript.*

6.10.2. Cortando figuras

Puedes mostrar únicamente parte de una figura utilizando el modo \includegraphics* del comando (agregando un asterisco al nombre del comando) y dándole parámetros opcionales entre corchetes. La primera pareja le indica la posición de la esquina inferior izquierda y la segunda pareja la posición de la esquina superior derecha. Puedes observar el resultado en la figura 6.4.

Figura 6.4 Tomando pedazos de figuras

```
includegraphics*[100pt,130pt][220pt,260pt]\{dummy.ps\}
```



En la figura 6.4 colocamos un marco alrededor del "recorte" para hacerlo más claro. Ese marco no aparece como resultado del comando.

6.10.3. Colores

En el contexto de generar archivos pdf, utilizaremos para colorear dos paquetes, color y tikz. Aunque el primero es claro su objetivo, el segundo, tikz, es un paquete de LaTeX mucho muy poderoso para graficación y que incluye un sistema de colores – además compila tanto con pdflatex como con latex – . Está incluido dentro del paquete pgf aunque aporta una interfaz más sencilla que la que soporta pgf. Es tan grande y poderoso que está dividido en varios paquetes para que elijas únicamente aquellos que va a usar. En la tabla 6.24 damos una lista de los que vamos a requerir, por lo pronto, para generar colores. Cabe aclarar que para utilizar tikz es necesario cargar también algunos paquetes que nos sirven como soporte, que listamos también a continuación[5].

Tabla 6.24 Paquetes de pgf para colores y dibujos

Nombre	Uso	Descripción
tikz	\usepackage{tikz}	Éste es el paquete más importante de pgf, ya que da una interfaz sencilla de usar para graficar. Incluye también un sistema para colores que permite combinar los colores principales.
xkeyval	\usepackage{xkeyval}	Permite asociar nombres de parámetros del paquete con los valores correspondientes. Se requiere también para otros paquetes.
pifont	usepackage{pifont}	Se encarga del manejo de bajo nivel para las gráficas. Lo requiere el paquete tikz.
color	\usepackage{color}	Proporciona una interfaz para poder crear nuevos colores y un sistema para hacerlo.

Veamos entonces algunas manipulaciones que involucran color [6]. Por ejemplo, si quieres cambiar el color de lo que estamos haciendo tenemos el comando

```
\color{red}
```

y desde ese momento en adelante, en el rango del ambiente en el que esté, todo se escribirá en rojo. Los colores básicos con los que contamos se encuentran en la tabla 6.25.

.

[5] Todos estos paquetes pueden obtenerse gratuitamente de la dirección
http://www.tex.ac.uk/tex-archive/help/Catalogue/catalogue.html
que es una excelente fuente de paquetes junto con su documentación

[6] Si tienes una impresora en blanco y negro, esto aparecerá en distintos tonos de grises, pero en la pantalla lo verás tal cual

Tabla 6.25 Colores básicos con los que se cuenta

Código	Resultado
\color{black}black	black
\colorbox{gray}{\color{white}white}	white
\color{red}red	red
\color{yellow}yellow	yellow
\color{cyan}cyan	cyan
\color{blue}blue	blue
\color{green}green	green
\color{magenta}magenta	magenta

y además tenemos los siguientes tonos de grises predefinidos:

\color{black}black	black
\color{darkgray}darkgray	darkgray
\color{gray}gray	gray
\color{ lightgray } lightgray	lightgray
\color{white}white	

Como colocamos todo en un tabular, el efecto del cambio de color es únicamente dentro de la celda en que está escrito.

Tenemos la posibilidad de cambiar únicamente el texto deseado con el siguiente código:

```
Vamos cambiar de \textcolor{red}{rojo}
a color \textcolor{cyan}{azul}
```

Vamos cambiar de rojo a color azul

El paquete color también nos ofrece la posibilidad de conformar cajas con el fondo de ciertos colores para realzar algunos textos:

```
\colorbox{red}{\textcolor{blue}{%
            Esta es una prueba}}
```

Esta es una prueba

Actividad 6.49 *Teclea en un archivo fuente de LaTeX donde uses el paquete* color *y* graphics*, las líneas correspondientes a color y ve en la pantalla cómo quedan (desafortunadamente la impresión no muestra el color, sino únicamente distintos niveles de grises, pero en la pantalla sí lo puedes apreciar).*

6.10.4. Dibujando objetos geométricos

Hay varios paquetes que te permiten dibujar objetos geométricos como líneas, círculos, arcos, etc. Utilizaremos acá TikZ – el subpaquete shapes y arrows –, que nos ofrece una gran versatilidad. No veremos acá más que algunos objetos, ya que este paquete es sumamente extenso.

Para poder usar comandos de tikz tenemos que estar dentro de un ambiente de este paquete. tikz tiene dos tipos de ambiente, el de línea y el de párrafo:

```
\tikz<[modificadores]>{<comandos>}
```
Ambiente de línea.

```
\begin{tikzpicture}<[modificadores]>
  ...
\end{tikzpicture}
```
Ambiente de párrafo.

Los modificadores van entre corchetes y tienen que ver con aspectos generales de las figuras que deseamos construir. Ambos ambientes presuponen un tamaño, ya sea de líneas o de párrafo, dado en la unidad de medida que se especifique. La unidad de medida por omisión es en centímetros, pero se puede cambiar. Prácticamente todo lo que deseemos dibujar va a tener implícitas coordenadas y longitudes, que estarán dadas implícitamente en la unidad de medida.

Para denotar a estas unidades de medida, tikz utiliza vectores en el eje de las x, de las y's y de las z's. Es conveniente asegurarnos con qué unidad de medida estamos trabajando. La puedes definir en milímetros, pulgadas, centímetros o puntos. Preferimos trabajar con puntos, ya que ésta es una unidad que se usa en muchos otros lugares. Para establecer al punto como unidad de medida en los comandos de tikz utilizamos el siguiente modificador:

```
\tikz[x=1pt,y=1pt,z=1pt]{ ... }

\begin{tikzpicture}[x=1pt,y=1pt,z=1pt]
  ...
\end{tikzpicture}
```

y a partir de ese momento y dentro del ambiente declarado, la unidad de medida será 1 punto (1pt). Si se declara otro ambiente en otro lugar, habrá que hacer lo mismo, o bien trabajar en centímetros.

6.10.5. Líneas

El comando que permite dibujar líneas es \draw y hay que darle como argumentos las coordenadas de los puntos por los que quieres que pase. El formato más simple del comando es el siguiente:

```
\draw (<coord inicial>) -- (<siguiente coordenada>) ... ;
```

donde los puntos sucesivos indican que puedes poner tantas coordenadas como desees. Las coordenadas se pueden dar como puntos relativos con la unidad de longitud del ambiente; o bien puede ser algún objeto etiquetado[7]; también puede aparecer allí una coordenada con una unidad de medida específica. Si se trata de coordenadas, consiste de una pareja con los elementos separados entre sí por coma, donde el primer elemento de la pareja es el desplazamiento sobre el eje x y el segundo sobre el eje y. También es importante notar que los comandos de tikz en su mayoría terminan con punto y coma (;).

Es importante mencionar que el texto seguirá inmediatamente después del punto que el comando considere que requiere. Veamos algunos ejemplos:

```
\tikz[x=1pt,y=1pt]\draw(0,0) -- (20,30) -- (30,0)
```

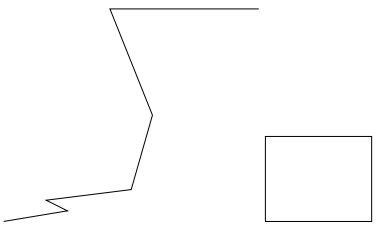
. Nota cómo el renglón se acomoda verticalmente para dar lugar al trazo y el texto continúa a partir del final del espacio ocupado. Si deseamos que el texto se acomode encima del dibujo, tenemos que hacerlo a pie recorriéndonos hacia arriba y hacia la izquierda con comandos de \vspace y \hspace.

```
\vspace*{-30 pt} %El máximo de la segunda coordenada y el cambio
                 %de línea.

\tikz[x=1pt,y=1pt]{\draw(0,0)--(20,30)--(30,0);}
\hspace*{-30pt}%
\tikz[x=1pt,y=1pt]{\draw(0,0)--(20,30)--(30,0);}%
    y las líneas se acomoda como se está observando
```

y las líneas se acomodan como se está observando.

Veamos más ejemplos de algunas figuras. Una manera de dejar espacio vertical para acomodar algunas de las figuras es con \vspace. Observa:

```
\tikz[x=1pt,y=1pt]{\draw (0,0) -- (30,5) -- (20,10) -- (60,15)%
        -- (70,50) -- (50,100) -- (120,100);}
\tikz[x=1pt,y=1pt]{\draw (140,40) -- (190,40) -- (190,80)
        -- (140,80) -- (140,40);}
```



Puedes cambiar el ancho de la línea también con un modificador, line width y el tamaño de la línea se puede dar en cualquier unidad de medida:

[7] Veremos más adelante cómo etiquetar objetos para usarlos en un dibujo.

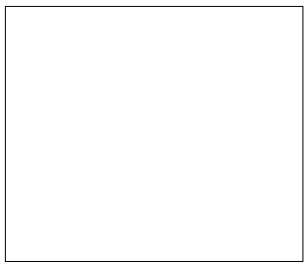
```
\begin{tikzpicture}[x=1pt,y=1pt]
  \draw[line width=2pt] (0,0) -- (30,20);
  \draw[line width=.13in] (30,20)-- (50,0);
  \draw[line width=.15cm] (50,0)-- (70,20);
  \draw (70,20)-- (90,0);
  \draw[line width=1pt] (90,0)-- (110,20);
\end{tikzpicture}
```

También se puede dar el ancho de línea para todo un ambiente, poniéndolo de modificador de ambiente:

```
\begin{tikzpicture}[x=1pt,y=1pt,%
                    line width=2pt]
  \draw (0,0) -- (30,20) -- (30,20)%
              -- (50,0) -- (50,0)-- (70,20)%
              -- (70,20)-- (90,0) -- (90,0)%
              -- (110,20);
\end{tikzpicture}
```

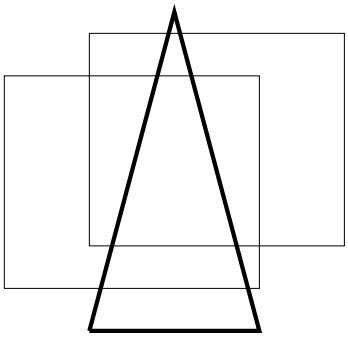
El comando \draw es fundamental en tikz ya que servirá también para acomodar objetos y dibujar figuras. Por ejemplo, para dibujar un rectángulo combinamos \draw con esa forma:

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \draw (0,0) rectangle (140,120);
\end{tikzpicture}
```



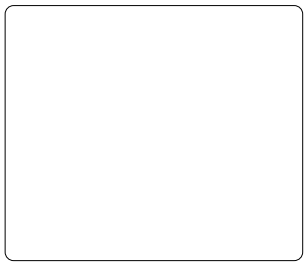
Siempre que introduces un ambiente tikz, debes pensar que reservas un espacio rectangular dado por lo que dibujaste. Ese espacio logra contener todos los objetos que hayas acomodado o dibujado en ese ambiente. El (0,0) es la esquina inferior izquierda (aunque esto lo puedes modificar) y la esquina superior derecha está dada por lo que hayas hecho dentro del ambiente.

Actividad 6.50 *Escribe el código necesario para que se dibuje lo que sigue (el tamaño relativo es lo que importa).*

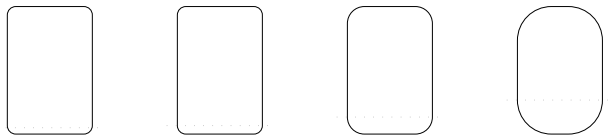


Como ya mencionamos, tikz provee varias figuras geométricas con las que puedes trabajar usando la biblioteca shapes de este paquete. Y como las figuras son cierto tipo de objetos, puedes acomodarlos también en la retícula mediante draw. Además del rectángulo, contamos con círculos (*circle*), diamantes (*diamond*) y elipses (*ellipse*). El formato para cada una de estas figuras es el siguiente: una vez que se ubica el punto donde se va a colocar la figura mediante draw, cada una de las figuras tiene sus propios parámetros. Quisiéramos antes mencionar un modificador más que altera la forma de los puntos de quiebre para redondearlos y que es rounded corners. Observa el efecto que tiene en el rectángulo que hicimos arriba:

```
\begin{tikzpicture}%[x=1pt,y=1pt]
  \draw[x=1pt,y=1pt,rounded corners]
        (0,0) rectangle (140,120);
\end{tikzpicture}
```



Puedes jugar con qué tan redondeadas estén las esquinas, agregándole el redondeo en términos de una unidad de medida a continuación del signo =. La medida se refiere a la distancia de lo que sería la esquina sin redondear a la redondeada, medido verticalmente.

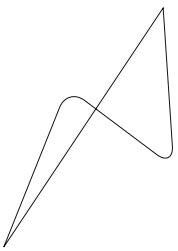


que se consiguió con el siguiente código:

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \draw[rounded corners] (0,60) rectangle (40,120);
  \draw[rounded corners=4pt] (80,60) rectangle (120,120);
  \draw[rounded corners=8pt] (160,60) rectangle (200,120);
  \draw[rounded corners=16pt] (240,60) rectangle (280,120);
\end{tikzpicture}
```

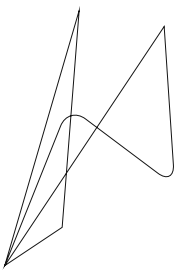
Otros de los modificadores que puedes poner a \draw son el color de las mismas – blue, red, yellow, cyan, green, magenta –; el tipo de línea – [loosely|densely]{solid|dotted|dashed} –. Estos últimos parámetros no llevan signo = o identificador cuando aparecen en el contexto de un comando \draw y puedes usarlos con figuras, líneas o arcos. Hay que mencionar también que el redondeo de esquinas puede aplicarse únicamente a un segmento de la trayectoria pintada, si se coloca antes de las coordenadas a partir de la cual debe aparecer. Veamos un pequeño ejemplo:

```
\begin{tikzpicture}[x=1pt,y=1.5pt]
  \draw (0,0)[blue,rounded corners=10pt]
        -- (30,50) -- (80,25)
        [sharp corners] -- (75, 75)
        -- (0,0);
\end{tikzpicture}
```



Como es muy común querer terminar en la coordenada en la que se empezó (cerrar un polígono), tikz proporciona la meta-coordenada cycle que, donde aparezca, regresa la primer punto dado en la trayectoria, como por ejemplo en el siguiente dibujo, en que ambas veces regresa la coordenada (0,0).

```
\begin{tikzpicture}[x=1pt,y=1.5pt]
  \draw (0,0)[blue,rounded corners=10pt]
        -- (30,50) -- (80,25)
        [sharp corners] -- (75, 75)
        -- cycle -- (27,12) -- (35,80) -- cycle;
\end{tikzpicture}
```



En la tabla 6.26, <dim> se refiere siempre a una cantidad de medida, como mm, cm, in, pt, em, ex, mientras que <num> es un número que puede o no ser un entero, y en algunos casos puede ser negativo. Estos modificadores pueden aparecer en el ambiente o en el comando.

Tabla 6.26 Modificadores para el trazo de líneas y figuras

Parámetro	Descripción	Por omisión
color=<color>	Especifica el color en que se va a pintar la línea. Puede aparecer directamente el color nada más.	color: black
x=<dim> y=<dim> z=<dim>	Unidades para cada uno de los ejes. Son opcionales cualquier subconjunto de ellos.	dim: 1cm
line width=<dim> very thick thick very thin thin	Especifica el grosor de las líneas que se dibujan (ver tabla 6.27.	dim: 1pt
pattern= patrón	Rellena la figura o el polígono con ese patrón[8].	estilo: solid
pattern color=<color>	Decide el color de los trazos que componen al patrón.	black
rounded corners rounded corners= dim sharp corners	Indica si los cambios de dirección se hacen redondeados o no, y si sí con cuánto redondeo.	sharp corners
fill fill = $< color >$	Indica que la figura deberá ser sólida, rellenada. Si se indica el color, el relleno deberá ser de ese $< color >$ – ver figura 6.5 donde se ejemplifica esta opción – .	no
dashed loosely dashed densely dashed dotted loosely dotted densely dotted solid	Da el tipo o estilo de lineas que se van a dibujar (ver tabla 6.27).	solid

[8] Para ver los tipos de patrones disponibles, consultar el manual de TikZ.

Tabla 6.27 Opciones de grosor y estilos de líneas

very thin	loosely dotted	dotted	densely dotted
thin	loosely dotted	dotted	densely dotted
normal	loosely dotted	dotted	densely dotted
thick	loosely dotted	dotted	densely dotted
very thick	loosely dotted	dotted	densely dotted
very thin	loosely dashed	dashed	densely dashed
thin	loosely dashed	dashed	densely dashed
normal	loosely dashed	dashed	densely dashed
thick	loosely dashed	dashed	densely dashed
very thick	loosely dashed	dashed	densely dashed

Figura 6.5 Ejemplo de opciones para dibujar

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \draw[fill] (0,0)[blue,rounded corners=10pt]
          -- (30,50) -- (80,25)[sharp corners]
          -- (75, 75)-- cycle -- (27,12)
          -- (35,80) -- cycle;
\end{tikzpicture}
```

Pasemos ya a ver las distintas figuras que puedes dibujar con el comando \draw. Las mostramos en la tabla 6.28.

Tabla 6.28 Ejemplos de figuras en TikZ

Comando	Figura
\tikz [x=1pt,y=1pt]{\draw(25,10) ellipse (20pt and 10pt);}	
Como puedes observar, la coordenada dada a continuación de \draw da el centro de la elipse, mientras que la primera dimensión da el diámetro horizontal y la segunda el vertical.	
\tikz [x=1pt,y=1pt]{\draw (25,10) circle (12.5pt);}	
En este caso, también la primera coordenada nos da el centro del círculo, mientras que la dimensión nos da el diámetro.	
\tikz [x=1pt,y=1pt]{\draw (25,10) arc (90:0:25pt);}	
Igual que antes, la primera coordenada te da el centro del círculo completo, aunque únicamente vayas a pintar un arco. Los tres números entre paréntesis tienen el siguiente significado: puedes ver al circulo en el eje coordenado con el centro en (0,0); el primer numero da el ángulo en el que empieza el arco; el segundo da el ángulo en el que termina, moviéndose siempre en el sentido de las manecillas del reloj; el tercer numero da el radio del círculo del que el arco forma parte.	
\tikz [x=1pt,y=1pt]{\draw (5,0) .. controls (15,15) and (30,15) .. (40,0);}	
Esta construcción te permite dibujar curvas que no son forzosamente arcos de círculos. Las coordenadas de control representan puntos para dibujar el bezier: entre el origen y el primer punto la recta que los une es tangente a la curva; similarmente para el segundo punto.	

Además de las variaciones en la forma y el tipo de línea, también puedes pedir que la línea sea doble, como lo ejemplificamos en la tabla 6.29.

Tabla 6.29 Líneas dobles

	\draw[very thick,double](0,0) -- (50,0);
	\draw[thin,double](0,0) -- (50,0);
	\draw[very thick,double distance=2pt](0,0) -- (50,0);
	\draw[thin,double distance=2pt](0,0) -- (50,0);

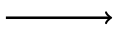
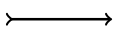
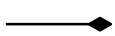
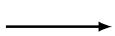
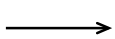
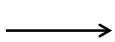
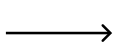
Realmente el número de variaciones sobre estos cuatro simples comandos es muy grande, así que te recomendamos consultar el manual de TikZ para poder usar todas ellas.

En el caso de las líneas y los polígonos, puedes tener el último segmento tome o no la forma de flecha. Los tipos de flechas que tenemos disponibles se encuentran en las tablas 6.30 y se consiguen poniendo como modificador de la línea el tipo de flecha que deseas. Vienen en el paquete de biblioteca de flechas de TikZ que se invoca con la siguiente línea en el preámbulo del documento.

```
\usetikzlibrary{arrows}
```

Algunas puntas de flechas vienen por omisión en TikZ. Los estilos del cuerpo de la flecha pueden ser cualesquiera de los listados anteriormente, por lo que los ejemplos los daremos con línea sólida para ejemplificar nada más las puntas. Los distintos tipos de puntas de flechas se encuentran en la tabla 6.30.

Tabla 6.30 Tipos de puntas de flechas en TikZ

	\draw [arrows=−>] (0,0) −− (50,0);
	\draw [arrows=>−>] (0,0) −− (50,0);
	\draw [arrows=|−>] (0,0) −− (50,0);
	\draw [arrows=|−diamond] (0,0) −− (50,0);
	\draw [arrows=−latex] (0,0) −− (50,0);
	\draw [arrows=−angle 45] (0,0) −− (50,0);
	\draw [arrows=−angle 60] (0,0) −− (50,0);
	\draw [arrows=−angle 90] (0,0) −− (50,0);

Como puedes observar hay bastante juego con el tipo de flecha y el origen del segmento. Para más información, consulta el manual de TikZ.

Otro aspecto bonito de TikZ es que puedes dar distintos patrones para el relleno de figuras. Esto se logra cargando el paquete de biblioteca patterns en el preámbulo del documento.

```
\usetikzlibrary{patterns}
```

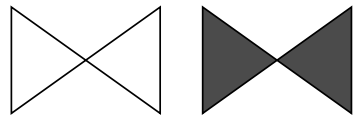
En el manual de TikZ viene una lista de patrones interesantes que puedes desear usar.

En las tablas 6.31 y 6.32 daremos ejemplos de distintos objetos geométricos que puedes pintar con TikZ. Es importante que recuerdes que las coordenadas que se dan suponen que el comando inicia en (0,0), y a partir de ahí se calcula la posición del dibujo. Sin embargo, el acomodo de las figuras se hizo con mecanismos que se darán a continuación. Asimismo se hicieron algunos trucos para que la altura del renglón fuera la adecuada para cada figura. Deberás tomar en cuenta la sección que sigue donde enseñaremos a acomodar de mejor manera las figuras.

Tabla 6.31 Figuras geométricas que se pueden lograr en TikZ

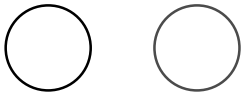
```
\begin{tikzpicture}[x=1pt,y=1pt,line width=3pt]%
\useasboundingbox (0,0) rectangle
                          (140,60);
    \draw[-latex,rounded corners]%
          (100,50) -- (0,20) -- (50,0);
\end{tikzpicture}
```

```
\begin{tikzpicture}[x=1pt,y=3pt]
    \useasboundingbox (0,0)
                rectangle (170,70);
    \draw[thick] (0,60) -- (70,10)
                -- (70,60) -- (0,10)
                -- cycle;
    \draw[thick,fill=red] (90,60)
                -- (160,10) -- (160,60)
                -- (90,10) -- cycle;
\end{tikzpicture}
```



```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0) rectangle (170,70);
  \draw[fill] (10,10) rectangle (80,60);
  \draw[fill=white] (20,20) rectangle (40,40);
\end{tikzpicture}
```

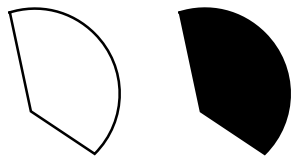
```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,-10) rectangle (140,60);
  \draw[very thick] (30,30) circle (20pt);
  \draw[very thick,red] (100,30) circle (20pt);
\end{tikzpicture}
```



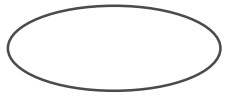
Continúa en la página siguiente

Tabla 6.31 Figuras geométricas que se pueden lograr en TikZ *Continúa de la página anterior*

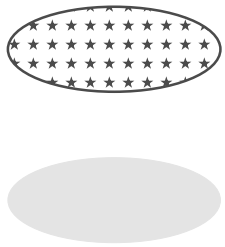
```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0)
                    rectangle (140,70);
  \draw[very thick] (20,38) -- (10,0)
       -- (50,0) arc (0:76:40pt);
  \draw[fill] (100,38) -- (90,0)
       -- (130,0) arc (0:76:40pt);
\end{tikzpicture}
```



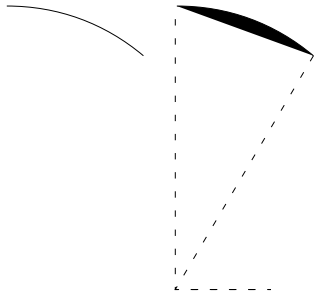
```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0)
                    rectangle (140,70);
  \draw[very thick,red] (70,30)
               ellipse (50pt and 20pt);
\end{tikzpicture}
```



```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0)
                    rectangle (140,70);
  \draw[very thick,step=5mm,red,
          pattern=fivepoint stars,
                  pattern color=red] \
        (70,30) ellipse (50pt and 20pt);
\end{tikzpicture}
```



```
\begin{tikzpicture}[x=1pt,y=1pt]
   \useasboundingbox (0,-50)
                rectangle (170,100);
   \draw (80,60) arc (50:90:100pt);
   \draw[fill] (160,60)
                   arc (50:90:100pt);
   \draw[very thin,loosely dashed]
           (160,60) -- (95,-50)
        -- (140,-50) -- (95,-50)
                   -- (95,83);
\end{tikzpicture}
```



Otro aspecto que es frecuente que quieras tener en un documento es la graficación de funciones. TikZ proporciona algunas funciones típicas que puedes querer graficar, como lo son la parábola, el círculo, seno y coseno. Para otras funciones se tendrán que preparar las tablas en un archivo aparte,

pero se pueden graficar. Veamos primero estas cuatro en la tabla 6.32.

Tabla 6.32 Graficación de funciones comunes en TikZ

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0)
              rectangle (140,70);
  \draw[thick] (45,50)
               arc (60:360:20pt);
  \draw[thick,fill] (115,50)
               arc (60:360:20pt);
\end{tikzpicture}
```

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0)
                 rectangle (100,50);
  \draw (0,0)
        parabola[parabola height=30pt]
                                (30,0);
  \draw(50,30)
        parabola[parabola height=-30pt]
                                (90,30);
\end{tikzpicture}
```

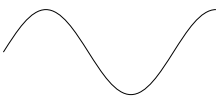
Puedes obtener pedazos de parábola si das como segunda coordenada algo que no sea la imagen del punto de origen, o bien con modificadores:

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0)
                rectangle (140,60);
  \draw (0,0) parabola (30,45);
  \draw (50,0)
              parabola[bend at end]
                      (80,45);
  \draw (100,0)
              parabola bend
              (122.5,52.25) (130,45);
\end{tikzpicture}
```

Continúa en la siguiente página

Tabla 6.32 Graficación de funciones comunes en TikZ *Continúa de la página anterior*

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,-30)
                rectangle (140,40);
  \draw (0,0) sin (20,20)
              sin (60,-20)
              cos (80,0)
              sin (100,20);
\end{tikzpicture}
```



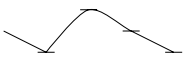
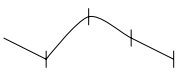
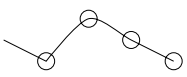
Tanto el seno como el coseno únicamente están definidos para el intervalo $(0, \frac{\pi}{2})$, por lo que es necesario combinar ambas funciones para pintar la trayectoria completa

Otro mecanismo también muy importante para graficar funciones es el de poder darle una lista de puntos y que vaya marcándolos con algún símbolo. Para marcar los puntos con símbolos requieres de la biblioteca plotmarks mediante el comando

```
\usetikzlibrary{plotmarks}
```

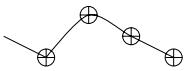
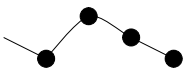
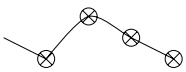
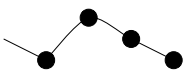
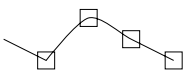
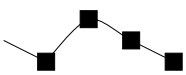
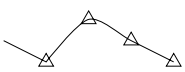
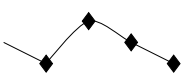
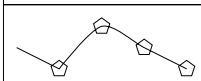
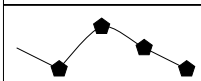
Una vez cargada la biblioteca de marcas, puedes elegir entre las que se encuentran en la tabla 6.33.

Tabla 6.33 Símbolos para marcar curvas que se grafican

Marca	Identificación	Descripción
	mark=–	guiones
	mark=|	barras verticales
	mark=o	círculos huecos
	mark=asterisk	asteriscos
	mark=star	estrellas

Continúa en la página siguiente

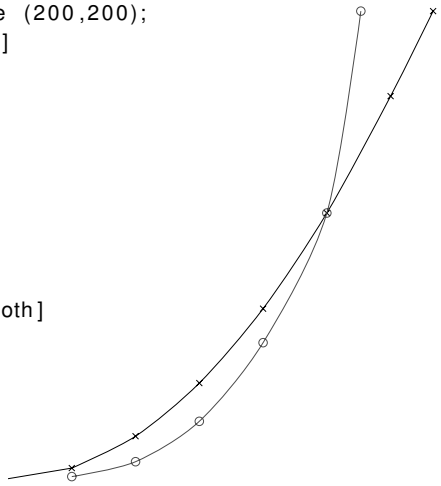
Tabla 6.33 Símbolos para marcar curvas que se grafican *Continúa de la página anterior*

Marca	Identificación	Descripción
	mark=oplus	círculos con suma
	mark=oplus*	círculos con suma
	mark=otimes	círculos con producto
	mark=otimes*	círculos con producto
	mark=square	cuadrado
	mark=square*	cuadrado sólido
	mark=triangle	triángulo
	mark=triangle*	triángulo sólido
	mark=diamond	diamante
	mark=diamond*	diamante sólido
	mark=pentagon	pentágono
	mark=pentagon*	pentágono sólido

Para graficar funciones usamos también el comando \draw con modificadores y parámetros.

Figura 6.6 Graficación de funciones arbitrarias

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,-2) rectangle (200,200);
  \draw (0,0) -- plot[mark=x,smooth]
    coordinates{%
      (30,5)
      (60,20)
      (90,45)
      (120,80)
      (150,125)
      (180,180)
      (200,220)};
  \draw[red] (0,0) plot[mark=o, smooth]
    coordinates {(30,1)
      (60,8)
      (90,27)
      (120,64)
      (150,125)
      (166,220)
    };
\end{tikzpicture}
```



Marcas un punto inicial y a continuación dices que quieres graficar una función (plot). A esta última opción le aplicas modificadores, como el símbolo para las marcas, posiblemente si quieres o no la curva suave (smooth) y el color de las marcas, que puede no ser el mismo que el de la curva.

Sin embargo, las gráficas que pintamos en la figura 6.6 son realmente muy pobres. Quisiéramos poder dibujar los ejes cartesianos y colocar indicaciones respecto a los valores en estos ejes. Tenemos dos formas de hacerlo.

- Usando el comando \draw con el argumento grid, cuya sintaxis es:
  \draw <*esq. inf. izquierda*> grid [<*modificadores*>] <*esq. sup. derecha*>;
  La parte del [< *intervalo* >] corresponde a que tan cerrada quieres la rejilla para las coordenadas. Sin embargo, con esta opción no tenemos todavía los ejes o las anotaciones en los mismos. Veamos cómo queda en la figura 6.7.
  Sin embargo, con esta opción debemos poner los ejes y las marcas a pie. Los ejes no es ningún problema pues lo hacemos simplemente utilizando \draw, como se muestra en la figura 6.8.
- La otra forma de pintar la retícula es "a pie" . Afortunadamente, como el pintar las líneas se trata de una tarea repetitiva, puedes usar la construcción \foreach que proporciona TikZ. Esta es una construcción muy poderosa que funciona como sustitución de macros, por lo que se puede introducir en casi cualquier enunciado. Daremos el ejemplo en la figura 6.9 y después analizaremos su formato.

Figura 6.7 Impresión de la retícula

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,-2)
              rectangle (200,250);
  \draw[dotted] (0,0)
        grid [xstep=15,ystep=12.5]
                          (210,225);
  \draw (0,0)
         ...
         ...
\end{tikzpicture}
```

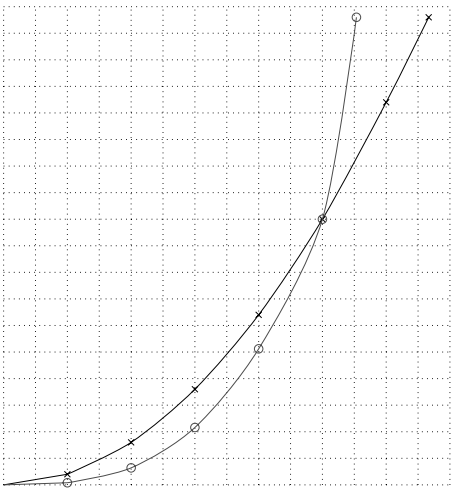
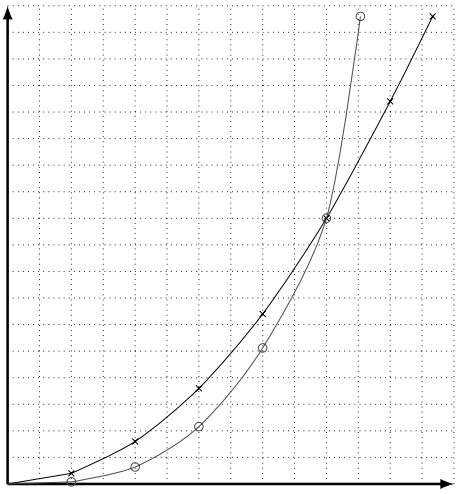


Figura 6.8 Colocación de los ejes

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,-2)
              rectangle (200,220);
  \draw[dotted] (0,0)
        grid [xstep=15,ystep=12.5)]
                    (210,225);
  \draw[very thick,-latex]
          (0,0) -- (0,225); %eje y
  \draw[very thick,-latex]
          (0,0) -- (210,0); %eje x
  \draw (0,0)
         -- plot[mark=x,smooth]
    ...
    ...
\end{tikzpicture}
```

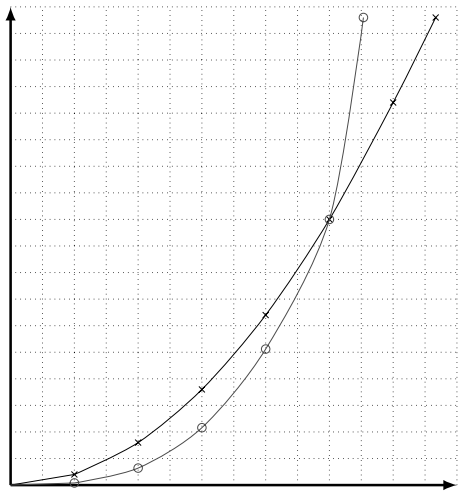


El enunciado \foreach tiene las siguientes partes:

- Las variables del foreach que van separadas entre sí por una diagonal y que consisten de diagonal inversa y un identificador. Se recomienda que los identificadores consistan únicamente de letras, porque a veces, dependiendo del contexto, al sustituirse pueden dar problemas inesperados. En los dos ejemplos anteriores, la lista de variables consiste de una variable a las que llamamos \x y y en el primero y segundo for, respectivamente.
- Una lista de valores que deben tomar las variables. Cada elemento de la lista consiste de un valor por cada variable enunciada, separados los elementos entre sí por comas y los valores de un mismo elemento por diagonales. Esta lista va entre llaves.
- Por último tenemos una lista de enunciados donde pueden o no aparecer las variables del for.

Figura 6.9 Segunda forma de pintar la retícula

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,-2)
            rectangle (200,220);
  \foreach \x in {15,30,45,60,75,
                  90,105,120,135,
                  150,165,180,195,
                  210} {
    \draw[line width=.3pt,dotted]
                 (\x,0) -- (\x,225);
  }
  \foreach \y in {12.5,25,37.5,50,
                  62.5,75,87.5,100,
                  112.5,125,137.5,
                  150,162.5,175,
                  187.5,200,212.5,
                     225}  {
    \draw[line width=.3pt,dotted]
                 (0,\y) -- (210,\y);
  }
  \draw (0,0)
        ...
        ...
\end{tikzpicture}
```



Ambos ejemplos tienen una sola variable y las líneas que se pintan están a intervalos constantes. En general, cuando se dan estas dos condiciones puedes reemplazar a parte o toda la lista por una lista donde se infieren los puntos que faltan y que tiene la siguiente forma:

⟨*primer valor*⟩ , ⟨*segundo valor*⟩ , ... , ⟨*último valor*⟩

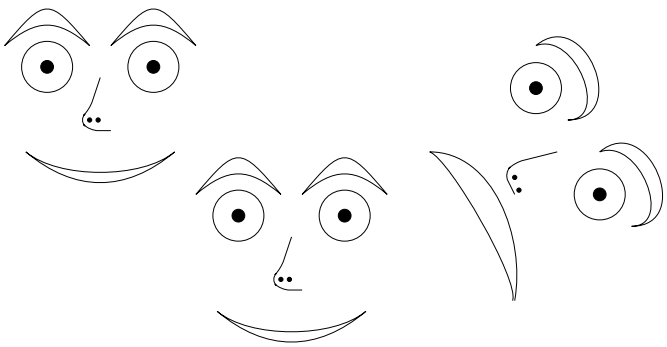
Los dos \foreach que tenemos los podríamos haber escrito de las formas que se muestran en el listado 6.1.

Código 6.1 Formas de hacer retículas

```
\foreach \x in {15,30,45,60,75,90,105,120,135,
          150,165,180,195,210} {
  \draw[line width=.3pt,dotted](\x,0) -- (\x,225);
}
\foreach \y in {12.5,25,37.5,50,62.5,75,87.5,100,
         112.5,125,137.5,150,162.5,175,
         187.5,200,212.5,225}  {
  \draw[line width=.3pt,dotted](0,\y) -- (210,\y);
}
\foreach \x in {15,30,...,210} {
  \draw[line width=.3pt,dotted] (\x,0) -- (\x,225);
}
\foreach \y in {%
      12.5,25,37.5,...,225}  {
  \draw[line width=.3pt,dotted] (0,\y) -- (210,\y);
}
```

Aunque aparentemente el colocar los números sobre los ejes parece sencillo, no sabemos cómo colocar objetos cuando trabajamos con TikZ, y no se logra simplemente con un \draw cuyo objetivo es pintar líneas. En la siguiente sección resolveremos este problema. Antes de abandonar esta sección quisiéramos hacerte notar que hay otra forma de alimentarle coordenadas a \draw y que es mediante un archivo con las mismas, que pudo ser generado por una utilería de linux. Para el lector interesado le recomendamos que vea el manual de TikZ. Terminamos esta sección con un divertimento en la figura 6.10 que usa únicamente los conceptos que hemos dado hasta ahora.

Figura 6.10 Divertimento con sólo trazo de líneas



Los comandos de TikZ que logran el dibujo anterior se encuentran en el listado 6.2.

Código 6.2 Dibujos arbitrarios (1/2)

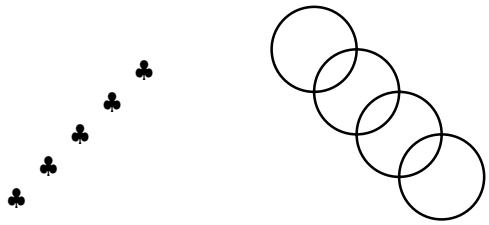
```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (30,30) rectangle (350,180);

  %ojos
  \foreach \x/\y in {%
        50/160,100/160, 140/90,190/90,310/100,280/150} {%
     \draw (\x,\y) circle (12pt);
     \draw[fill] (\x,\y) circle (3pt);
  }

  %cejas
  \foreach \x/\y/\xx/\yy/\a/\b/\aa/\bb in {%
    30/170/70/170/40/140/50/130,
    80/170/120/170/40/140/50/130,
    120/100/160/100/40/140/50/130,
    170/100/210/100/40/140/50/130,
    280/170/295/135/20/0/40/0,
    310/120/325/85/20/0/40/0}  {
    \draw (\x,\y) .. controls +(\a:20pt) and +(\b:20pt) ..
       (\xx,\yy);
    \draw (\x,\y) .. controls +(\aa:30pt) and +(\bb:30pt) ..
       (\xx,\yy);
  }

  %nariz
  \foreach \x/\y in {0/75,90/0} {%
    \pgfputat{\pgfxy(\x,\y)}{%
      \draw[rounded corners] (75,80) -- (70,65) -- (65,60) --
         (70,55) -- (80,55);
      \foreach \xx in {70,74} {%
        \draw[fill] (\xx,60) circle (1pt);
      }

      %boca
      \draw (40,45) .. controls +(320:20pt) and +(220:20pt) ..
         (110,45);
      \draw (40,45) .. controls +(320:30pt) and +(220:30pt) ..
          (110,45);
    } %\pgfputat
  }%foreach
```

Código 6.2 Dibujos arbitrarios (2/2)

```
    %Tercera cara
    \pgfputat{\pgfxy(-40,50)}{%
        \draw[rounded corners] (330,70) -- (310,65) --
             (305,60) -- (310,50);
        \foreach \x/\y in {310/58,312/52}
           \draw[fill] (\x,\y) circle (1pt);
      %boca
      \draw (270,70) .. controls +(0:30pt) and +(80:30pt) ..
            (310,0);
      \draw (270,70) .. controls +(340:10pt) and +(70:10pt) ..
            (309,0);
    }%pgfputat
  \end{tikzpicture}
```

El único comando nuevo en el listado anterior es el de pgfputat que lo que hace es tomar las coordenadas dentro de su alcance (entre las llaves) relativas al primer parámetro (\pgfxy(x,y)).

Actividad 6.51 *Escribe los comandos necesarios en TikZ para dibujar lo siguiente:*



6.10.6. Acomodando objetos

El poder pintar una retícula de la superficie que usamos para dibujar suele ser útil no nada más para graficar funciones, sino también para dibujar y colocar objetos. En general, asociamos objetos – que pueden ser letreros, dibujos, puntos, etc. – a *nodos*. Un nodo tiene las siguientes características:

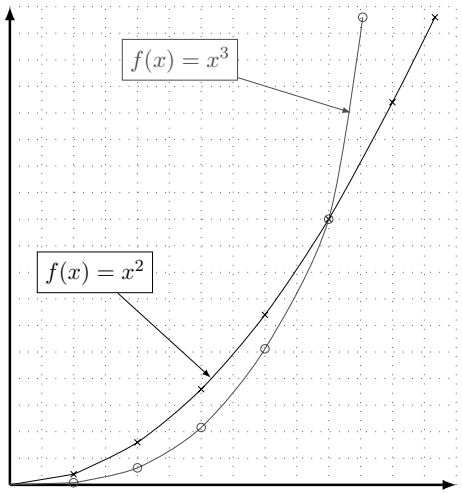
- Ocupa una cierta posición, dada por una pareja (x,y).
- Tiene alguna forma que puede ser circle, rectangle, diamond, ellipse.
- Puede tener un identificador asociado.
- Puede dibujarse o no su contorno con cualquier tipo de línea.
- Puede rellenarse o no con distintos patrones y en colores diversos.

El formato general para colocar un nodo es el siguiente:

\node (⟨*etiqueta*⟩) [⟨*modificadores*⟩] at (⟨*posición*⟩) {⟨*contenido*⟩};

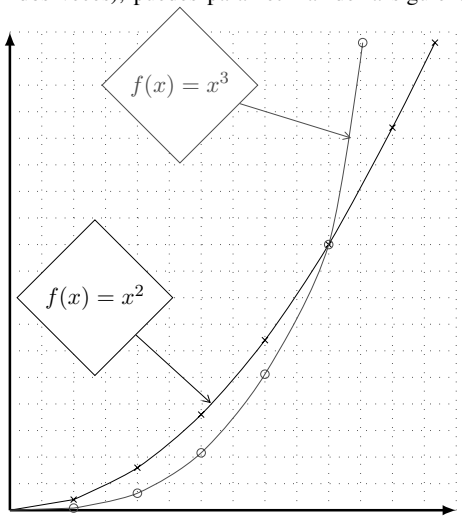
Por ejemplo, puedes marcar algunas posiciones en el espacio en el que graficas las funciones para colocar allí una descripción de las mismas.

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0) rectangle (200,220);
% retícula
  \foreach \x in {15,30,45,60,75,90,
    ......
%nodos
  \node (x2) %etiqueta
      [inner sep=0pt,outer sep=0pt]
            %modif.
      at (95,50) %dónde
      {}; %contenido
  \node (letx2)
      [rectangle,draw,fill=white]
      at (40,100) {$f(x)=x^2$};
  \draw[-latex] (letx2) -- (x2);
       %línea entre dos nodos
  \node (x3)
      [inner sep=0pt,outer sep=0pt]
      at (161,175) {};
  \node (letx3)
      [red,rectangle,draw,fill=white]
      at (80,200) {$f(x)=x^3$};
  \draw[-latex,red] (letx3) -- (x3);
\end{tikzpicture}
```



Puedes pensar en acomodar los nodos con un \foreach. Aunque en este caso sólo son en realidad dos objetos (tres líneas que se repiten – casi – dos veces), puedes parametrizar de la siguiente manera:

```
  ......
e\foreach \etiq/%
  \xe/\ye/\letr/\xl/\yl/\cont/\clor
     in {x2/95/50/letrx2/40/
    100/$f(x)=x^2$/black,
    x3/161/175/letx3/
    80/200/$f(x)=x^3$/red} {
  \node (\etiq)%
    [inner sep=0pt,outer sep=0pt]
     at (\xe,\ye) {};
  \node (\letr)%
    [\clor,diamond,draw,fill=white]
     at (\xl,\yl)
     {\cont};
  \draw[-latex,\clor]
     (\letr) -- (\etiq);
}
  ......
```



Explicamos ahora algunos de los modificadores.

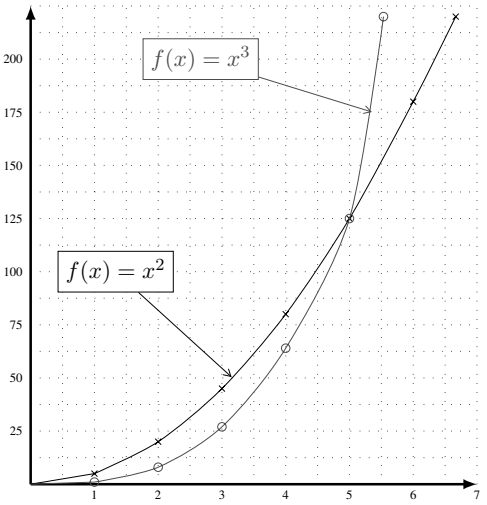
- La forma (*shape*) puede ser cualquiera de las que se encuentran en la biblioteca shapes de TikZ, que se carga mediante el comando

  ```
  \usetikzlibrary{shapes}
  ```

  colocado en el preámbulo del documento. En este caso, en el ejemplo no parametrizado utilizamos rectangle, mientras que en el parametrizado utilizamos diamond. La forma por omisión es rectangle.
- En algunos casos no deseas separación entre la posición del nodo y su exterior. Esta separación se da por omisión alrededor del texto contenido en el nodo, aun cuando no haya texto. Por eso tienes que aclarar que la separación entre el texto y el borde (inner sep) y la separación entre el borde y el entorno (outer sep) es de cero puntos. Este modificador lo utilizamos para marcar que quieres colocarte *exactamente* sobre la línea de la función. Hay otra manera de hacer esto, que veremos un poco más adelante.
- La opción de dibujar o no contorno está dada por draw, que indica que sí se haga. La opción por omisión es no pintar contorno.
- Ya vimos la opción de rellenar el nodo (fill). Para darle un color al relleno y un patrón se hace exactamente igual que con el comando \draw. El valor por omisión es transparente, por lo que deja ver lo que sea que se haya pintado "abajo"; para que no se viera la retícula abajo de nuestras etiquetas es que rellenamos de blanco.
- Por último, el contenido de un nodo puede ser **cualquier cosa**, entre otros, un arreglo, un dibujo, otra retícula, ecuaciones (como lo hicimos nosotros). Ésta es una de las características principales por las que nos gusta usar TikZ: te permite trasplantar el poderío y lo fino de LaTeX a cualquier dibujo o gráfica.

Para colocar los números a lo largo de los ejes haremos uso de la construcción \foreach y del enunciado \node.

```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0)
           rectangle (200,220);
           ...
  \foreach \x in {15,30,45,60,
  ...
  \foreach \x/\etiq in {%
    30/1,60/2,90/3,120/4,
    150/5,180/6,210/7}  {
    \node at (\x,-5){\tiny \etiq};
  }
  \foreach \y in {%
    25,50,...,200}  {
    \node[left] at (0,\y){\tiny \y};
  }  ......
\end{tikzpicture}
```



Al acomodar los números en el eje x tienes que utilizar posición y etiqueta, pues éstas no coinciden. En el eje de las y, en cambio, como coinciden la posición y la etiqueta, puedes utilizar la misma variable para ambas.

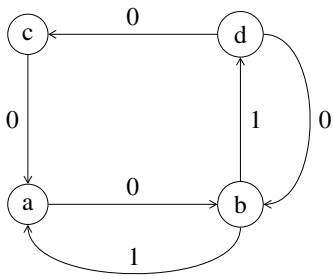
Al colocar las etiquetas en el eje de las y utilizamos un nuevo modificador en el nodo – nota que no utilizamos nombre para el nodo pues no es necesario – que acomoda al contenido del nodo respecto a la posición del mismo. En este caso deseamos que los números se alineen a la derecha, por lo que indicamos que el contenido se extienda *hacia* la izquierda (*left*). Tenemos las siguientes posibilidades para este modificador:

above	anchor=north
above left	anchor=north west
above right	anchor=north east
left	anchor=west
right	anchor=east
below	anchor=south
below left	anchor=south west
below right	anchor=south east

En los modificadores a la izquierda en la tabla puedes agregar un desplazamiento, usando por ejemplo left =2pt.

También puedes usar el \foreach para repetir figuras. Por ejemplo, si deseas dibujar una gráfica, puedes colocar nodos y después unirlos con líneas, como se puede apreciar en la figura 6.11 que se hizo con el código del listado 6.3.

Figura 6.11 Dibujo de una gráfica en LaTeX



Código 6.3 Dibujo de una gráfica

```
\begin{tikzpicture}[x=1pt,y=1pt]
        %Colocación de los nodos
 \foreach \name/\etiq/\x/\y in {
   a/a/10/40 ,b/b/110/40,
   c/c/10/120,d/d/110/120}  {
   \node[circle,draw](\name) at (\x,\y){\etiq};
 }

        %dibujo de los arcos rectos
 \foreach \ffr/\tto/\etiq/\pos in {
   a/b/0/above,b/d/1/right,
   d/c/0/above,c/a/0/left}  {
   \draw[-angle 60] (\ffr) -- node[\pos]{\etiq} (\tto);
 }

        %dibujo de los arcos curveados
 \foreach \ffr/\tto/\etiq/\pos/\sale/\entra in {
   b/a/1/above/270/270,
   d/b/0/right/0/0}  {
   \draw[-angle 60] (\ffr) .. controls +(\sale:40pt)
          and +(\entra:40pt) .. node[\pos]{\etiq}(\tto);
 }
\end{tikzpicture}
```

De entre lo que no revisamos antes está el signo de + al frente de las coordenadas de controls. Esto se refiere a que el ángulo, que corresponde a la primera coordenada, sea tomado en relación al centro del nodo al que se refiere; la segunda coordenada indica la distancia del centro del nodo a la parte más alta de la curva (un bezier). También es nueva la colocación de etiquetas, que se hace también colocando un nodo *sobre la trayectoria*, antes del destino de la línea.

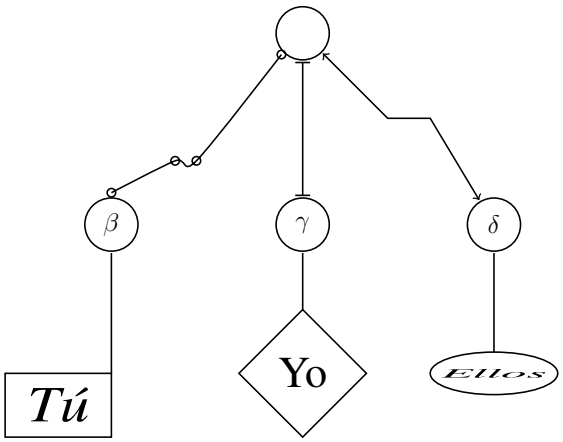
Este tipo de dibujos pueden alcanzar muchísima complejidad en TikZ, por lo que no seguimos con el tema. Recomendamos nuevamente al lector interesado que consulte el manual de TikZ.

Actividad 6.52 *Codifica los dibujos de la figura 6.12 en LaTeX.*

Actividad 6.53 *Haz el código necesario para dibujar una cara similar a la que se usó en esta sección.*

Con esto damos por terminada la parte de dibujar con LaTeX. Sobre todo en el paquete TikZ hay una casi infinidad de posibilidades. Nos faltó, por ejemplo, cómo dibujar árboles o gráficas de manera más directa, pero se supone que este material lo que debe hacer es abrirles el apetito y no emparcharlos, así que por lo pronto acá dejamos este tema.

Figura 6.12 Actividad 6.52



6.11. Referencias e índices

Un documento científico o, en general, un documento profesional como un artículo, un libro, un reporte, requiere, además del contenido, una serie de elementos estéticos y de ayuda muy importantes para el lector de la obra. Estos elementos incluyen: tipos de letras, sus tamaños, la numeración de los capítulos, secciones, fórmulas, cuadros o tablas, así como una serie de elementos (opcionales) como tablas de contenido, de figuras, índices. Al conjunto de estos elementos se les conoce como tipografía.

La tipografía de una obra, en general, la fija y atiende el editor, sobre todo en cuanto a cuestiones estéticas. Los índices y la bibliografía o referencias son, necesariamente, provistas por el autor. En muchos de los casos, tendrás que actuar como ambos: autor y editor de tus trabajos.

LaTeX es ideal para ayudarte en la definición de la tipografía, incluyendo los elementos opcionales. En este capítulo hemos revisado ya prácticamente todo lo que necesitas para realizar documentos complejos, por lo que nos resta aclarar cómo realizar algunos de los elementos opcionales que mencionamos en el párrafo anterior.

6.11.1. Referencias bibliográficas

Como ya discutimos, la presencia de referencias bibliográficas en tus trabajos te permite incursionar en más temas que consideres interesantes. Esto significa que las referencias deben ser precisas y dirigir al lector de tu obra con el menor esfuerzo posible.

Hay varias formas de darle formato a las bibliografías; de hecho, en la mayoría de los casos te encontrarás siguiendo las reglas que para tal efecto dicte la editorial. Por tanto, una de las tareas más importantes cuando envías un libro, artículo o cualquier otro trabajo escrito para su publicación, es generar la lista de referencias bibliográficas siguiendo dichas reglas.

En la prehistoria de la computadora

En la pre-historia de las computadoras, es decir hace una veintena de años, teníamos que componer a mano la lista de bibliografías. Por ejemplo, teníamos que teclear esto:

```
\begin{thebibliography}{alguna-entrada}
\bibitem[etiqueta1]{llave1} Información sobre la entrada bibliográfica
\bibitem[etiqueta2]{llave2} Información sobre la entrada bibliográfica
  . . .
\end{thebibliography}
```

Para citar esas referencias usamos el comando \cite y usamos la *llave* apropiada, por ejemplo, \cite{llave1}.

Algunos de los problemas con esta forma tradicional de ingresar bibliografía en nuestros trabajos son:

1. Es muy difícil mantener la consistencia en las citas, particularmente en un documento con contribuciones de varios autores. Entre las dificultades más comunes, encontramos, ewntre otros, variaciones en el uso de abreviaturas o nombres completos, el uso de itálicas o comillas para los títulos.
2. Si ya tienes una lista bibliográfica ordenada de acuerdo a cierto estilo, por ejemplo alfabético por autor y año, es extremadamente difícil de convertir a otro estilo, por ejemplo usando números de acuerdo al orden de cita.
3. Es virtualmente imposible mantener una gran base de referencias bibliográficas que pueda ser reutilizada en distintos documentos.

BibTeX

A continuación describiremos una posible forma de resolver el problema de las listas de bibliografías. Está basado en LaTeX y su programa hermano BibTeX. Es importante hacer notar que BibTeX ha estado en circulación por muchos años y, por lo tanto, existen muchos estilos que hoy podríamos llamar *estándar* y que están a nuestra disposición, por lo que es muy fácil encontrar el estilo adecuado para tu trabajo.

La estructura básica de una entrada BibTeX tienes tres partes básicas:

(a) El tipo de la entrada (book, article, inproceedings y otros). Nótese que el tipo siempre va en inglés.

(b) Una llave elegida por el usuario que identifica la publicación. Esta llave es la que debe usarse como parámetro para el comando \cite.

(c) Una serie de campos separados entre sí por comas, donde cada campo consiste de un identificador de campo y sus datos entre comillas o llaves, por ejemplo:

```
author="Fulanito Labrador",
journal = "Las bicicletas",
title = {Los paseos por la ciudad},
```

Existen varios esquemas para asociar convenientemente las llaves con sus respectivas entradas en la base. Un esquema popular es el conocido como Harvard, donde tomamos el apellido del autor (en minúsculas) y el año de la publicación y los combinamos para crear la llave usando dos puntos. Por ejemplo, *smith:1987*. BIBTEX lee las entradas del archivo de bibliografías, con extensión *.bib*, y al formato que se le dará a las entradas lo controla un archivo de estilo, con extensión *.bst*. Por razones de espacio, no discutiremos en este trabajo los comandos que pueden usarse en los archivos de estilo, además de que, como ya hemos mencionado, hay muchos estilos ya definidos por personas o por organizaciones, por ejemplo por la Sociedad Matemática Americana (AMS, por sus siglas en inglés), por la Association for Computing Machinery (ACM), revistas de prestigio, casas editoriales. A continuación describimos algunos estilos muy usados y que no están asociados a ninguna asociación o casa editorial. Es importante notar que no necesitas obtener una copia de los archivos de estilo (.bst) de los que usamos aquí como ejemplos, ya que se incluyen con todas las distribuciones de LATEX.

plain El estilo estándar de BIBTEX. Las entradas son ordenadas alfabéticamente con etiquetas numéricas.

unsrt Similar al anterior, pero las entradas son impresas en el orden en que fueron citadas en el documento. Utiliza etiquetas numéricas.

alpha Similar a plain, pero las etiquetas de las entradas son formadas a partir del nombre del autor y del año de publicación.

abbrv Similar a plain, pero las entradas son más compactas, dado que los nombres de pila, los meses y los nombres de las revistas son abreviados.

acm Usado para las publicaciones de la Association for Computing Machinery. Tiene el nombre del autor (apellido y luego nombre) en mayúsculas pequeñas y usa números como etiquetas.

Ahora te vamos a mostrar el efecto de usar BIBTEX en un documento y luego te mostraremos qué fácil es cambiar el estilo y los dramáticos cambios que generan estos cambios en la salida. Nuestros archivos fuente serán: `ejemplo.tex`, nuestro documento LATEX y `bejemplo.bib`, nuestra base de datos de bibliografías. En las figuras 6.13 y 6.15 puedes ver estos documentos.

Figura 6.13 Ejemplo de LATEX usando BIBTEX (1/2)

```
\documentclass{article}
\usepackage[latin1]{inputenc}
\pagestyle{empty}
\begin{document}
\section{Ejemplo de citas}
```

Figura 6.13 Ejemplo de LaTeX usando BibTeX (2/2)

```
Iniciamos con la cita de un libro~\cite{nonaka95:_knowl_creat_compan}
y una tesis doctoral a continuación~\cite{van96:_impac_kbs}.

También quieres citar un artículo escrito por un único
autor~\cite{boehm88} y luego otro donde hay más de un
autor~\cite{boehm89:_theor_w}. También quieres intentar, por
supuesto, hacer una cita a unos proceedings de una
conferencia~\cite{01:_progr_web_high_level_languag} y para terminar
vamos a hacer una cita múltiple, donde la primer entrada es a un
artículo publicado en el Web~\cite{wadler-how,boehm89:_theor_w}.

\bibliographystyle{plain}
\bibliography{bejemplo}
\end{document}
```

Lo primero que debes entender es cómo funcionan LaTeX y BibTeX juntos: primero corres el comando `latex` con el archivo fuente como argumento (figura 6.14); después corres el comando `bibtex` con el mismo argumento y luego deberás ejecutar `latex` dos veces para que LaTeX pueda resolver todas las referencias. El resultado, puedes apreciarlo en la figura 6.16, donde lo hemos puesto usando tanto el estilo `plain` como el estilo `acm`.

Figura 6.14 Salida de LaTeX al ejecutarlo por primera vez sobre el archivo usando BibTeX

```
%latex ejemplo
This is TeX, Version 3.14159 (Web2C 7.4.5)
LaTeX Warning: Citation 'nonaka95:_knowl_creat_compan' on page 1
undefined on input line 6.
LaTeX Warning: Citation 'van96:_impac_kbs' on page 1 undefined
on input line 7.
LaTeX Warning: Citation 'boehm88' on page 1 undefined on input
line 10.
LaTeX Warning: Citation 'boehm89:_theor_w' on page 1 undefined
on input line 11.
LaTeX Warning: Citation '01:_progr_web_high_level_languag' on
page 1 undefined on input line 13.
LaTeX Warning: Citation 'wadler-how' on page 1 undefined on
input line 15.
LaTeX Warning: Citation 'boehm89:_theor_w' on page 1 undefined
on input line 15.

No file ejemplo.bbl.
```

Figura 6.15 Ejemplo de una base de datos BIBTEX

```
@Book{nonaka95:_knowl_creat_compan,
  author =       {Nonaka, I. and Takeuchi, H.},
  title =        {The Knowledge-Creating Company},
  publisher =    {Oxford Univ Press},
  year =         1995,
  address =      {New York}}

@PhdThesis{van96:_impac_kbs,
  author =       {Van Wegen, B},
  title =        {Impacts of KBS on cost and structure of
                  prouduction processes},
  school =       {Universeit van Amsterdan},
  year =         1996}

@Article{boehm88,
  author =       {Boehm, B.},
  title =        {A spiral model of software development and
                  enhancement},
  journal =      {IEEE Computers},
  year =         1988,
  pages =        {61-62}}

@Article{boehm89:_theor_w,
  author =       {Boehm, B. and Ross, R.},
  title =        {Theory-W software project management: principles
                  and examples},
  journal =      {IEEE Trans. Softw. Eng.},
  year =         1989,
  volume =       15,
  number =       7,
  pages =        {902-916}}

@Proceedings{01:_progr_web_high_level_languag,
  title =        {Programming the Web with High-Level Languages},
  year =         2001,
  volume =       2028,
  series =       {Lecture Notes in Computer Science},
  month =        {April},
  organization = {10th European Symposium on Programming,
                  ESOP 2001},
  publisher =    {Springer Verlag}}

@misc{ wadler-how,
  author = "Philip Wadler",
  title = {How to Add Laziness to a Strict Language Without
           Even Being Odd},
  url = "citeseer.nj.nec.com/102172.html" }
```

Figura 6.16 Ejemplo usando los estilos BibTeX: acm (arriba) y plain (abajo)

1 Ejemplo de citas

Iniciamos con la cita de un libro [4] y una tesis doctoral a continuación [5].

También queremos citar un artículo escrito por un único autor [2] y luego otro donde hay más de un autor [3]. También queremos intentar, por supuesto, hacer una cita a unos proceedings de una conferencia [1] y para terminar vamos a hacer una cita múltiple, donde la primer entrada es a un artículo publicado en el Web [6, 3].

References

[1] 10th European Symposium on Programming, ESOP 2001. *Programming the Web with High-Level Languages*, volume 2028 of *Lecture Notes in Computer Science*. Springer Verlag, April 2001.

[2] B. Boehm. A spiral model of software development and enhancement. *IEEE Computers*, pages 61–62, 1988.

[3] B. Boehm and R. Ross. Theory-w software project management: principles and examples. *IEEE Trans. Softw. Eng.*, 15(7):902–916, 1989.

[4] I. Nonaka and H. Takeuchi. *The Knowledge-Creating Company*. Oxford Univ Press, New York, 1995.

[5] B Van Wegen. *Impacts of KBS on cost and structure of prouduction processes*. PhD thesis, Universeit van Amsterdam, 1996.

[6] Philip Wadler. How to add laziness to a strict language without even being odd.

1 Ejemplo de citas

Iniciamos con la cita de un libro [4] y una tesis doctoral a continuación [5].

También queremos citar un artículo escrito por un único autor [2] y luego otro donde hay más de un autor [3]. También queremos intentar, por supuesto, hacer una cita a unos proceedings de una conferencia [1] y para terminar vamos a hacer una cita múltiple, donde la primer entrada es a un artículo publicado en el Web [6, 3].

References

[1] 10th European Symposium on Programming, ESOP 2001. *Programming the Web with High-Level Languages* (April 2001), vol. 2028 of *Lecture Notes in Computer Science*, Springer Verlag.

[2] Boehm, B. A spiral model of software development and enhancement. *IEEE Computers* (1988), 61–62.

[3] Boehm, B., and Ross, R. Theory-w software project management: principles and examples. *IEEE Trans. Softw. Eng. 15*, 7 (1989), 902–916.

[4] Nonaka, I., and Takeuchi, H. *The Knowledge-Creating Company*. Oxford Univ Press, New York, 1995.

[5] Van Wegen, B. *Impacts of KBS on cost and structure of prouduction processes*. PhD thesis, Universeit van Amsterdam, 1996.

[6] Wadler, P. How to add laziness to a strict language without even being odd.

6.11.2. Índices

Para encontrar un tema de interés en un documento grande, el lector tiene dos alternativas, usar la tabla de contenidos o el índice. En general, el índice es la forma de acceso más común a un documento técnico y, por tanto, una parte esencial del documento.

Al igual que las referencias bibliográficas, para generar un índice requerimos de la combinación de LaTeX y otro programa externo llamado *MakeIndex*. En tu documento LaTeX debes usar el comando \index para indicar una nueva entrada en el índice del documento. Cuando ejecutes latex en tu documento, se crea un nuevo archivo con extensión .idx en el que aparecen las entradas seleccionadas por \index y las páginas en que éstas aparecen. Después de esto, cuando ejecutes el comando makeindex, al igual que como sucedía con bibtex, leerá un archivo de estilo (con extensión .ist) y creará el archivo con extensión .ind que tiene la versión con formato del índice de tu documento.

Crear un buen índice no es una tarea sencilla y discutir el tema escapa de los alcances de este documento. Sin embargo, te podemos recomendar crear el índice conforme avances en el desarrollo de tu documento. Debes tomar en cuenta a quién va dirigido el documento, qué es importante y organizar el índice de manera tal que el lector pueda encontrar fácilmente lo que busca.

El comando \index soporta hasta tres niveles de entradas en el índice. Para acceder a los sub y subsub niveles, el argumento de \index debe contener la entrada principal y las sub-entradas separadas por el carácter **!**. Por ejemplo:

```
Página 3: \index{funciones ! par&\'a&metros ! paso de}
Página 4: \index{funciones ! recursivas}
Página 5: \index{cerradura ! funcional}
Página 6: \index{cerradura ! con ambientes}
Página 10: \index{paradigma}
Página 11: \index{paradigma ! tipos de}
```

Index

```
funciones, 3
    parámetros, 3
      paso de, 3
    recursivas, 4
cerradura, 5
    funcional, 5
    con ambientes, 6
paradigma, 10
    tipos de, 11
```

6.12. Emacs

Como ya sabes, editar un archivo LaTeX es simplemente editar un archivo de texto con comandos especiales que interpreta el comando latex o pdflatex. En este sentido, con lo revisado de Emacs en la sección 4.3, puedes editar cómodamente tus documentos LaTeX.

Sin embargo y como una muestra más de la versatilidad y poderío de Emacs, te mostraremos algunos paquetes que extienden Emacs y lo convierten en un poderoso editor especializado para LaTeX.

6.12.1. AUCTeX

AUCTeX es un ambiente integrado para editar archivos LaTeX y TeX e incluye una gran cantidad de herramientas que iremos revisando a lo largo de esta sección.

Una pregunta que es sano hacernos es ¿qué puede hacer AUCTeX que no podamos hacer sólo con la ayuda de Emacs? Varias cosas: te permite ejecutar TeX/LaTeX y demás comandos y herramientas relacionados con el proceso de documentos, como visores de DVI, Postscript o PDF. En particular, la habilidad de ejecutar LaTeX desde AUCTeX es interesante porque te permite *navegar* los errores y te ofrece documentación sobre cada tipo de error.

AUCTeX también nos ayuda a indentar nuestro código fuente y tiene un par de herramientas para ver una vista preliminar, un esquema de tu documento. Además, al estar íntimamente relacionado con Emacs, AUCTeX ofrece una gran cantidad de macros y funciones auxiliares que te permiten insertar comandos y editar tus documentos rápida y fácilmente.

Instalando AUCTeX

```
$ smart
install
emacs-auctex
```

AUCTeX es un programa escrito en Emacs-Lisp y es parte estándar de varias distribuciones mayores de Emacs, incluyendo XEmacs. Desgraciadamente AUCTeX no viene con Emacs por omisión; sin embargo existen paquetes pre-compilados para la mayoría de las distribuciones de Linux.

En caso de que no encuentres paquetes para tu distribución de Linux, no hay nada de qué preocuparse, pues ya sabes lo fácil que es instalar paquetes en Emacs (ver la sección 4.3.18). En la siguiente dirección puedes obtener la versión más reciente de AUCTeX:

http://www.gnu.org/software/auctex/.

Para indicarle a Emacs que vas a utilizar AUCTeX, debes poner lo siguiente en tu ∼/.emacs:

```
(require 'tex-site)
(require 'tex)
(require 'latex)
```

Sangrías y formato

AUCTeX puede indentar tu código automáticamente conforme lo escribes. Si presionas C-j en lugar de Enter funcionará exactamente igual. Si presionas tab la línea actual es indentada y el cursor permanece donde está. Incluso el comando format-paragraph de Emacs, M–q, es re-implementado por AUCTeX para que haga lo correcto con tu documento LaTeX. Finalmente, la función LaTeX-fill-buffer (que puedes ejecutar con M–x) indenta todo tu documento, lo cuál es particularmente útil cuando alguien te mandó un documento sin formato.

En LaTeX es común tener documentos que ocupen más de un archivo; por ejemplo, para libros y tesis es común utilizar un archivo por cada capítulo o sección. AUCTeX es capaz de entender documentos que ocupan varios archivos y ayudarte en su edición y manejo, para lo cual debes poner en tu ∼/.emacs lo siguiente:

```
(setq TeX-auto-save t)
```

```
(setq TeX-parse-self t)
(setq-default TeX-master nil)
```

Edición de documentos

AUCTEX estudia el comando \documentclass de tu documento y en función de eso te ayuda a completar comandos de LaTeX, lo cual tiene dos aplicaciones importantes y que a la postre agradecerás infinitamente.

1. Te permite completar comandos parcialmente escritos. Por ejemplo, puedes escribir \renewc y presionar M–tab (TeX-complete-symbol) y AUCTEX hace el resto, es decir, completar el comando \renewcommand por ti. Si más de un comando casa con el prefijo, entonces AUCTEX te ofrece una lista de los posibles comandos para que escojas.
2. Te ayuda a insertar ambientes, esto es pares de la forma \begin — \end.

Un número importante de macros acompañan a AUCTEX y están aquí para ayudarte a editar rápidamente. Las más importantes son las siguientes:

Tabla 6.34 Emacs: comandos de AUCTEX

Comando	Descripción
LaTeX-environment C-c C-e	Inserta un par de la forma \begin — \end.
LaTeX-section C-c C-s	Inserta uno de \chapter, \section, ...
TeX-font C-c C-f seguido de C-r, C-i, C-b, ...	Insertan uno de \textrm{}, \textit{}, \textbf{}, ...
LaTeX-insert-item M–Enter	Inserta un nuevo *item* en un ambiente. Esto es útil en todo tipo de listas, AUCTEX propone el formato adecuado.
TeX-insert-macro (C-c enter)	Con este comando puedes insertar un macro en tu documento y AUCTEX te preguntará por los argumentos del mismo.

AUCTEX provee varios comandos más, pero en la mayoría de los documentos LaTeX estándar, con los que listamos arriba puedes avanzar rápidamente en la edición. Más adelante veremos algunos comandos y utilerías para hacer matemáticas en un modo especial que AUCTEX provee para tal efecto.

Para lograr que AUCTEX te pregunte por los títulos de las secciones, capítulos y otras subpartes del documento que insertas y te proponga nombres inteligentes para etiquetas, es conveniente

agregar a tu ∼/.emacs lo siguiente:

```
(setq LaTeX-section-hook
      '(LaTeX-section-heading
        LaTeX-section-title
        LaTeX-section-toc
        LaTeX-section-section
        LaTeX-section-label)
      )
```

Cómo ejecutar LaTeX

AUCTeX provee un comando que, dependiendo del contexto en que se use, ofrece varias alternativas para ejecutar comandos sobre tu documento fuente.

El comando TeX-command-master (C-c C-c) o su hermano TeX-command-region (C-c C-r) ejecuta LaTeX en todo el documento o en la región especificada, respectivamente.

Cuando ejecutamos LaTeX de esta forma, la vista de Emacs se divide en dos y la salida de la ejecución se despliega en la segunda mitad para que puedas editar y ver el resultado de la ejecución simultáneamente. Si LaTeX encuentra errores, entonces puedes llamar a la función TeX-next-error (C-c `) que moverá el cursor al primer error y desplegará un texto explicativo junto con el mensaje que arrojó LaTeX. Puedes repetir la ejecución de este comando hasta que no encuentres más errores.

Una vez que hayas concluido con el formato de tu documento exitosamente, puedes volver a invocar el comando TeX-command-master y te ofrecerá una alternativa para visualizar el resultado.

Modo para matemáticas

AUCTeX incluye un modo para facilitar la edición de matemáticas y, aunque sigue siendo el mismo AUCTeX, se comporta tan distinto que lo tratamos por separado.

¿Ya te dijimos que TeX fue escrito por un matemático? En caso de que no lo hayamos hecho, sí, TeX fue escrito por un matemático e históricamente siempre ha ofrecido un soporte inigualable para escribir matemáticas. Fiel a esta tradición, AUCTeX te ofrece un modo menor especial para escribir símbolos matemáticos rápidamente. Para entrar a este modo, debes teclear C-c ~, que es el comando LaTeX-math-mode – también se usa para salir del modo matemático de AUCTeX–.

Este modo menor agrega un comando, LaTeX-math-abbrev-preffix o simplemente ` y una vez que llamas este comando (i.e. presionas `) AUCTeX leerá un carácter del teclado y luego insertará un símbolo matemático asociado. La lista de asociación entre carácteres y símbolos se encuentra en la variable LaTeX-math-list y, por supuesto, puede configurarse. Así, por ejemplo, estando en modo matemático la secuencia de teclas ` a inserta α.

Nota: para nosotros, hispano parlantes, escribir acentos es muy importante y, en general, ` sirve para poner acentos invertidos, lo cual dificulta el acceso al modo matemático de AUCTeX. Por lo tanto tienes dos opciones, presionar dos veces ` y luego la letra deseada o bien cambiar el prefijo utilizado por AUCTeX.

6.12.2. RefTEX

RefTEX es un paquete para manejar etiquetas, referencias, citas e índices desde Emacs en documentos TEX o LATEX. No es necesario usar AUCTEX y RefTEX juntos, pero hay pocas cosas que combinan mejor, con la notable excepción de tacos y cerveza.

RefTEX es otro programa escrito, al igual que AUCTEX, en Emacs-Lisp y es un producto de software complejo y grande, pero te daremos una breve introducción a su uso y te podemos garantizar que una vez que te acostumbres a RefTEX te preguntarás cómo pudiste vivir sin él (nosotros nos preguntamos eso con casi todos los productos de software, aplicaciones y paquetes que incluimos en este libro, por ello es que ahora las promovemos incansablemente.)

Para cargar RefTEX en Emacs, agrega lo siguiente a tu ∼/.emacs:

```
(require 'reftex)
```

Tabla de contenidos

La tabla de contenidos de un documento LATEX es algo muy importante y te puede dar una visión global de la estructura de tu trabajo. Sin embargo, la tabla de contenidos existe hasta que *compilas* tu documento fuente y exclusivamente en el documento generado como salida (DVI, PostScript, PDF, etc.). ¿Qué harías y cómo aprovecharías el poder tener una tabla de contenidos generada dentro de tu editor? ¿Qué harías con ella si además esta tabla fuera *navegable*?

El comando reftex-toc (C-c =) hace justo eso, te muestra secciones, etiquetas y entradas de índice definidos en tu documento y más aún, te permite saltar a cualquiera de estos puntos rápidamente.

Etiquetas y referencias

Crear etiquetas para los distintos elementos de tus documentos como figuras, tablas y fórmulas es usualmente una tarea tediosa y poco gratificante. La gente se ha inventado fórmulas y técnicas para asignar etiquetas ordenadamente para después recordarlas fácilmente y hacer referencia a éstas sin tener que buscar en todo el documento. Los resultados varían mucho, pero en general puedes asegurar que es una mala idea y no funciona bien.

RefTEX te ayuda a crear etiquetas únicas y a encontrar la etiqueta adecuada cuando quieres hacer una referencia rápidamente. RefTEX distingue entre etiquetas para distintos ambientes, conoce todos los ambientes estándar (figure, equation, table) y puedes configurarlo para que entienda aún más tipos de etiquetas cambiando el valor de la variable reftex-label-alist.

Crear etiquetas: El comando reftex-label – (C-c () inserta una etiqueta en el punto. Cuando usas este comando RefTEX realizará una de las siguientes acciones:

- Derivar una etiqueta del contexto (por omisión usa una para sección).
- Pedir que le des una cadena descriptiva para la etiqueta (esto lo hace para figuras y tablas).

- Insertar una etiqueta simple hecha con un prefijo y un número (para todos los demás ambientes).

La variable reftex-insert-label-flags controla qué etiquetas son creadas y cómo.

Referir etiquetas: Para hacer una referencia usa el comando reftex-reference (C-c]). Esto te muestra una vista (*outline*) de tu documento con todas las etiquetas de un cierto tipo (*figure*, *equation*, etc.) y algo de contexto para cada etiqueta para que puedas reconocerlas fácilmente. Al seleccionar una etiqueta de esta vista, RefTeX inserta \ref{LABEL} en el buffer original.

Citas

El comando reftex-citation (C-c [) te permite especificar una expresión regular o patrón para buscar en el archivo de base de datos de BibTeX para el documento (tal y como se especifica en el comando \bibliography). Todas las entradas que concuerden con el patrón de búsqueda te serán mostradas para que elijas la apropiada. La lista aparece con *formato* y ordenada.

Una vez que seleccionas una entrada, aparecerá en tu documento fuente una referencia de la forma \cite{KEY}.

Soporte para la generación de índices

RefTeX también te ayuda a generar entradas para el índice de tu documento. Más aún, te permite incluso compilar las entradas existentes alfabéticamente y te las presenta en un buffer para que puedas editarlas.

Para crear una entrada de índice utiliza reftex-index-selection-of-word (C-c /) y para desplegar y editar el índice, reftex-display-index (C-c >).

RefTeX puede hacer muchas cosas más, pero para efectos prácticos lo que hemos cubierto es lo más importante y útil del programa y su uso. Estamos seguros te traerá gratas sorpresas y un incremento considerable de productividad científica.

6.12.3. Preview

El paquete preview-LaTeX se distribuye como parte estándar de AUCTeX y extiende tanto a Emacs como a LaTeX. Esto es porque incluye un estilo para LaTeX y código en Emacs-Lisp para aprovechar este estilo y ayudarte en la edición de documentos.

Para los usuarios de TeX/LaTeX siempre ha sido un problema serio la discusión sobre tener editores que te muestren lo que estás escribiendo (por sus siglas en inglés, *WYSIWYG*), que como ya sabes es algo que no sucede con LaTeX. Sin embargo, la intención del paquete *preview* para Emacs no es exactamente ésta.

Con *preview* se busca un balance entre elementos gráficos y texto. Utiliza la salida gráfica para ciertas construcciones (configurable); hace esto cuando se le solicita y lo hace ahí mismo, dentro del código fuente. Cambiar entre la versión gráfica y el código fuente es fácil y natural y puede hacerse para cada imagen de manera independiente.

Para activar *preview* debes poner lo siguiente en tu ∼/.emacs:

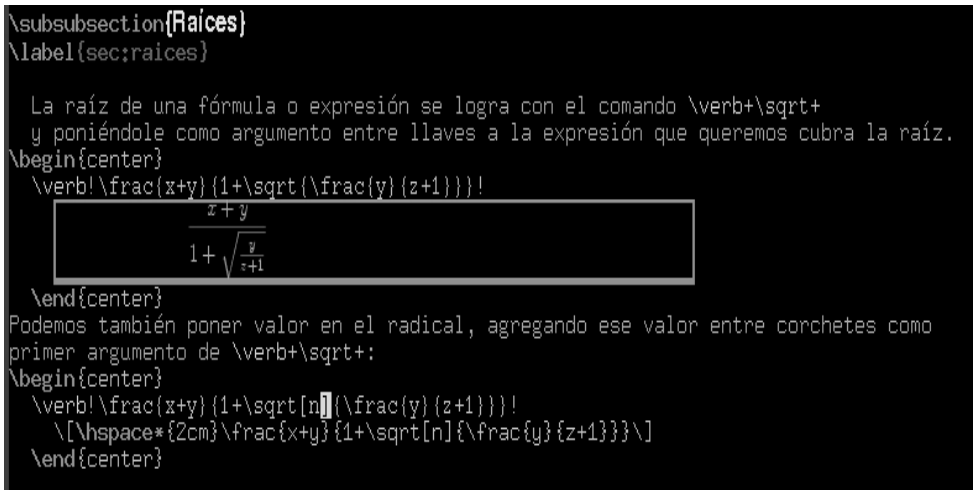
```
(load "preview-latex.el" nil t t)
```

Uso básico de preview

Una vez activado y cuando estás editando un archivo de LaTeX, puedes utilizar el comando preview-document (C-c C-p C-d). En este momento se comenzarán a generar vistas preliminares para distintos objetos en tu documento. *Nota importante*: mientras se generan las vistas preliminares, puedes navegar tu documento en el buffer; sin embargo no es recomendable que edites algo porque el resultado de las vistas podría terminar en lugares erróneos.

En caso de querer editar el buffer mientras se están generando las vistas preliminares, te conviene detener todos los procesos ejecutándose en el fondo con el comando TeX-kill-job (C-c C-k).

Figura 6.17 Ejemplo de preview en la edición de este capítulo



Para ver/editar el código LaTeX para un objeto específico, que es probablemente más útil, puedes ejecutar el comando preview-at-point (C-c C-p C-p) o bien presionar el botón del medio del ratón sobre la vista previa. Ahora puedes editar el código y generar una nueva vista preliminar con C-c C-p C-p nuevamente.

Lenguajes de marcado

7

Los lenguajes de marcado (*markup languages*) utilizan texto común y corriente combinado con información adicional que sirve para definir su semántica o su presentación (a veces ambas). En palabras sencillas: esta información adicional nos dice *qué* significa el texto o *cómo* se presenta al usuario. La información adicional suele estar intercalada en el mismo texto utilizando *marcas* especiales; de ahí el nombre de lenguajes de marcado.

Los lenguajes de marcado tienen sus orígenes en la industria editorial, mucho antes de la creación de computadoras digitales: cuando un manuscrito se preparaba para impresión, un especialista (coloquialmente conocido como "marcador") escribía en los márgenes anotaciones especiales (marcas) que servían de guía a los que transcribían el texto en su forma final ya lista para imprimirse. Estas marcas eran un lenguaje de marcado primitivo (no había necesariamente un estándar, ni tenían una sintaxis formalmente definida), que servía para explicar la presentación del texto: el tamaño de la fuente, el tipo, el estilo, etc.

Con la llegada de las computadoras, los lenguajes de marcado ganaron además la capacidad de obtener automáticamente información semántica: por ejemplo, si en nuestro lenguaje de marcado definimos la marca AUTOR: para que con ella identifiquemos al autor del documento, es trivial hacer un programa que automáticamente obtenga el autor o autores de todos los documentos disponibles y haga una relación entre ellos.

Aunque hay muchísimos lenguajes de marcado actualmente (TEX y PostScript son lenguajes de marcado procedurales), de especial importancia es SGML, el Lenguaje de Marcado Estándar Generalizado (*Standard Generalized Markup Language*). SGML es un meta lenguaje para definir lenguajes de marcado, y de ahí se derivan los dos lenguajes de marcado probablemente más famosos y usados en la actualidad: XML y HTML. Nos centraremos en estos últimos durante el resto del capítulo.

7.1. XML

SGML es un metalenguaje grande y complejo. Implementar un programa (o sistema de programas) que pueda lidiar con *cualquier* lenguaje definido con SGML tendrá que ser igualmente grande y complejo. La mayor parte de los programas que se utilizan para manejar lenguajes definidos con SGML, en la práctica sólo soportan un subconjunto de todas las opciones que ofrece.

En gran medida por este motivo, y las necesidades especiales que presentan las aplicaciones que necesitan comunicarse por Internet (SGML precede por varios años a Internet), fue que se creó XML, el Lenguaje de Marcado Expandible (Expansible Markup Language). XML es un subconjunto simplificado de SGML, lo que hace el escribir programas que lo manejen mucho más fácil, y además está pensado para compartir información, especialmente a través de la red.

XML es inmensamente popular y existen muchísimos lenguajes definidos con XML que se usan hoy en día; por nombrar sólo algunos: RSS (para noticias), MathML (para representar fórmulas matemáticas complejas), XHTML (el sucesor *de facto* de HTML), Scalable Vector Graphics (para gráficos escalares), MusicXML (para notación musical) y miles más. XML también es utilizado por muchos programas para guardar sus archivos, especialmente en el mundo del software libre: OpenOffice, AbiWord, Gnumeric y KOfficen usan XML para guardar sus documentos, por nombrar unos cuantos.

Un documento XML presenta su información en una estructura jerárquica, que podemos ver como un árbol. Todo documento XML tiene un *elemento raíz*, que a su vez puede tener más elementos y/o texto común y corriente. Cada elemento a su vez puede tener más elementos y/o texto y así sucesivamente. Además, cada elemento puede tener *atributos*. Veamos un ejemplo en el listado 7.1

Código 7.1 Documento en XML (1/2)

```
<?xml version="1.0" encoding="UTF-8"?>
<libro>
 <titulo>
  El ingenioso hidalgo don
  Quijote de la Mancha
 </titulo>
 <autor>Miguel de Cervantes Saavedra</autor>
  <parte numero="1">
  <capitulo numero="1">
   <resumen>
    Que trata de la condici&oacute;n y ejercicio
    del famoso hidalgo don Quijote de la Mancha
   </resumen>
```

Código 7.1 Documento en XML (2/2)

```
    <parrafo>
     En un lugar de la Mancha, de cuyo nombre no
     quiero acordarme, no ha mucho tiempo que
     viv&iacute;a un hidalgo de los de lanza en
     astillero, adarga antigua, roc&iacute;n flaco
     y galgo corredor.
    <parrafo>
    ...
   </capitulo>
   ...
  </parte>
  ...
 </libro>
```

La primera línea es la *declaración* XML, que sirver para definir qué versión del estándar estamos usando (generalmente la "1.0") y el conjunto de carácteres en que está el documento (generalmente "UTF-8").

El ejemplo obviamente es de un libro: libro es el elemento raíz, titulo, autor, parte, capitulo, extracto y parrafo son elementos, mientras que numero es un atributo que poseen parte y capitulo. Como estamos definiendo un documento XML que utiliza el conjunto de carácteres UTF-8, sería perfectamente legal definir a los elementos como título, capítuo, párrafo, pero no es lo que suele acostumbrarse. De igual forma, podrían utilizarse acentos (o eñes o diéresis), pero utilizamos "ó" en lugar de "ó" para explicar lo que son las *entidades*.

Las entidades son usadas para poder escribir carácteres en XML que de otra forma sería complicado hacerlo. Por ejemplo, el símbolo de "menor que" (<) no es válido dentro de un documento XML porque es el utilizado para definir cuándo empieza una etiqueta de inicio o final: entonces se utiliza la entidad < (por *"less than"*, en inglés). De igual forma, para usar el símbolo de ampersand (&) se utiliza la entidad &. En todo documento XML están definidas las siguientes cinco entidades:

Entidad	Símbolo
&amp;	&
&gt;	>
&lt;	<
&apos;	"
&quot;	'

pero se pueden definir más entidades y veremos cómo más adelante. Además, todo símbolo en UTF-8 puede utilizarse usando su número identificador; por ejemplo, en lugar de ó para la "ó", podríamos usar Č, porque 268 es el número que le corresponde a "ó" en UTF-8.

El documento de este ejemplo está *bien formado*; esto quiere decir que todos sus elementos tienen etiquetas de inicio y finales (por ejemplo <parte> y </parte>) están bien anidados (todo

elemento, excepto el raíz, está completamente contenido dentro de otro) y los valores de los atributos están entre comillas (también podrían estar entre comillas simples, como en '3').

Para que un documento XML sea correcto, debe estar bien formado; pero eso no es suficiente; además, el documento debe ser *válido*. Un documento XML válido es aquel que cumple con un *esquema* particular. Un esquema nos dice la estructura que debe seguir un documento XML; hay varias formas de definir esquemas.

La forma más vieja de definir esquemas viene de tiempos de SGML y se llama DTD, Definición de Tipo de Documento (*Document Type Definition*). El DTD para los XML de libros sería como se ve en el ejemplo del listado 7.2.

Código 7.2 DTD para archivo en XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY  iacute "&#262;">
<!ENTITY  oacute "&#268;">
...
<!ELEMENT libro ( titulo, autor+, parte+ ) >
<!ELEMENT titulo (#PCDATA) >
<!ELEMENT autor (#PCDATA) >
<!ELEMENT parte ( capitulo+  >)
<!ATTLIST parte numero CDATA #REQUIRED >
<!ELEMENT capitulo ( resumen, parrafo+ ) >
<!ATTLIST capitulo numero CDATA #REQUIRED >
<!ELEMENT resumen ( #PCDATA ) >
<!ELEMENT parrafo ( #PCDATA ) >
```

El DTD sencillamente nos dice qué puede contener cada elemento y qué atributos puede tener; también puede definir las entidades válidas en el documento. El elemento libro tiene un titulo, uno o más autores (autor+) y una o más partes (part+). El DTD también nos dice si un atributo es obligatorio (#REQUIRED) u opcional.

El problema con los DTDs es que son poco flexibles y que no soportan varias características específicas de XML que SGML no tiene (como espacios de nombres). Sin embargo, para documentos con estructuras sencillas, los DTDs cumplen razonablemente el trabajo.

Otra forma de definir esquemas es con XML Schema. Sin embargo, los esquemas de este estilo son algo más complejos que los DTDs; por razones de espacio no los veremos aquí.

Una gran ventaja que ofrece XML para manipular información, es que ya existen las herramientas necesarias para manejar los documentos, en casi cualquier lenguaje de programación existente. Comprobar que un documento XML esté bien formado y sea válido es muy sencillo, por lo que el programador ya sólo necesita encargarse de extraer y manipular la información contenida en el documento. Para esto último también ya existen las herramientas necesarias.

7.2. HTML

HTML nació casi al mismo tiempo que la World Wide Web, de la necesidad de poder presentar información en la red de forma rápida y sencilla. SGML era muy complejo como para lo que se requería y XML todavía no se inventaba; así que HTML fue el resultado de querer resolver un problema muy concreto de la forma más sencilla posible. Se utilizó una estructura similar a la de un lenguaje SGML, pero mucho menos estricto y con mucha tolerancia a fallos.

Eso, aunado a que al inicio no había un estándar propiamente sino sólo una serie de reglas no muy claramente especificadas, hizo que HTML terminara siendo un lenguaje poco consistente y demasiado permisivo en su manejo de errores (en general los navegadores desplegaban una página no importaba cuántos errores tuviera el documento HTML).

Con la fundación del Consorcio de la World Wide Web (*World Wide Web Consortium*), o W3C, se empezaron a corregir muchos de los problemas que inicialmente tenía HTML; aunque ahora el estándar de XHTML es bastante estricto y formalmente definido, todavía existen miles de páginas en la red que siguen usando (o abusando) de las versiones iniciales de HTML.

El estándar actual es el de XHTML 1.0 y 1.1 y es en ése en el que nos centraremos aquí.

7.2.1. XHTML

XHTML es básicamente una limpieza de HTML para que se comporte como un documento XML bien formado y que sea válido, utilizando uno de los tres DTDs que la W3C especifica para XHTML.

La W3C ofrece 3 DTD distintos para poder tener uno que ofrezca compatibilidad con versiones anteriores de HTML; un DTD es el llamado *estricto* y es el que debería usarse para páginas nuevas si se puede garantizar que quienes las vean tendrán navegadores razonablemente modernos que puedan desplegarlas sin problemas. El segundo es el *transicional* y es un poco más flexible que el estricto; es el que hay que usar si se quiere convertir paulatinamente páginas escritas antes de que se definiera XHTML o para conservar compatibilidad para navegadores ya algo viejos. El tercero es para utilizar marcos. Por razones de espacio aquí sólo veremos XHTML estricto.

Un documento XHTML mínimo podría ser el que se ve en el listado 7.3.

Una vez más, podríamos utilizar directamente "é" en lugar de é, porque el documento utiliza el conjunto de carácteres UTF-8, pero usamos la entidad para recalcar que en XHTML casi todas las letras acentuadas tienen definida una entidad, así como muchos símbolos de otros idiomas (por ejemplo, α despliega α).

Dado que un documento en XHTML es un documento XML también, empieza con la declaración XML. Después viene el tipo de documento, donde le decimos que utilice la versión estricta del estándar XHTML 1.0. El elemento raíz de un documento XHTML es html y éste a su vez tiene los elementos de cabeza (head) y cuerpo (body). En este ejemplo sólo ponemos el título de la página en la cabeza y un encabezado de nivel 1 (h1) y un párrafo (p) en el cuerpo.

Código 7.3 Documento en XHTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
  <title>P&aacute;gina Personal de Fulano de Tal</title>
 </head>
 <body>
  <h1>P&aacute;gina Personal de Fulano de Tal</h1>
  <p>
   Hola, yo soy Fulano de Tal y &eacute;sta
   es mi p&aacute;gina personal.
  </p>
 </body>
</html>
```

Todos los elementos definidos en XHTML cumplen una función semántica, no de presentación. Aunque por omisión ciertos elementos se despliegan con características especiales (por ejemplo, los encabezados de nivel 1 tienen una fuente de mayor tamaño que los de nivel 2), esto es sólo por convención y tratando de emular cómo funcionaba HTML originalmente. XHTML realmente no dice casi nada acerca de cómo debe verse un documento; la presentación del mismo es independiente de su *contenido*.

Para modificar la presentación de un documento XHTML se utilizan hojas de estilo, que es lo que veremos a continuación.

7.3. CSS

Las Hojas de Estilo en Cascada (*Cascading Style Sheets*) es el medio a partir del cual se modifica cómo se ve un documento XHTML; pero también se pueden utilizar para documentos XML arbitrarios o para documentos HTML aunque no sean necesariamente XHTML.
Veamos cómo funcionan siguiendo el ejemplo en el listado 7.4.

Código 7.4 Hoja de estilo en cascada (1/2)

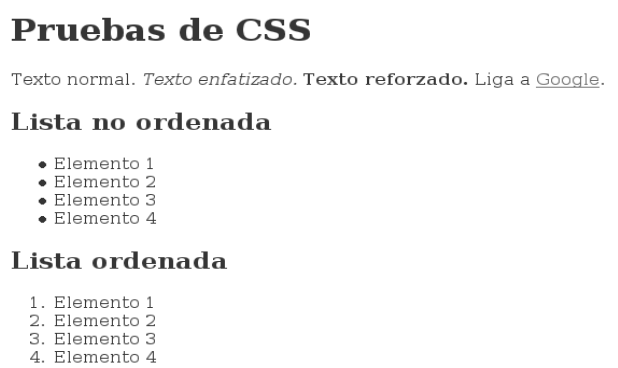
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
  <title>Pruebas de CSS</title>
 </head>
```

Código 7.4 Hoja de estilo en cascada (2/2)

```
<body>
 <h1>Pruebas de CSS</h1>
 <p>
  Texto normal.
  <em>Texto enfatizado.</em>
  <strong>Texto reforzado.</strong>
  Liga a <a href="http://www.google.com">Google</a>.
 </p>

 <h2>Lista no ordenada</h2>
 <ul>
  <li>Elemento 1</li>
  <li>Elemento 2</li>
  <li>Elemento 3</li>
  <li>Elemento 4</li>
 </ul>

 <h2>Lista ordenada</h2>
 <ol>
  <li>Elemento 1</li>
  <li>Elemento 2</li>
  <li>Elemento 3</li>
  <li>Elemento 4</li>
 </ol>
</body>
</html>
```

Sin utilizar ninguna hoja de estilo, la página se vería como en la figura 7.1.

Figura 7.1 Página sin hoja de estilo



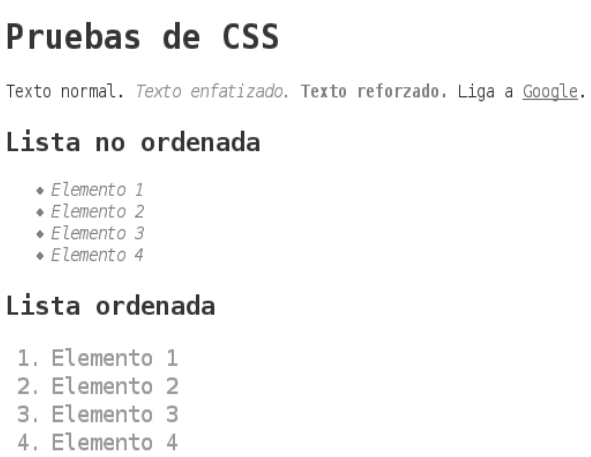
Sin modificar en *nada* el contenido del documento, podemos cambiar por completo cómo se ve, usando la hoja de estilo del listado 7.5.

Código 7.5 Documento en XML

```
body {
    font-family: mono;
    font-size: 14px;
}
em {
    color: #0f0;
}
strong {
    color: #f00;
}
ul {
    color: #f0f;
    font-style: italic;
}
ol {
    color: #0ff;
    font-size: large;
}
```

El resultado puede apreciarse en la figura 7.2.

Figura 7.2 Página con hoja de estilo



Para que el documento XHTML use la hoja de estilo, se utiliza el elemento link en la cabeza del mismo:

Código 7.6 Documento en XHTML con *link*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
  <title>Pruebas de CSS</title>
  <link rel="stylesheet" href="example.css" type="text/css" />
 </head>
```

Sin importar que tan bien o mal se vea la página, lo relevante es que su apariencia es completamente independiente de su contenido. El que lo primero sea ortogonal a lo segundo, permite a los creadores del contenido concentrarse sólo en la información y dejarle la presentación a diseñadores gráficos o a alguien que se encargue exclusivamente de eso (y así evitar que las páginas se vean como en el ejemplo).

Las hojas de estilo también sirven para poder determinar cómo se representará una página en distintos medios. Se puede utilizar una hoja de estilo para presentarla en un navegador normal; otra especialmente hecha pensando en impresoras; una más para lectores de pantalla (para usuarios ciegos o con visión limitada).

7.3.1. Javascript

JavaScript (que no tiene casi nada que ver con el lenguaje de programación Java) es un lenguaje de programación que se usa principalmente dentro de un navegador. Dado que es posible acceder al contenido de una página a través de JavaScript, es una opción muy sencilla para crear contenido dinámico y darle la posibilidad al usuario de interactuar con la página sin necesidad de comunicarse todo el tiempo con el servidor (porque el código JavaScript se ejecuta en el cliente, en el navegador).

Todo el contenido de una página está disponible a través del Modelo de Objeto del Documento (*Document Object Model* o DOM), que es básicamente una representación en memoria de la jerarquía tipo árbol que tiene un documento HTML o XML.

Por ejemplo, si tenemos la página en el listado 7.7,

Código 7.7 Documento en XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
  <title>JavaScript</title>
  <script type="text/javascript" src="javascript.js" />
 </head>
 <body>
  <h1>JavaScript</h1>
   <table id="myTable">
    <tr>
     <td><strong>Elementos</strong></td>
     <td><strong>N&uacute;mero</strong></td>
    </tr>
   </table>
   <input type="button" value="Agregar rengl&oacute;n"
                        onclick="addRow(document);" />
   <input type="button" value="Borrar rengl&oacute;n"
                        onclick="delRow(document);" />
 </body>
</html>
```

y el archivo javascript.js tiene definidas las funciones del listado 7.8.

Código 7.8 Documento en XML

```
function addRow(document)
{
        var table   = document.getElementById('myTable');
        var numRows = table.rows.length;
        var newRow  = table.insertRow(numRows);
        var cell1   = newRow.insertCell(0);
        var cell2   = newRow.insertCell(1);

        cell1.innerHTML = 'Nuevo Elemento';
        cell2.innerHTML = numRows;
}
function delRow(document)
{
        var table   = document.getElementById('myTable');
        var numRows = table.rows.length;
        if (numRows > 1)
                table.deleteRow(numRows-1);
}
```

Entonces cada vez que se haga click en el botón “Agregar renglón”, un nuevo renglón aparecerá en la tabla; y cada vez que se haga click en “Borrar renglón”, se le quitará un renglón a la tabla.

El primer botón de la página tiene definido (usando el atributo onclick) que cada vez que se le haga click, se mande llamar a la función addRow con el parámetro document. En JavaScript, document es una variable que representa a todo el documento HTML o XML. El segundo botón tiene definido de forma análoga que se mande llamar la función delRow.

La función addRow hace lo siguiente: primero obtiene el objeto de la tabla usando la función getElementById. Ésta es una función del DOM y nos permite obtener cualquier elemento, siempre y cuando tengamos su identificador (en la página, al definir la tabla usamos el atributo id).

Después, obtiene cuántos renglones tiene la página e inserta un nuevo renglón al final; los renglones se numeran a partir del 0, por lo que si hay n renglones están numerados de 0 a $n - 1$. Después añade dos celdas al renglón y define el código HTML de cada una de ellas.

La función delRow también obtiene el objeto de la tabla; si hay al menos dos renglones borra el último (esto es para no quedarnos con una tabla sin renglones).

Este ejemplo tan sencillo permite ver el poder que tiene JavaScript: nos permite modificar dinámicamente el documento HTML y XML, y de esta forma interactuar con el usuario sin necesidad de comunicarnos con el servidor, porque todo ocurre del lado del cliente (el navegador).

7.4. Emacs

Emacs tiene soporte para manipular HTML, XML y XHTML, incluyendo validación utilizando el DTD. Si planeas utilizar XHTML, sólo tienes que añadir lo siguiente al archivo ∼/.emacs:

```
(add-to-list 'auto-mode-alist '("\\.html\\'" . xml-mode))
```

Al abrir un documento con extensión .html, el modo XML se cargará automáticamente. Para cargar el DTD definido en el tipo de documento, sólo hay que oprimir C–c C–p; lo validas usando el comando C–c C–v. Para definir el programa externo para validar puedes añadir lo siguiente al archivo ∼/.emacs:

```
(custom-set-variables
 '(sgml-xml-validate-command "xmllint --valid --noout %s %s")
 )
```

Aquí se utiliza el programa xmllint, pero se puede utilizar cualquier programa que valide un documento XML contra su DTD.

Si estás editando un documento XML, con C–c C–e añades un elemento y Emacs se encargará de sólo mostrar los elementos válidos (de acuerdo al DTD). Con el comando C–c / cerrará cualquier etiqueta de inicio que siga abierta.

Herramientas para desarrollo

8

8.1. Control de versiones

El control de versiones es el manejo de cambios a la información. Desde siempre este manejo ha sido crítico para los programadores, ya que buena parte del desarrollo de sistemas involucra realizar pequeños cambios y, al siguiente día, deshacer dichos cambios. El control de versiones, sin embargo, va más allá del mundo del desarrollo de software y tiene aplicación en cualquier situación donde se utilicen computadoras para manejar información de amnera colectiva y que cambia constantemente.

Existen muchos sistemas de control de versiones, algunos propietarios y otros libres. Uno de los primeros sistemas de control de versiones y, sin lugar a dudas, el más utilizado es CVS.

Este libro, por ejemplo, fué escrito utilizando el sistema de control de versiones *subversion*, que revisaremos en esta sección.

8.1.1. *subversion*

subversion es un sistema de control de versiones abierto y libre. Esto significa que *subversion* maneja archivos y directorios a través del tiempo. Su forma de operar involucra un árbol de archivos en un repositorio central. Este repositorio funciona como un servidor de archivos ordinario, excepto que *recuerda* cada cambio que se realiza a los archivos y directorios. Esto te permite recuperar versiones anteriores de los datos o examinar la historia de cómo éstos fueron cambiando.

subversion puede acceder a sus repositorios a través de redes, lo que le permite ser utilizado

en distintas computadoras. En este sentido, la habilidad de tener varias personas modificando y trabajando sobre el mismo conjunto de datos, desde sus respectivos lugares, mejora las posibilidades de colaboración. Cada miembro del equipo aporta al proyecto de manera directa, sin intermediarios y, dado que todo está etiquetado por versiones, no se corre el riesgo de sacrificar calidad al no tener un filtro de calidad, ya que si algún cambio reciente es incorrecto, *subversion* permite echar atrás los cambios realizados.

subversion es una arquitectura cliente-servidor. Dado que la instalación y configuración de servicios de red escapa el alcance de este libro, aquí hablaremos brevemente de los aspectos relativos al cliente.

8.1 Clientes gráficos para *subversion*

Existen varios clientes gráficos para manejar tu copia local de *subversion* para KDE. Entre los más conocidos encontramos:

kdesvn Es una interfaz para *subversion* y, a diferencia de otras herramientas, utiliza directamente un API en C desarrollado por *Rapid SVN*, por lo que no revisa la salida del cliente de *subversion* svn, sino que implementa un cliente de *subversion*.

ksvn Es una extensión para Konqueror que le permite manejar copias de trabajo de *subversion*.

8.1.2. Introducción a *subversion*

subversion, CVS y otros sistemas de control de versiones utilizan un modelo conocido como *copiar-modificar-intercalar*. En este modelo, cada cliente de usuario contacta al repositorio central y crea una *copia de trabajo*, que es un reflejo de los archivos y directorios en el repositorio central. Los usuarios trabajan en paralelo, modificando sus copias privadas y, finalmente, intercalan sus copias en una versión final nueva. El sistema de control usualmente asiste el proceso de intercalado, pero al final del día es un humano el encargado de que esto ocurra correctamente.

Existen otros modelos más sencillos en el papel, pero con grandes desventajas a largo plazo. El modelo de *copiar-modificar-intercalar* puede sonar un tanto caótico, pero en la práctica funciona muy bien. Incrementa la productividad, ya que los usuarios pueden trabajar en paralelo, sin esperar a los otros. Cuando trabajan en los mismos archivos, en general, los cambios no se enciman y los conflictos son poco frecuentes y toma poco tiempo arreglarlos.

A continuación te daremos una guía del ciclo usual de desarrollo cuando utilizas un sistema de control de versiones.

Tu copia de trabajo

Usualmente comienzas a utilizar *subversion* haciendo una copia local del proyecto (*checkout*). Esta copia contiene la última revisión de *subversion*, llamada *HEAD*.Obtienesd ls copia con una línea como ésta:

```
% svn checkout http://svn.collab.net/repos/svn/trunk
```

Obviamente la liga cambia para reflejar el repositorio de tu proyecto y existen diversos protocolos para especificarlo. Puede ser vía http, como en el ejemplo, o vía el sistema de archivos local o a través de ssh. En este ejemplo, extraemos el directorio *trunk* del proyecto, que generalmente incluye todos los archivos y directorios, pero puedes seleccionar rutas más detalladas para obtener ciertas partes del proyecto.

Una vez que tienes una copia de trabajo, no tienes nada más que esperar. Puedes comenzar a modificar esos archivos y directorios e iniciar el ciclo típico de trabajo.

Ciclo de trabajo con *subversion*

subversion tiene muchas características y opciones, pero en general te encontrarás siguiendo el siguiente ciclo de trabajo:

1. Actualiza tu copia de trabajo:
   - svn update
2. Realiza algunos cambios:
   - svn add
   - svn delete
   - svn copy
   - svn move
3. Revisa tus cambios:
   - svn status
   - svn diff
   - svn revert
4. Intercala los cambios de otros con tu copia:
   - svn update
   - svn resolved
5. Finalmente, compromete tus cambios al repositorio central. No olvides acompañarlos con un comentario inteligente.
   - svn commit

En la tabla 8.1 explicamos qué hacen los comandos más comunes de *subversion*:

Tabla 8.1 ***subversion***: comandos más importantes

Comando	Descripción
svn add	Usualmente agregas directorios y archivos a tu copia local, realizas tu trabajo de edición y eventualmente se los presentas a *subversion* con este comando.
svn update	Con este comando pones tu copia local al día y *subversion* te avisa de cambios, uno por línea. Cada línea inicia con una letra que te indica algo: **A** Un nuevo archivo en tu copia. **U** Un archivo no sincronizado en tu copia, pero *subversion* lo actualizó en ella, no te preocupes. **D** Un archivo fue eliminado de tu copia. **G** Una archivo no sincronizado en tu copia, porque tú lo editaste en tu copia local. **C** Un archivo no sincronizado en tu copia, pero *subversion* no puede actualizarlo porque hay conflictos que requieren tu intervención.
svn delete	Si un archivo o directorio ya no sirve, así lo borras y avisas a *subversion*.
svn copy	Si el nombre de un archivo es desafortunado, con este comando puedes cambiar el nombre del archivo sin perder su historia.
svn move	Con este comando puedes mover un archivo o directorio a otro lugar en tu copia local y *subversion* se encarga de acompañarlo con su historia.
svn status	Te muestra todos los cambios realizados en tu copia local con respecto a la revisión de la misma.
svn diff	Te muestra las diferencias textuales en uno o varios archivos con respecto a la revisión de tu copia local.
svn revert	Te permite retroceder a otra versión de un mismo archivo o directorio. Usualmente utilizas este comando cuando quieres descartar los cambios que realizaste en tu copia local.
svn update	Sincroniza tu copia local con la versión especificada (por omisión, la más reciente o HEAD) del repositorio central. Este comando requiere que puedas conectarte al servidor de *subversion*.
svn resolved	Una vez que has arreglado un conflicto, con este comando se lo indicas a *subversion*.
svn commit	Una vez que terminas tu trabajo de edición, has intercalado tus cambios y resuelto los conflictos, con este comando le indicas a *subversion* que integre todos tus cambios. Cuando envías tus cambios tienes que acompañarlos de un mensaje que describa tus cambios. Si lo haces en la línea de comandos, tienes que agregar la bandera -m seguida del mensaje entre comillas.

Como ya mencionamos, *subversion* ofrece una plétora de comandos y opciones más, pero el ciclo que hemos presentado es el más común.

8.2. Emacs

¿Ya mencionamos que Emacs es un editor para programadores hecho por programadores? En esta sección te demostramos por qué esto es cierto, ya que revisaremos uno de los más sofisticados modos para programar en Java que existen, JDEE, que son siglas en inglés de *Java Development Environment for Emacs* .

8.2.1. JDEE

Aunque hacemos una cobertura parcial de JDEE, esta sección supone que estás familiarizado tanto con Emacs como con el lenguaje de programación Java.

JDEE permite conectar a Emacs con aplicaciones para desarrollo de terceros, tales como las que provee Sun Microsystems como parte de JDK. El resultado es un ambiente integrado de desarrollo, también conocido como IDE, por sus siglas en inglés. Existen en el mercado muchos IDE comerciales para desarrollar en Java, como Netbeans o Eclipse; JDEE está a la altura de éstos. Entre sus características más importantes se encuentran:

- Editar código con resaltado de sintaxis y sangrías automáticas.
- Completar de manera automática campos y métodos de clases.
- Compilar con brincos automáticos de mensajes de error a la línea de código responsable.
- Generar esqueletos de clases y métodos de manera automática.
- Ejecutar aplicaciones java en un buffer de Emacs de manera interactiva.
- Integrar un depurador con un buffer para comandos interactivos y desplegar de manera automática el archivo/línea mientras *recorres* el código.
- Posibilidad de navegar la documentación de JDK utilizando el navegador de tu elección.
- Navegar a través del código utilizando las herramientas etags de Emacs o bien las estructuras de árbol provistas por otra extensión de Emacs, speedbar.

Instalando JDEE

```
$ smart
install
sun-java5-jdk
```

Antes de poder utilizar JDEE con Emacs debes instalar una serie de extensiones y aplicaciones. Además, por supuesto, debes contar en tu sistema con una implementación de Java; aquí suponemos que ya está instalado JDK.

JDEE no es parte estándar de Emacs y utiliza otras herramientas que no lo son tampoco. Estas herramientas son:

CEDET: en inglés, *Collection of Emacs Development Enviromnent Tools*, que es un proyecto que incluye varios paquetes individuales:

- EIEIO - capa CLOS para Emacs Lisp.

- Semantic - Infraestructura para parsear en Emacs.
- Speedbar - Navegador de... cualquier cosa.
- EDE - Manejador de archivos y generador de Makefile.
- COGRE - *Connected Graph Editor*.

Una vez que obtienes e instalas la última versión, debes agregar lo siguiente a tu ∼/.emacs:

```
(load-file "~/elisp/cedet/common/cedet.el")
(semantic-load-enable-code-helpers)
```

ELIB: Es una biblioteca estándar para programar en Emacs Lisp. Tiene código para:

- Estructuras de datos (colas, pilas, árboles AVL y otras).
- Funciones para manejo de cadenas no incluidas en Emacs estándar.
- Funciones para manejo del mini-buffer no incluidas en Emacs estándar.
- Rutinas para manejar listas de galletas (*cookies*) en un buffer.

Una vez que obtienes e instalas la última versión, debes agregar lo siguiente a tu ∼/.emacs para que JDEE lo encuentre:

```
(add-to-list 'load-path (expand-file-name "~/elisp/elib"))
```

JDEE: Finalmente estamos listos para JDEE, que sigue una ruta similar a las extensiones anteriores y debes poner lo siguiente en tu ∼/.emacs

```
(add-to-list 'load-path (expand-file-name "~/elisp/jde/lisp"))
(require 'jde)
```

A partir de este momento cada vez que visites un archivo relacionado con un proyecto o programa Java, JDEE proveerá sus controles para apoyarte en tu sesión de desarrollo.

Registro y selección de JDK

JDEE descansa en herramientas de desarrollo Java que funcionen en la línea de comandos, como las que ofrece Sun Microsystems como parte del *Java 2 Software Development Kit* (SDK), también conocido como *Java Development Kit* (JDK). Así que antes que puedas utilizar JDEE para compilar, depurar o ejecutar aplicaciones, debes indicarle dónde viven las herramientas adecuadas.

Por omisión JDEE supone que el compilador, depurador y máquina virtual de java se llaman javac, jdb y java respectivamente y que están en un directorio de la variable de ambiente PATH. En la mayoría de los sistemas Linux modernos, las instalaciones de Java se localizan en sitios estándar, por lo que JDEE debe funcionar perfectamente.

En caso de que tengas varias versiones de JDK instaladas y quieras registrarlas, tienes que realizar los siguientes pasos:

1. M-x customize-variable;
2. teclea jde-jdk-registry;
3. haz click en el botón INS;

4. teclea el número de versión de JDK y la ruta al directorio donde está instalado;
5. repite este procedimiento para todas las versiones de JDK en tu sistema;
6. haz click en el botón State; y
7. a continuación selecciona Save for Future Sessions.

A continuación debes seleccionar un JDK:

1. M-x customize-variable;
2. teclea jde-jdk y verás un buffer con todos los JDK que registraste arriba;
3. haz click en el botón junto a la versión que deseas utilizar;
4. haz click en el botón State;
5. y a continuación selecciona Save for Future Sessions;
6. selecciona el botón Finish para deshacerte del buffer.

Editando archivos fuente de Java

Para editar un archivo existente con código Java, carga en un buffer de Emacs el archivo con C-x C-f. Al hacer esto, Emacs asignará el modo jde-mode al buffer de edición. Este modo extiende los modos estándar de Emacs: java-mode y cc-mode, agregando una serie de comandos para compilar, construir, ejecutar y depurar archivos fuente de Java.

Además de esto, JDEE provee un soporte completo para generar documentación en formato HTML para las clases java, a partir de comentarios en el código fuente.

Para insertar el esqueleto de un comentario javadoc para una clase o método en tu programa fuente, posiciona el punto en la primera línea del método o clase y ejecuta el comando: C-c C-v j (jde-javadoc-autodoc-at-line). Por ejemplo, supongamos que tenemos una clase que luce así:

```
public class MiClase
  extends MiSuperClase implements Runnable,java.io.Serializable
{
...
```

y el punto está justo antes de la palabra reservada public. Cuando ejecutas C-c C-v j, JDEE insertará lo siguiente:

```
/**
  * Describe class <code>MiClase</code> here.
  *
  * @author <a href="mailto:Juan.Doe@ciencias.unam.mx">Juan Doe</a>
  * @version 1.0
 */
public class MiClase
    extends MiSuperClase implements Runnable,java.io.Serializable
{
  ...
```

Para generar la documentación del proyecto actual puedes abrir cualquier archivo fuente del proyecto y ejecutar M-x jde-javadoc-make, lo que ejecuta el comando javadoc de JDK para generar la documentación.

Abreviaturas

JDEE te permite utilizar abreviaturas para algunas palabras clave y estructuras de control de java. A continuación te mostramos como utilizarlas.

Cuando escribes una de las abreviaturas definidas, seguidas de un espacio en blanco, JDEE las expande en el buffer, siempre y cuando el modo de abreviaturas esté activado. Para activar/desactivar este modo debes cambiar el valor de la variable jde−enable−abbrev−mode.

Esta extensión de JDEE está basada en abbrev-mode de Emacs, que es un sistema para generar abreviaturas generales.

Dentro de las abreviaturas más comunes encontrarás las que tienen que ver con estructuras de control, por ejemplo, if-then-else. Cuando escribes una de estas abreviaturas seguida de espacio, JDEE la expande al esqueleto de la estructura de control correspondiente.

Por ejemplo, ife se expande a if () { } else { }. Aquí están algunas más de las abreviaturas predefinidas:

Tabla 8.2 JDEE: abreviaturas de estructuras de control

Abreviatura	Enunciado
if	if-then
ife	if-then-else
fori	for (int I=0;I<UL;I++)
foriter	for (Iterator i = c.iterator(); i.hasNext();)
main	main method
tryf	try finally

Completar expresiones, métodos y campos

Emacs provee comandos para completar expresiones incompletas que funcionan para cualquier expresión, pero sólo si éstas existen en un buffer abierto. JDEE provee comandos para completar comandos y funciona en cualquier método Java o nombre de campo que exista en el jde-global-classpath.

JDEE ofrece distintos comandos para completar; todos ellos determinan un conjunto de posibilidades para el nombre del campo o método incompleto en el punto. La diferencia entre ellos, sin embargo, radica en cómo te presentan esas posibilidades. Los comandos se encuentran en la tabla 8.3.

Por omisión, JDEE utiliza jde-complete-menu. Para seleccionar otra función debes asignar el valor de la función deseada a la variable jde−complete−function.

Además de ayudarte a completar, JDEE también incluye un modo para lograr que la tecla Enter cierre las llaves al final de la línea. Para habilitar este modo debes cambiar el valor de la variable jde-electric-return-p.

Tabla 8.3 JDEE: formas para completar campos y métodos

Comando	Descripción
jde-complete	(C–c C–v c) Éste es el comando *estándar* de JDEE y le deja la tarea de completar a otro de los comandos para completar.
jde-complete-in-line	(C–c C–v ⊡). Este comando utiliza la primera opción que encuentra para completar. Si hay más de una opción, te permite recorrerlas, cada una de éstas sustituye a la anterior en el buffer.
jde-complete-menu	Este comando despliega todas las posibilidades en un menú e introduce al buffer la que tú selecciones.
jde-complete-minibuf	Este comando utiliza el mini-buffer para desplegar las posibilidades e introduce al buffer la que tú selecciones.

Generación de código

JDEE provee mecanismos para generación de código. Los hay de dos tipos: *wizards* y basados en patrones. También ofrece comandos para expandir paquetes (jde-import-expand-imports).

Tabla 8.4 JDEE: enunciados *import*

Enlace	Descripción
C-c C-v C-z	(jde-import-find-and-import) genera un enunciado *import* para la clase en el punto. Inserta el enunciado en la cabeza del buffer.
C-c C-v z	(jde-import-all) importa todas las clases que son requeridas en el buffer actual.

Los comandos para importar te permiten convertir un enunciado como:

```
import java.io.*
```

en los siguientes:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
```

y viceversa (jde-import-collapse-imports). JDEE además incluye el comando jde-import-organize que acumula los enunciados *import* del buffer en dos bloques, uno que contiene todos los java y javax y después de una linea en blanco, todos los demás. Dentro de cada bloque ordena alfabéticamente, por ejemplo:

```
import java.io.InputStreamReader;
import java.util.Hashtable;
import javax.swing.JFrame;
import javax.swing.JPanel;

import jmath.LinearSystem;
import jmath.Test;
```

Finalmente, con jde-import-kill-extra-imports borras los import que no son necesarios.

JDEE incluye además una serie de generadores de código o *wizards*, que se muestran en la tabla 8.5.

Tabla 8.5 JDEE: generadores de código, *wizards*

Wizard	Descripción
Sobrecarga de operadores	(jde-wiz-overrided-method). Un método que sobrecarga otro heredado por la clase donde está el punto.
Interfaz	(jde-wiz-implement-interface). Implementación de una interfaz en la clase que contiene al punto.
Delegado	Métodos que delegan tareas a una clase específica.
Clase abstracta	Implementación de los métodos heredados de una clase abstracta por la clase que contiene al punto.
Get/Set	(jde-wiz-get-set-methods) Métodos *get* y *set* para los campos privados de la clase que contiene al punto.

Existen otras maneras de generar código, como clases patrón vacías, pero éstas no tienen un único comando asociado y debes ejecutarlas desde el menú de JDE. Por ejemplo, para crear un buffer con una clase pública genérica de Java, puedes utilizar Files ⇒ JDE New ⇒ Class.

Compilando programas Java

Para compilar debes ejecutar C-c C-v C-c (jde-compile), lo que ejecutará el comando javac en el buffer actual y el resultado aparecerá en un nuevo buffer de compilación.

Si ya existía un buffer de compilación, el comando reutilizará el buffer; si no, creará uno nuevo. Por omisión, si no existen errores en la compilación, JDEE borra el buffer de compilación dos (2) segundos después de terminada la compilación.

El modo mayor de este buffer es compilation-mode, que te permite moverte rápidamente a la fuente de cada error, simplemente haciendo click o presionando Enter en el buffer de compilación.

JDEE utiliza una serie de variables para controlar la manera de compilar tu programa; para cambiar la forma estándar debes utilizar la secuencia de menús y submenús de JDEE: JDE ⇒ Options ⇒ Compile.

Si utilizas JDE ⇒ Build, entonces JDEE llamará un programa para construir tu aplicación. Por

omisión utiliza el comando make de Linux, pero puedes configurarlo para que utilice la herramienta de compilación Ant, del grupo Apache, o una función especial que tú definas. Las funciones predefinidas son: jde-make y jde-ant-build y, para utilizar Ant, utiliza lo siguiente en tu ∼/.emacs:

```
(setq jde-build-function '(jde-ant-build))
```

Existen muchas variables para controlar el comportamiento tanto de make, como de Ant, aunque en la mayoría de las distribuciones de Linux no debe ser necesario que cambies los valores por omisión de estas variables.

Ejecución de aplicación Java

JDEE te permite ejecutar tus aplicaciones Java desde un subproceso de Emacs. Puedes ejecutar varias aplicaciones de manera concurrente, pero sólo un ejemplar de cada aplicación en un momento dado. JDEE te mostrará tanto la salida como el error estándar de tu aplicación en un buffer de Emacs y, a través de estos buffers, puedes interactuar con la aplicación si ésta es interactiva.
Para ejecutar la aplicación utiliza C-c C-v C-r (jde-run).

Por omisión en JDEE, el comando para ejecutar aplicaciones supone que la clase en el buffer actual contiene el método main de tu programa. Si éste no es el caso, tienes que indicarle cuál es la clase principal, asignando el nombre adecuado a la variable jde-run-application-class.

Con la variable jde-run-working-directory puedes indicarle un directorio distinto al actual para ejecutar tu aplicación.

Navegando excepciones

Si ocurren excepciones durante la ejecución de tu programa, la máquina virtual entrega como salida un registro de excepción. Este registro aparece en el buffer de ejecución. Para ver la línea de código correspondiente a un punto en el registro de excepción, puedes hacerlo con el botón derecho del ratón.
Para navegar el registro de excepción hay dos combinaciones de teclas: C-c C-v C-[y C-c C-v C-].

JDEE ofrece un excelente soporte para trabajar con *applets*, pero por razones de espacio no lo cubriremos en este libro.

Depurando aplicaciones

JDEE ofrece dos opciones para depurar aplicaciones Java:

- Una interfaz en Emacs para jdb, el depurador en línea de comandos que se distribuye con JDK.
- JDEbug, un depurador desarrollado específicamente para ser utilizado con JDE.

JDEbug ofrece mejores características para depurar, pero requiere JDK 1.2 o más reciente, por lo que si tu aplicación utiliza una versión anterior, sólo podrás utilizar jdb. En este trabajo revisamos exclusivamente JDEbug.

JDEbug te permite ejecutar un programa Java paso por paso y despliega y modifica el estado interno del programa. Entre sus características principales contamos:

- Depuración a nivel de fuente. JDEbug mantiene un apuntador a la linea de código que se está ejecutando conforme avanzas. Te permite además asignar puntos de evaluación en los buffers fuente.
- Despliega de manera automáticas el valor de las variables.
- Navegador de objetos. Puedes expandir el despliegue de la variable local para mostrar los campos del objeto a cualquier nivel de profundidad.
- Te permite navegar la pila de ejecución.
- Puedes depurar varios procesos simultáneamente en la misma sesión.
- Evaluación de expresiones. Puedes evaluar cualquier expresión que sea válida en el punto actual de suspensión.

Por omisión JDEE está configurado para ofrecerte una interfaz de jdb, por lo que debes indicarle que deseas utilizar JDEbug. Esto lo logramos asignando el valor JDEbug a la variable jde-debugger:

```
(setq jde-debugger '("\command{JDEbug}"))
```

JDEbug se apoya en la biblioteca JPDA, que en Linux se distribuye como parte de JDK, por lo que no es necesario hacer nada más.

Iniciando el depurador

Para iniciar el depurador selecciona Processes ⇒ Debugger del menú de JDEbug, con lo que se inicia la ejecución del depurador. Para depurar una aplicación debes:

- Iniciar la aplicación dentro del depurador. Adelante te decimos cómo.
- Añadir el depurador a tu aplicación si es que ésta fue ejecutada fuera del depurador.
- Poner el depurador en modo de escucha e iniciar la aplicación fuera de éste.

Es muy importante que cuando hayas terminado de utilizar el depurador, termines su ejecución seleccionando Exit Debugger del menú JDEbug. De otra manera, aun cuando termines la ejecución de Emacs, pueden quedarse procesos (máquinas virtuales) huérfanos ejecutados por el depurador.

Iniciando procesos

JDEbug provee dos comandos para iniciar procesos:

1. JDEbug ⇒ Processes ⇒ Launch Process, que ejecuta un ejemplar de la aplicación señalada por jde-run-application-class y, en caso de que su valor sea nil, la aplicación correspondiente a la del buffer actual de java.
2. JDE ⇒ Debug App; este comando realiza las siguientes acciones:
   a) Inicia el depurador de ser necesario.
   b) Lanza la aplicación.
   c) Asigna todos los puntos de revisión que hayas especificado con anterioridad.
   d) Ejecuta la aplicación.

Nota: si no asignaste ningún punto de revisión, la aplicación será ejecutada sin interrupción hasta terminar.

Cualquiera de estos comandos que ejecutes, dividirá la pantalla de Emacs en dos y desplegará una nueva ventana para las variables locales.

La pantalla superior de Emacs muestra el código fuente, la siguiente muestra el buffer fuente o el archivo donde pusiste puntos de revisión y, finalmente, la ventana del fondo muestra los mensajes del depurador para el proceso recién lanzado.

En este momento, si el proceso fué iniciado exitosamente y no se se ha ejecutado hasta terminar, puedes depurar la aplicación.

8.2.2. *psvn*

8.1 PCL-CVS frente para CVS

Otro modo muy importante para controlar versiones es PCL-CVS, que es parte estándar de Emacs. Este modo sirve como frente para manejar proyectos a través de CVS.

CVS es el sistema controlador de versiones por omisión. Fue uno de los primeros sistemas utilizados y, probablemente, aun hoy día sea el más utilizado en el mundo del software libre. Sin embargo, en nuestra opinión, *subversion* es un sistema más versátil y tiene varias ventajas sobre CVS.

Este sistema es una interfaz para el control de versiones *subversion* (ver la sección 8.1.1); sin embargo, *psvn.el* no es parte estándar de Emacs por lo que tienes que descargarlo e instalarlo, siguiendo las instrucciones en la sección 4.3.18 y agregar lo siguiente a tu ∼/.emacs:

```
(require 'psvn)
```

Una vez cargado el programa, puedes ejecutarlo con M-x svn-status; te preguntará en el mini-buffer por el directorio con tu copia de trabajo. Por omisión te mostrará el directorio del archivo actual. Este comando ejecuta svn status en el directorio que tecleaste y te muestra un buffer con el resultado que, como notarás, se parece mucho a la salida del comando en una terminal, pero a diferencia de ésta, en este buffer cuentas con una serie de enlaces de teclas para ejecutar rápida y cómodamente comandos de *subversion*.

Tabla 8.6 Emacs: comandos de psvn

Enlace	Descripción
↑, p, C-p	(svn-status-previous-line) Te lleva a la línea anterior.
↓, n, C-n	(svn-status-next-line) Te lleva a la siguiente línea.

Continúa en la siguiente página

Tabla 8.6 Emacs: comandos de psvn *Continúa de la página anterior*

Enlace	Descripción
Enter	(svn-status-find-file-or-examin-directory) Abre el archivo (directorio) en la línea actual.
x	(svn-status-update-buffer) Actualiza la vista en el buffer con respecto a tu revisión local.
U	(svn-status-update-cmd Ejecuta svn update en el directorio actual.
a	(svn-status-add-file) Añade el archivo al repositorio.
R	(svn-status-mv) Ejecuta el comando svn move en el archivo (directorio) actual.
D	(svn-status-rm) Ejecuta el comando svn delete en el archivo (directorio) actual.
C	(svn-status-cp) Ejecuta el comando svn copy en el archivo (directorio) actual.
r	(svn-status-revert) Ejecuta el comando svn revert en el archivo actual.
=	(svn-status-show-svn-diff) Ejecuta el comando svn diff del archivo (directorio) actual con la versión de la revisión en tu copia de trabajo.
c	(svn-status-commit) Compromete el cambio en el archivo (directorio) con el repositorio actual.
m	(svn-status-set-user-mark) Pone una marca en el archivo o directorio. Esto te permite poner marcas a varios archivos y el siguiente comando se ejecutará sobre todos los archivos marcados. Por ejemplo, añadirlos, borrarlos, comprometer los cambios y otros.
u	(svn-status-unset-user-mark) Elimina la marca del archivo (directorio) en la línea actual.

Extendiendo tu ambiente 9

La capacidad de extender y acoplar el sistema a tus necesidades es una de las principales características de UNIX en general. Linux, por supuesto, hereda esta característica y en este capítulo te mostraremos una de las formas más poderosas de extender tu ambiente con la programación en shell.

También te mostramos cómo extender Emacs a través de su lenguaje de programación de extensión, Emacs-Lisp. De hecho, la gran mayoría de subsistemas y paquetes para extender Emacs que revisamos en este libro, por ejemplo JDEE (sección 8.2.1) o ERC (5.15.4), son programas escritos en Emacs-Lisp. Entonces, lo que sucede cuando cargas uno de estos programas en Emacs, es que éste actúa como máquina virtual e *interpreta* (ejecuta) el programa.

9.1. Programación en shell

El shell de UNIX cuenta con una característica muy importante que es la capacidad de ser programado. La programación del shell se hace por medio de los llamados guiones de shell, los cuales básicamente toman una serie de instrucciones y las ejecutan en un nuevo proceso de shell. Los guiones de shell son archivos que contienen instrucciones para el shell que llevan a cabo una tarea específica, por ejemplo automatizar la entrega de ciertos correos a un grupo de personas, y en general se utilizan para automatizar todo tipo de tareas.

En un guión de shell las instrucciones se ejecutan una después de otra en el orden dado por el guión. Además de éstas, los guiones de shell también pueden recibir parámetros de la misma manera en que los programas lo hacen, pueden definir variables para guardar ciertos valores que

sean de interés para la ejecución del programa y cuentan con estructuras de control de flujo, algunas de las cuales veremos más adelante.

En un guión de shell también podemos tener *comentarios*, que son mensajes para los que lean el guión (en particular, nosotros mismos después de algún tiempo). Los comentarios son ignorados por el shell al momento de ejecutar, así que no hacen daño y sirven para dar mayor claridad a nuestro guión de shell. Los comentarios son todo lo que siga al carácter # hasta el fin de la línea.

Los guiones de shell pueden ser ejecutados de manera *autónoma* simplemente cambiando los permisos de ejecución y poniendo un comentario especial en la primera línea del archivo. Esta línea luce algo así `#!/bin/bash` y le indica al shell que debe de usar el programa `/bin/bash` para procesar el resto del archivo.

A continuación te mostramos un guión de shell muy sencillo:

```
#! /usr/bin/bash
echo hola;
```

Actividad 9.1 *Crea un guión de shell que ejecute un* `ls -la`*, lo redirija a un archivo llamado* `listado.dir` *y despliegue un mensaje al usuario, informándole de la creación de este archivo; da permisos de ejecución y ejecútalo un par de veces en directorios distintos.*

9.1.1. Variables

Los guiones de shell utilizan variables como almacenes para ciertos valores, por ejemplo el mensaje a desplegar o los nombres de los archivos a borrar. Para las variables se utilizan palabras formadas únicamente de mayúsculas[1] y se les asigna valor por medio del operador =; por ejemplo `MENSAJE="saluditos por la mañana"`; después, para usarlas recuperando su valor, precedemos el nombre de la variable con el carácter '$', como en la siguiente línea:

```
% echo $MENSAJE
```

Entonces el programa anterior, usando variables para el mensaje, quedaría así:

```
#! /usr/bin/bash
MENSAJE=hola; echo $MENSAJE;
```

Actividad 9.2 *Modifica el guión de shell anterior de manera que uses variables para el mensaje al usuario así como para el nombre del archivo a donde se dirige la entrada.*

9.1.2. Preparación

Hasta ahora has aprendido cómo hacer algunos programas sencillos, hechos con base en los comandos aceptados por el shell estándar de Linux, bash, y conectando la salida de algunos con la

[1] Esto es parte de la tradición; se pueden usar minúsculas también.

entrada de otros. Ahora estás capacitado para emprender labores un poco más complejas. Aprenderás cómo hacer programas ejecutables por bash, más grandes y que ofrecen servicios más útiles.

Antes que nada y con vistas a organizar decorosamente tu actividad, deberás crear en tu directorio personal un subdirectorio donde puedas guardar todos tus programas ejecutables de uso frecuente. Esto, además de la evidente ventaja de organizar tu labor, te proporciona la posibilidad de incluir este subdirectorio como parte de tu variable de ambiente $PATH. Procedamos en orden. Lo que tenemos que hacer es:

1. Crear un directorio llamado ∼/bin: cd; mkdir bin.
2. Agregar a nuestra variable $PATH el directorio que acabamos de crear. Esto es, edita el archivo ∼/.bashrc y agrega al fin de la línea que comienza: PATH=, lo siguiente: :/$HOME/bin.

Ahora todos los programas que hagas y coloques en ese subdirectorio podrán ser llamados desde cualquier otro directorio en el que estés. El sistema se encargará de ejecutarlos desde tu directorio bin. Por supuesto todos los programas deberán tener permiso de ejecución (recuerda el uso de chmod).

Actividad 9.3 *Despliega la variable de ambiente* $PATH *luego de editar el archivo* ∼/.bashrc. *Compara el resultado con la que acabas de poner en el archivo. ¿Que ocurrió?*

En general, todos los programas que elabores en esta sección deberán almacenarse en tu directorio bin.

9.1.3. Vigilando el espacio en disco

Nuestro primer programa, como todos los que seguirán, estará enfocado a resolver un problema, a facilitarte la vida de alguna manera. Algo con lo que todo usuario de un sistema compartido debe aprender a convivir es con los demás usuarios. Todos comparten un mismo disco duro, un mismo procesador, una misma memoria; es necesario regular el uso de estos recursos para garantizar su disponibilidad a todos los usuarios. Es por ello que en muchos sistemas de cómputo compartidos se establecen políticas generales de uso de recursos. Por ejemplo, se establecen cuotas máximas de uso de disco. Mediante este procedimiento cada usuario del sistema tiene un límite máximo de espacio disponible, es decir, la cantidad de información guardada en su directorio personal no deberá exceder nunca este límite; en caso de que se pretenda rebasar el límite, el sistema lo impide: es como si el disco duro se hubiese acabado. El usuario se entera de esto porque se le presenta un mensaje como *Disk quota exceeded*.

Para que nunca te suceda esta desagradable situación y se te impida guardar información importante, es necesario que mantengas constantemente vigilado el monto de disco que utilizas. Sería conveniente entonces poseer un listado de cuánto espacio de disco ocupa cada uno de tus subdirectorios para que, en caso necesario y tomando en consideración lo que sea más relevante, eliminar información que ya no sea necesaria.

Para poder hacer esto necesitas un programa que te permita conocer cuánto espacio de disco ocupa un directorio y te lo reporte en unidades de uso cotidiano. El comando que hace esto en

UNIX es du (*disk usage*). Puedes saber cuanto espacio ocupa tu directorio actual y cada uno de sus subdirectorios con el siguiente comando: du -k .[2].

Actividad 9.4 *Logra que la salida de* du *se muestre paginada.*

Actividad 9.5 *Logra que* du *reporte sólo el tamaño del directorio actual, pero sin desglosar por subdirectorios.*

Lo malo de hacer du -k . es que el listado de la cantidad de disco ocupada sale desordenado. Si tienes muchos subdirectorios no es evidente cuáles son los que más espacio ocupan y, por tanto, a cuáles deberías prestar más atención. Pero afortunadamente existe el comando sort, que permite ordenar aquello que recibe como entrada. Como has hecho en el pasado, puedes pasar la salida del comando du como entrada del comando sort y rematar esto pasando la salida de sort como entrada de more (o less si lo prefieres). sort ordena creciente y quisieras (aunque en gustos se rompen géneros) que la salida fuera ordenada decreciente para que pudieras fijarte quizás en los 3, 5 o 10 subdirectorios más grandes fácilmente; además, quieres que para ordenar use los valores *numéricos* entregados por du. Esto último puede parecer evidente pero no lo es. Hay dos maneras de ver un 423: como el número 423 o como la secuencia de tres símbolos "423", una cadena de carácteres. Visto de la primera forma, 423 es mayor que 52; pero desde el segundo punto de vista, la cadena "423" es menor que "52" porque, en orden lexicográfico el "5" está después del "4".

En síntesis, y agregando algunas líneas explicativas al programa, tienes lo siguiente:

```
1  #!/bin/bash
2  #
3  # Reporta el espacio en Kbytes ocupado por cada
4  # subdirectorio del directorio actual ordenando de mayor a
5  # menor (orden inverso para sort).
6  #
7  echo "Espacio ocupado en disco (en Kbytes)"
8  echo "por el directorio actual: `pwd` y subdirectorios"
9  du -k . | sort -nr | more
```

Nota que está entre entre comilla invertida (acento grave, que está en la esquina superior izquierda del teclado, junto al "1") el comando pwd en la segunda línea de mensaje – línea (8) – . Cuando se pone un comando entre ese tipo de comilla, al momento de ejecutar la línea que lo contiene se reemplaza el comando por su salida, es decir la ejecución de la línea (8),

```
echo "por el directorio actual:`pwd` y subdirectorios,
```

produce:

```
por el directorio actual: /home/users/jose y subdirectorios
```

Actividad 9.6 *Haz que el monto de espacio ocupado se despliegue en* mega bytes *en vez de en* kilo bytes.

Actividad 9.7 *Haz que la salida de* sort *sea en orden creciente y sea posible hacer paginación.*

[2] El . se refiere al directorio actual.

9.1.4. Listado de directorios

Por supuesto habrás notado que el listado de los elementos de un directorio incluye tanto subdirectorios como archivos y que en dicho listado las entradas aparecen ordenadas alfabéticamente. Esto hace que subdirectorios y archivos aparezcan mezclados en la lista, lo que dificulta distinguirlos, sobre todo si tenemos una larga lista. Sería bueno tener un programa que liste sólo los subdirectorios de un directorio dado.

Entonces queremos hacer un programa al que podamos decirle el nombre, o en general, la ruta de algún directorio y nos muestre en la pantalla un listado del nombre de todos los subdirectorios que se encuentran contenidos en él. Sería bueno que ese nombre o ruta del directorio pudiera darse al programa del mismo modo en que se dan los parámetros de entrada de los programas que conocemos, es decir, poniéndolos inmediatamente después del nombre del programa que se quiere ejecutar. Cuando llamamos a ls por ejemplo, podemos decirle la ruta de algún directorio del que deseamos ver el listado. Por ejemplo:

```
% ls /usr/include
```

te muestra el listado del directorio /usr/include que pusiste junto al nombre del programa.

bash tiene prevista la situación en la que se dan *parámetros* de entrada a los programas. En esencia sólo se requiere guardarlos en algún lugar accesible al programa y ponerle nombre a dicho lugar para poder referirse a lo que está guardado allí desde el programa. Además, el nombre de estos lugares deberá de ser un estándar para que cualquier programador sepa cómo referirse a él. En casa todos los miembros de la familia pueden acceder a las llaves de la puerta porque están en un lugar estándar que todos conocen y al que todos hacen referencia cuando le piden las llaves a alguien más. A estos lugares con nombre para almacenar valores se les llama variables. Ya has tenido experiencia con variables de ambiente (como PATH) y ahora estamos hablando de las variables de tus programas (en particular las variables de ambiente son variables de tus programas).

Pues bien, existen ciertas variables que existen siempre automáticamente en todos los programas en shell y que se refieren a los parámetros de entrada de un programa:

Tabla 9.1 Programación en shell: variables automáticas

Variable	Descripción
$0, $1, $2, ..., $n	$0 es el nombre del programa mismo que invocaste, es decir, lo primero que pones en la línea de llamada. $1 es el primer parámetro que se pone luego del nombre del programa (v.gr.. en ls -l, -l sería $1). Análogamente $2, . . . , $n son los siguientes parámetros.
$#	es el número de parámetros que pones frente al nombre del programa, es decir no se cuenta $0.

Continúa en la siguiente página

Tabla 9.1 Programación en shell: variables automáticas *Continúa de la página anterior*

Variable	Descripción
$?	es el resultado del último comando ejecutado. Todos los programas que se ejecutan pueden regresar un resultado; más adelante abordaremos el tema.

Nota además que existe un medio para contar cuántos parámetros de entrada se le dieron a un programa. Basta con observar el valor que posea la variable $# para saber cuántas palabras (separadas por espacios) se dieron frente al nombre del programa cuando éste fue llamado a ejecutarse.

En nuestro programa requeriremos saber cuántos parámetros de entrada se dieron; si existe algún parámetro, ése debe ser la ruta de acceso al directorio del que se desea conocer el listado de subdirectorios. Llamemos a este directorio el *directorio objetivo*. Si no existe parámetro alguno entonces supondremos que el directorio objetivo es el actual. En caso de que sí exista el parámetro, entonces deberemos cambiarnos al directorio objetivo especificado por dicho parámetro y listar sólo el nombre de sus subdirectorios.

Es decir, el esquema general de lo que debemos hacer es el siguiente:

```
1  Si se dio la ruta del directorio objetivo
2     nos cambiamos a ese directorio.
3  Si no,
4     no nos movemos.
5  Ya parados en el directorio objetivo, para cada entrada del listado
6           de directorio
7        Si la entrada es el nombre de un subdirectorio,
8              la mostramos.
9        Si no,
10             nos vale.
```

Nóta que en las líneas 1 y 7 debes tomar decisiones, debes ser capaz de preguntar si algo ocurre o no. Esto lo puedes hacer en bash usando la construcción:

```
if [ condición ]
then
  # instrucciones que se ejecutan si la condición es verdadera
else
  # instrucciones que se ejecutan si la condición es falsa
 fi
```

Existen dos maneras de escribir la condición, una es poniéndola entre corchetes (“[“ y “]”) y la otra usando explícitamente el comando test[3]. test −d <algo> pregunta si <algo> es un subdirectorio; si en vez de −d se pone −f entonces pregunta si <algo> es un archivo (file).

[3] En realidad [condición] es una manera implícita de usar el comando test .

Además debemos hacer algo para cada entrada del listado de directorio, como dice nuestro esquema general. Esto es posible usando la construcción for del shell (ver man bash):

```
for <variable de índice> in <conjunto de valores>
do
   #instrucciones que hay que repetir
done
```

Esto hace que las instrucciones entre el do y el done se repitan tantas veces como elementos tenga el conjunto especificado a continuación de la palabra in. En esencia, lo que hace el for es ponerle a la variable de índice el primer valor de la lista, luego ejecutar las instrucciones a repetir, cambiar el valor de la variable de índice poniéndole el segundo del conjunto y repetir las instrucciones; y así sucesivamente hasta que la variable adquiera el último valor del conjunto; ésa será la última vez que se ejecuta el ciclo.

A continuación se muestra el código del programa que queremos:

```
1 #!/bin/bash
2 #
3 # Entrega un listado de aquellas entradas del directorio que
4 # son  subdirectorios.
5 #
6 # Si hubo un parámetro en la linea de comandos
7 # es el directorio del que deseamos los subdirs.
8 if [ $# -ne 0 ]; then
9  cd $1
10 fi
11 # si no, queremos el listado del directorio actual
12 echo "Subdirectorios en `pwd`"
13 for i in `ls -aA`; do
14   if test -d $i; then
15     ls -ld $i
16   fi
17 done
```

Actividad 9.8 *¿Por qué se le puso la opción* d *al comando* ls *(línea 15)?*

Actividad 9.9 *¿Por qué se le pusieron las opciones* aA *al comando* ls *(línea 13)?*

Actividad 9.10 *¿Para qué sirven las líneas 13–17?*

Actividad 9.11 *Analiza la línea 8 del programa. ¿Qué ocurre si el usuario, tontamente, da más de un parámetro? ¿Como harías para restringir más, si es posible, la entrada? (sugerencia:* man test*).*

9.1.5. Recorridos recursivos

El problema de borrar basura: planteamiento

Ya sabes cómo hacer para conocer el espacio en disco que ocupas en tu directorio personal. Esto te provee de un medio para saber cuáles de tus subdirectorios son más grandes, lo que puedes usar como un criterio para decidir qué borrar o comprimir y qué no, con el fin de reducir el espacio ocupado.

Pero si meditas un poco en el proceso que deberás llevar a cabo para depurar tu directorio, te darás cuenta de que puede no ser agradable. Habría que navegar por todos los subdirectorios buscando archivos inútiles y borrándolos, teniendo cuidado de no borrar los que no debes.

Es posible establecer criterios generales, basados en los nombres de los archivos, para decidir qué borrar y qué no. Ya conoces, por ejemplo, los archivos de respaldo de Emacs, esos que terminan con $\sim$ y que suelen ser muy útiles cuando aún no se obtiene la versión final de un archivo que está en proceso de edición, pero que una vez que dicha versión final se ha obtenido dejan de serlo. Con frecuencia, archivos como éstos se quedan por allí ocupando espacio innecesariamente. Hay otros casos similares como los archivos de bitácora (.log) o auxiliares (.aux) que son generados por TEXy LATEX.

Podrías pensar entonces en hacer un programa que descienda por todo el árbol de tu directorio personal eliminando todos los archivos que satisfagan los criterios establecidos en su nombre. Eso automatizaría en buena medida el proceso de depuración de espacio y reduce significativamente la probabilidad de que te equivoques borrando algo que no querías borrar (por supuesto que pueden seguir ocurriendo errores indeseables: si en algún momento deseas conservar un archivo cuyo nombre cumple con los criterios de borrado, deberás cambiarle de nombre para no borrarlo accidentalmente en el futuro).

El patrón de descenso

Para resolver el problema que te ocupa tienes que ser capaz, primero, de poder descender por un directorio y todos sus subdirectorios. ¿Cómo haces esto? Conceptualmente hablando, lo que quieres es:

1. Haces lo que deseas hacer en el directorio actual (lo que quieres hacer aun cuando éste sea el único directorio presente).
2. Te fijas en cada entrada del listado del directorio.
3. Si la entrada corresponde a un subdirectorio te introduces a él y haces lo mismo que estabas haciendo en el nivel anterior (haces lo que deseas... fijarte en cada entrada...).
4. Si le entrada no es un directorio quizás haces algo más con ella.
5. Cuando terminas con cada una de las entradas del directorio sales al directorio inmediato anterior.

El paso número 2 significa que haces *exactamente* lo mismo que estabas haciendo en el directorio superior; es decir, ejecutas cada una de las acciones que estabas llevando a cabo, en el mismo orden; esto es, ejecutas el mismo programa. Cuando un programa, como parte de sus instruccio-

nes, ordena la ejecución de sí mismo, el programa recibe el apelativo de *recursivo*. Imagina que tu programa se llama paseo_rec. Entonces el esquema general de lo que quieres hacer es algo así:

```
1  Hacemos lo que deseamos hacer, el objetivo de nuestro programa
2  Para cada entrada del directorio
3  Si la entrada es un subdirectorio
4        Entramos al subdirectorio
5  Ejecutamos paseo_rec a partir del directorio actual
6  Salimos del directorio al inmediato superior
7  Si no es un subdirectorio
8     Decimos su nombre
```

Deseamos repetir al proceso de las líneas 3 a 8 para cada entrada del directorio. Esto se dice for i in ‘ls‘ es decir *para cada elemento del listado producido por el comando ‘ls‘*. Dado que ls entrega una por una las entradas del directorio, esa i que usas sirve para referirse, uno por uno, a cada elemento del listado de directorio. Entonces, una vez que *estás parados* en la entrada i del directorio, puedes preguntar si esa i corresponde a un subdirectorio o a un archivo. Esto es posible hacerlo con los comandos if y test, tal como lo hiciste en el programa que lista directorios. Así:

```
1 #!/bin/bash
2 # Programa recursivo para bajar por todo un árbol de directorios
3 echo "Directorio actual: " ‘pwd‘
4 for i in ‘ls‘; do        #Para cada entrada del directorio
5   if test -d $i; then  # si la entrada es de un directorio
6     cd $i                        # nos cambiamos a éste
7     $HOME/bin/paseo_rec #Llamada recursiva
8     cd ..
9   elif test -f $i; then  #si no, si es un archivo
10    echo " archivo: $i"
11  fi
12 done
```

El problema de borrar la basura revisitado

Ahora puedes usarlo para tus fines, hacer un programa que borre todos los archivos que terminen con: ∼, aux o log. Como realmente no te interesa hacer algo con las entradas del directorio que no son directorios, eliminas el trozo de código de las líneas9–11 y en vez de la línea 4 pondrás también las líneas que necesitas para eliminar los archivos. El resultado es el código que se encuentra en el listado 9.1 de la siguiente página – se está suponiendo que el archivo donde este programa se encuentra se llama borramugres.rec.

Nota que en la línea 14 se llama al programa borramugres.rec, que se supone se encuentra en el subdirectorio bin del directorio personal del usuario; así que si cambias el nombre del programa o su ubicación deberás modificar esa línea para que pueda encontrarse a sí mismo.

Actividad 9.12 *Modifica el programa para que pregunte si realmente desea borrar cada archivo de respaldo de Emacs, (sugerencia:* man rm*).*

Código 9.1 Borrado recursivo de basura

```
1 #!/bin/bash
2 # Programa recursivo para borrar basura
3 #
4 echo "Directorio actual: " `pwd`
5 rm -f *.log
6 rm -f *.aux
7 rm -f *~
8 rm -f .*~
9 for i in `ls`   # Para cada entrada del directorio
10   do
11   if test -d $i  # si la entrada es de un directorio
12   then
13     cd $i           # entra al directorio
14     $HOME/bin/borramugres.rec
15     cd ..
16   fi
17 done
```

Un *grep* recursivo

Ya conoces el programa grep que te permite, en particular, buscar presencias de una cadena cualquiera en conjuntos de archivos. En muchas situaciones es deseable conocer no sólo en cuáles archivos de un directorio aparece la cadena, sino en cuáles archivos de sus subdirectorios, en cuáles de los subdirectorios de éstos y así sucesivamente. Es simple pensar en una situación así; por ejemplo, si quisieras saber cuántos y cuáles de los archivos contenidos en un directorio o sus subdirectorios son programas para bash podrías buscar la cadena "/bin/bash" en todos los archivos.

Lo que quieres es hacer un grep recursivo que busque una cadena en todos y cada uno de los archivos contenidos en un directorio y sus subdirectorios. Tienes que hacer que tu grep recursivo (al que llamaremos rgrep) busque la cadena deseada en todos los archivos del directorio actual y luego proceda a la manera de paseo_rec, metiéndose a cada subdirectorio que se encuentra y llamándose a sí mismo. Cuando la entrada del directorio sea un archivo no haces nada, como puedes ver en el listado 9.2 de la siguiente página.

Actividad 9.13 *¿Qué significan las opciones* Hns *de* grep *que se usan en* rgrep *en la línea 11?*

Código 9.2 grep recursivo

```
1 #!/bin/bash
2 #
3 # Hace un grep recursivo a mano (es decir emulando
4 # egrep -r). Como hacían los hombres de mi época.
5 #
6 #
7 if [ $# -ne 1 ]
8 then
9   exit -1
10 fi
11 grep -Hns $1 * | more
12 for i in `ls`
13 do
14   if test -d $i
15   then
16     cd $i
17     echo ""
18     echo "estoy en " `pwd`
19     $HOME/bin/rgrep $1
20     cd ..
21   fi
22 done
```

Actividad 9.14 *¿Qué significa la opción* -r *de* grep *(o mejor dicho de* egrep*, que ocupó el lugar del antiguo* grep*)?*

Cambiando permisos, propietario y grupo recursivamente

Podrías utilizar también el esquema general de programa recursivo para mejorarlo, ahora haciendo otra labor: la de cambiar permisos, propietario y grupo de todos los archivos de un directorio y sus subdirectorios.

Quieres un programa, llamado chmogr, que reciba como parámetros de entrada en la línea de llamada:

1. los permisos,
2. el propietario y
3. el grupo,

que quieres poner a todos los archivos y subdirectorios a partir del directorio actual.

Ya sabes usar los comandos chmod, chown y chgrp que son en los que nos apoyaremos para realizar nuestra labor. Pero hay algo adicional que considerar: no es posible entrar a un subdirectorio si éste no posee permiso de ejecución (curioso ¿no?). Así que en tu programa debes tener esto en cuenta y asegurarte de poder entrar a todos los subdirectorios poniéndoles permiso de ejecución.

Seguramente ya has usado varias veces el comando ls con la opción -l y habrás visto la salida que produce. A diferencia de ls a secas, ls -l despliega una línea por cada archivo. Si observas esta salida, en el extremo izquierdo de cada archivo están los permisos de acceso del mismo. En la sección 3.2.1 revisamos con detalle el sistema de permisos de UNIX y aquí vas a utilizar esta información para acceder a cada subdirectorio.

```
1 #!/bin/bash
2 #
3 # Programa para cambiar permisos, el propietario y el
4 # grupo de todos los archivos y directorios recursivamente
5 #
6 if [ $# -eq 3 ]
7 then
8   for i in `ls`
9   do
10    if test -d $i
11    then
12      chown $2 $i
13      chgrp $3 $i
14      # le ponemos permiso de entrar temporalmente
15      chmod +x $i
16      cd $i
17      echo "Directorio actual: " `pwd`
18      # OJO aquí se pone la ruta completa absoluta
19      # para acceder a este programa
20      $HOME/bin/chmogr $1 $2 $3
21      cd ..
22      # le ponemos los permisos que nos dijeron
23      chmod $1 $i
24    elif test -f $i
25    then
26      echo "   Archivo: "$i
27      chown $2 $i
28      chmod $1 $i
29      chgrp $3 $i
30    fi
31  done
32 else
33  echo "USO: $0 [perm] [prop] [grupo]"
34  echo "Cambia permisos, propietario y grupo recursivamente"
35  echo "Jos+\'e+ Galav+\'i+, Facultad de Ciencias, UNAM."
36 fi
```

Actividad 9.15 *El programa recursivo de cambio de permisos propietario y grupo fue hecho hace mucho cuando aún no existían ciertas extensiones de los comandos tradicionales* chown, chmod *y* chgrp. *¿Cómo harías actualmente un programa con la misma funcionalidad pero más sencillo? (sugerencia:* man chown*)*

Actividad 9.16 *Elabora, a partir de* paseo_rec*, un programa que actualice la fecha y hora de cada archivo descendiendo recursivamente por los subdirectorios a partir de la posición actual. (Sugerencia:* man touch*).*

9.1.6. Comparando archivos

Frecuentemente posees diferentes versiones de tus archivos, por ejemplo diferentes versiones del mismo programa con el mismo nombre. Posiblemente, haya directorios completos cuyos archivos están duplicados o son versiones diferentes de los contenidos en otro. En estas circunstancias sería bueno poder saber si los archivos de un directorio son, de hecho, diferentes de los que tienen el mismo nombre, contenidos en otro directorio.

En UNIX existe un comando para saber si dos archivos son diferentes: diff. Usaremos este comando para hacer un programa en shell que te indique cuántos y cuáles archivos son diferentes y cuáles no. Para poder contar necesitarás poseer contadores que puedas ir incrementando cada vez que te percates de que un archivo es diferente de su análogo o cada vez que son iguales. Para ello puedes usar el comando let de bash que permite evaluar expresiones aritméticas. Además para que te puedas percatar de la igualdad o diferencia de dos archivos necesitarás saber cuál es el resultado entregado por diff.

diff, como casi todos los comandos ejecutables en el shell, regresa un valor resultado de su ejecución. Es posible *atrapar* este resultado para poder tomar la decisión de incrementar el contador de archivos iguales o el de archivos diferentes. Ya antes mencionamos la variable $? que entrega el resultado del último comando ejecutado, así que ésa será la manera de conocer el valor entregado por diff. diff entrega un cero si los archivos que se le encomendó comparar resultan ser iguales, uno si son diferentes y dos si no se pudo efectuar la comparación por algún motivo. En el listado 9.3 en la siguiente páginaa se muestra el código del programa que deseas.

En las líneas 6–8 se establece el valor inicial de los contadores IGUAL, DIFER y CONTADOR que nos permiten contar archivos iguales, diferentes y el total respectivamente. En la línea 23 se pregunta si el valor entregado por diff es cero, en cuyo caso incrementamos el valor del contador IGUAL en uno (línea 25) usando let. De no ser iguales los archivos (lo que significa que o bien son diferentes o no se pudieron comparar), entonces incrementamos el contador DIFER y siempre incrementamos CONTADOR, el que cuenta todos los archivos.

Actividad 9.17 *¿Qué significan las opciones* qs *de* diff *en la línea 22?*

Código 9.3 Usando el resultado que entrega *diff*

```
1 #!/bin/bash
2 #
3 # Programa para comparar todos los los archivos del
4 # directorio actual con los de otro directorio
5 #
6 CONTADOR=0
7 DIFER=0
8 IGUAL=0
9 if [ $# -ne 1 ]
10 then
11   echo "uso: $0 <directorio>"
12   echo "Se comparan uno a uno los archivos del directorio"
13   echo "actual con los del directorio especificado"
14   else
15   echo " Comparando los archivos en `pwd`"
16   echo " con los de $1 ..."
17   echo ""
18   for i in `ls`
19   do
20     if [ -f $i ]
21     then
22       diff -qs $i $1/$i
23       if [ $? -eq 0 ]
24       then
25         let IGUAL=IGUAL+1
26       else
27         let DIFER=DIFER+1
28       fi
29       let CONTADOR=CONTADOR+1
30     fi
31   done
32   echo ""
33   echo "$CONTADOR archivos comparados"
34   echo "$IGUAL iguales"
35   echo "$DIFER diferentes o incomparables"
36 fi
```

Actividad 9.18 *¿Qué recibe como parámetro de entrada el programa?*

Actividad 9.19 *¿Qué ocurre con el programa cuando hay un archivo en el directorio actual que no existe en el otro directorio?*

Actividad 9.20 *Modifica el programa para que, en caso de que los archivos sean diferentes diga cuál es más reciente (sugerencia:* man test*).*

9.1.7. Otra vez el problema de borrar la basura

Has hecho ya un programa recursivo basado en paseo_rec que elimina archivos innecesarios a partir de un directorio dado. Pero ahora tienes muchos más elementos para poder hacer una nueva versión, mejorada, del programa. Es de notar que el problema esencial del programa es la manera de encontrar los archivos inútiles; esto lo haces con el descenso recursivo por el árbol de directorios, pero hay otra manera de hacerlo sin necesidad de descender por todo el árbol a *pie*. Existe un programa que se encarga de encontrar archivos que cumplan con ciertas características. find nos permite encontrar archivos con características comunes, localizados a partir de cierta ruta específica (es decir, a partir de cierto directorio); también se le dice a find qué hacer cada vez que se encuentre algún archivo que cumpla con las características especificadas, por ejemplo, desplegar el nombre del archivo. Si find te puede regresar un conjunto de archivos tales que su nombre cumpla con la característica de terminar con ∼, por ejemplo, estarás en el camino correcto. Bastaría entonces borrar cada archivo de ese conjunto para eliminar los respaldos de Emacs, por ejemplo.

El siguiente programa automatiza el proceso de borrar la basura y además genera un informe detallado de su actividad, registrándola en un archivo de tu directorio personal. Es de notar que, a diferencia de la versión anterior que borraba basura, este programa no borra la basura a partir del directorio actual, sino que lo hace a partir de tu directorio $HOME. En este programa se hace uso de find de la siguiente forma:

```
% /usr/bin/find . -name "core" -print
```

Los parámetros con los cuáles está siendo ejecutado significan:

`.` El primer parámetro de find es . (punto), busca a partir del directorio actual, previamente el programa se cambia al directorio del usuario.

`-name` Encuentra archivos cuyo nombre coincida con el patrón que se establece a continuación. Es posible también establecer condiciones que no tengan que ver con el nombre del archivo sino con otras características, por ejemplo su fecha de último acceso, lo que permitiría eliminar archivos viejos.

`"core"` Archivos que se llamen core en este ejemplo; en otros casos de este mismo programa se buscan aquellos que cumplan con el patrón *∼, por ejemplo, lo que significa *los que terminan con tilde* (respaldos de Emacs).

`-print` Este último parámetro le indica a find qué hacer con el archivo que encuentre; en este caso -print significa *imprime su nombre*.

Código 9.4 Borrado de archivos usando *find* (1/2)

```
1 #!/bin/bash
2 #
3 # Programa para borrar la basura del directorio del usuario
4 #
5 cd $HOME
6 echo "Limpiando `pwd` ..."
7 FECHA=`date '+%d%m%y'`
8 echo `date`
9 rm -rf Informe.*
10 echo " Salvando estado actual de espacio ocupado..."
11 echo " Archivo: Informe.$FECHA"
12 /usr/bin/du -ks . > "Informe.$FECHA"
13 /usr/bin/du -ks .
14 echo "Borrado cache de Netscape y Trash" >> Informe.$FECHA
15 echo " Limpiando cache de netscape..."
16 rm -rf  $HOME/.netscape/cache
17 mkdir  $HOME/.netscape/cache
18 echo " Limpiando el bote de basura..."
19 rm -rf  $HOME/Desktop/Trash
20 mkdir  $HOME/Desktop/Trash
21 echo "ARCHIVOS BORRADOS">>Informe.$FECHA
22 # cores
23 echo " Borrando cores..."
24 for i in `/usr/bin/find . -name "core" -print`
25 do
26   rm -f $i
27   echo $i >> $HOME/Informe.$FECHA
28 done
29 # respaldos de Emacs
30 echo " Borrando respaldos de Emacs..."
31 for i in `/usr/bin/find . -name "*~" -print`
32 do
33   rm -f $i
34   echo $i >> $HOME/Informe.$FECHA
35 done
36 for i in `/usr/bin/find . -name ".*~" -print`
37 do
38   rm -f $i
39   echo $i >> $HOME/Informe.$FECHA
40 done
```

Código 9.4 Borrado de archivos usando *find* (2/2)

```
41 # archivos auxiliares y de bitácora de TeX y LaTeX
42 echo " Borrando archivos auxiliares de TeX y LaTeX..."
43 for i in `/usr/bin/find . -name "*.log" -print`
44 do
45   rm -f $i
46   echo $i >> $HOME/Informe.$FECHA
47 done
48 for i in `/usr/bin/find . -name "*.aux" -print`
49 do
50   rm -f $i
51   echo $i >> $HOME/Informe.$FECHA
52 done
53 echo " Salvando el nuevo estado actual de espacio ocupado..."
54 echo "Espacio actual:">>Informe.$FECHA
55 /usr/bin/du -ks .  >> "Informe.$FECHA"
56 /usr/bin/du -ks .
57 echo " El directorio `pwd` ha sido limpiado."
58 echo ""
```

Este programa entrega una lista de tu espacio ocupado antes y después de la limpieza, y una lista de los archivos eliminados. El nombre de dicho archivo tiene el prefijo Informe y su nombre se construye en las líneas 9 y 14. También elimina los directorios de cache de Netscape y de depósito de basura Trash del ambiente de trabajo.

Actividad 9.21 *¿Cuál es el sufijo del nombre del archivo de informe que genera el programa? ¿Cómo se obtiene dicho sufijo?*

Actividad 9.22 *Modifica el programa para que te pregunte si deseas borrar cada archivo encontrado (Sugerencia:* man rm*).*

Adicionalmente, este programa (la versión original, no la obtenida en el segundo ejercicio) puede ser ejecutado a una hora determinada en días determinados usando el comando crontab. Para usarlo primero debes hacer lo siguiente, tanto en tu terminal actual como en tu archivo ∼/.bashrc: hay que hacer que el editor de crontab por omisión sea nuestro conocido Emacs. Así que en ∼/.bashrc tienes que añadir las líneas:

```
EDITOR=/usr/bin/emacs
export EDITOR
```

y por lo pronto lo pones en tu shell actual.
Ahora puedes teclear:

```
% crontab -e
```

Eso te pondrá en Emacs listo para editar tu tabla de trabajos a ejecutarse automáticamente por el sistema en tiempos determinados. Teclea lo siguiente con los cambios pertinentes (en vez

de /home/jose debe aparecer tu directorio personal, en vez de borramugres debe aparecer el nombre que le hayas dado al programa para borra basura y en vez de la dirección de correo jose@pateame.fciencias.unam.mx debe aparecer la dirección de tu correo).

```
#-------------------------------------------------------------
#                Borrado de basura
#-------------------------------------------------------------
#
# Se efectúa todos los viernes a las 2:30 AM
#
30 02 * * 6 ( /home/jose/bin/borramugres ) | /bin/mail -s
'borramugres' jose@pateame.fciencias.unam.mx > /dev/null 2>&1
#
```

Nota: A pesar de que se muestran dos líneas sin # en realidad es sólo una; si aparecen dos es sólo para mejorar la presentación.
Esta línea de crontab hace lo siguiente:

1. Ejecuta el comando entre paréntesis, es decir nuestro programa que borra basura.
2. La salida del programa, lo que éste escriba en la pantalla, se pasa a mail con tema *borramugres* y dirección de envío jose@pateame.fciencias.unam.mx. El resto de la línea es para redireccionar la salida estándar de errores a la salida estándar y a su vez ésta es enviada a /dev/null (lo que equivale a tirarla a la basura).

Conviene ahora leer el manual de crontab para entender los primeros cinco campos de la línea, que esencialmente indican cuándo deberá ejecutarse el resto de la línea; en este caso todos los viernes (día 6) a los 30 minutos después de las 02 horas cualquier día (*) de cualquier mes (*).

Sólo por no dejar

Hay una opción más para hacer el programa de eliminación de basura. Como ya se dijo antes, el último parámetro de entrada de find es lo que tiene que hacer cada vez que se encuentra un archivo que cumple con las condiciones buscadas. Es posible decirle a find que, de hecho, ejecute otro comando y pasarle a dicho comando el nombre del archivo encontrado por find; eso hace que puedas quitar los ciclos for que aparecen en el programa y que procesan la salida de find. Esto lo harías reemplazando todo el ciclo por una instrucción como la siguiente:

```
% /usr/bin/find . -name "core" -exec rm -f {} ';'
```

El significado de esta línea hasta antes de -exec es el que ya se ha explicado. -exec le indica a find que ejecute un comando, en este caso rm. En el momento en que find ha encontrado un archivo que cumple con la condición, guarda el nombre del archivo encontrado para que pueda ser pasado como parámetro del comando que se ejecutará. Para hacer referencia a ese archivo encontrado se utiliza {}. Por último, para avisarle al programa ejecutado por find (rm en este caso) que se han terminado los parámetros que le son pasados para su ejecución, se pone un ';' entre comilla simple (para que el shell no crea que se trata de un terminador de instrucción). En síntesis, el significado de la línea mostrada es: find busca a partir del directorio actual (.) archivos cuyo nombre (-name) cumpla con la condición de ser core; una vez que se encuentra uno de ésos ordena la ejecución

(-exec) de rm diciendo: rm -f {} ';' borra (rm), sin preguntar (-f) el archivo que encontré ({}) y nada más. Nota que todo lo que está después del -exec es lo que find le dice a rm.

9.1.8. Para saber más

bash es producto de la *Free Software Foundation* y puedes encontrar información adicional acerca de la programación en este *shell* en: http://www.gnu.org.
También hay un tutorial de programación en bash accesible como documento HOWTO en el *Linux Documentation Project*: http://www.linuxdoc.org.

9.2. Emacs

9.2.1. La madre de todos los lenguajes

A pesar de que Emacs es el editor de texto con mayor funcionalidad que ha existido en la historia de la computación, es increíble – incluso un poco loco – que sus usuarios inviertan tanto tiempo en extender Emacs. Sin embargo, lo hacemos. En esta sección te mostramos por qué, para qué y cómo se hace.

Para nosotros, programadores o aprendices de, el editor de texto es una herramienta, útil para crear más software. Haciendo un símil, podemos pensar en el editor de texto como un martillo para un carpintero o una llave española para un mecánico, como algo útil para resolver la tarea en puerta. Entonces, ¿por qué Emacs es diferente?

La respuesta es que el carpintero y el mecánico *modificarían* sus herramientas para hacerlas mejores, si tan sólo supieran cómo. ¿Quién sabe mejor que ellos mismos lo que necesitan? Pero ellos no son herreros y no pueden modificarlas. Por otro lado, Emacs es un tipo de herramienta especial: es software, lo que significa que la herramienta está hecha de la misma cosa que los usuarios de Emacs hacen con Emacs. Por lo tanto, tú, nosotros y todos los usuarios/programadores de Emacs en el mundo estamos en la feliz posición de ser los *herreros* (diseñadores sería un mejor término) de nuestra herramienta principal.

Damos por hecho que conoces algún lenguaje de programación de alto nivel, aunque no es necesario que manejes un lenguaje de la familia de Lisp. Desarrollaremos varios ejercicios de programación con ejemplos reales y que te serán de utilidad. Es de esperarse también que estés familiarizado con la terminología de Emacs; en caso contrario puedes leer la sección llamada *Emacs* en todos los capítulos anteriores de este libro.

Emacs-Lisp es un lenguaje de programación muy grande y no tenemos el tiempo suficiente para mostrar todas sus construcciones, pero intentaremos mostrarte suficiente como para que puedas manejar cómodamente variables (creación, asignación, búsqueda, etc.), keymaps, comandos interactivos, buffers y ventanas.

Por conveniencia, a lo largo de estas prácticas nos referiremos a Emacs Lisp, como *elisp*. También hablaremos del lenguaje de programación LISP (*LISt Processing*) y en general no haremos

distinción entre este último y elisp: para propósitos de este material, hablaremos indistintamente de LISP o Emacs-Lisp como elisp o simplemente LISP.

9.2.2. LISP en 15 minutos

LISP maneja listas (y listas de listas) y éstas se presentan entre paréntesis. Muchas veces encontrarás las listas precedidas por un apóstrofo (’). Las listas son la parte medular de LISP.

Listas en elisp

En elisp una lista luce como ’(rosa violeta clavel azucena). Esta lista está precedida por un apóstrofo. Los elementos de la lista son nombres de distintas flores, separados entre sí por un espacio en blanco, aunque cabe aclarar que el número de blancos (espacios en blanco, tabuladores o nuevas líneas) no es relevante.

Las listas también pueden tener números dentro, por ejemplo (+ 2 2), que tiene al símbolo ’+’ y al número 2, dos veces.

En elisp, tanto los datos como los programas se representan de la misma forma, esto es, son listas de símbolos, números o bien otras listas, separadas por blancos y rodeadas por paréntesis (dado que los programas lucen como datos, pueden pasarse a otros programas; esto significa que los programas en elisp también son elementos de primera clase y esto constituye una gran herramienta de elisp).

Átomos en elisp

En elisp, a los elementos o palabras que hemos usado en nuestras listas arriba, los llamamos *átomos*. Los átomos, desde el punto de vista de elisp, son unidades indivisibles, al igual que los números y carácteres como ‘+’. Por el contrario, las listas, como veremos más adelante, sí pueden ser divididas (sección 9.6).

Ya dijimos que una lista puede tener átomos, números y otras listas, pero hay un caso particular y es la lista que no tiene nada, y que gráficamente luce así: ( ); se llama la lista vacía, también representada por nil. Esta lista vacía tiene la facultad de ser una lista y un símbolo al mismo tiempo.

La representación escrita de listas y átomos se conoce como expresión simbólicas (*symbolic expression*), o bien *s-expression*. Aclaramos que la representación escrita no tiene nada que ver con la forma interna de las expresiones.

En elisp siempre trabajamos con átomos y listas, pero usualmente a los programadores les gusta hablar en términos más precisos de los átomos que usan. Así, tenemos varios tipos de átomos: números, por ejemplo 13, 295 o 1763; y símbolos tales como ‘+’, ‘foo’, ‘forward-line’, etc.; todos los nombres que hemos usado en las listas de ejemplo, aquí arriba, también son símbolos. Adicionalmente, cualquier texto entrecomillado, incluyendo oraciones o párrafos completos, es un símbolo. Por ejemplo:

```
’(esta lista incluye "texto entre comillas dobles.")
```

es una lista que tiene cuatro símbolos. En elisp todo el texto entre comillas, incluyendo el signo de puntuación y los blancos, es un átomo. Este tipo particular de átomo se conoce como *cadena* y es el tipo de átomos que se utiliza para comunicar mensajes que un humano puede leer. Las cadenas son un tipo distinto de átomo a los números y símbolos y no sólo por su forma, sino también por su uso.

Dado que los blancos en una lista no importan, las siguientes son representaciones de la misma lista:

```
'(esta lista
    luce asi)

'(esta lista luce asi)

'(esta
  lista
  luce
  asi)
```

Por extraño que parezca, ¡hemos cubierto con estos breves ejemplos casi todo lo que podemos ver en elisp! Todo lo demás son combinaciones de lo anterior, aunque posiblemente usando listas más largas y anidadas, pero ningún concepto nuevo. Hay muchas construcciones esotéricas en elisp que utilizan otro tipo de átomos, pero no son comunes y podemos avanzar mucho sin conocerlas.

En resumen, no debes olvidar que: una lista va entre paréntesis, una cadena entre comillas dobles, un símbolo luce como una palabra (o secuencia de ellas, unidas por algún carácter no blanco) y un número luce como un número.

Emacs te ayuda a dar formato a código en elisp

Si tecleas una expresión de elisp en Emacs usando ya sea el modo de *elisp Interaction*, o *Emacs LISP Mode*, entonces Emacs te proveerá con suficientes comandos para que la expresión sea fácilmente legible. Por ejemplo, presionando TAB automáticamente indentará la expresión en el renglón actual usando el número adecuado de espacios.

Además, cuando tecleas un paréntesis que cierra, Emacs momentáneamente saltará al paréntesis correspondiente que abre, de forma tal que tú puedes siempre ver cuál es y detectar errores rápidamente.

Nota: El modo mayor *elisp Interaction* es el modo en que por omisión se encuentra el buffer **scratch**, que casualmente es el único buffer que existe cuando inicias Emacs.

Ejecución de programas

Una lista en elisp es un programa listo para ser ejecutado. Si lo ejecutas (*evalúas* en el caló de elisp) hará una de tres posibles cosas: nada, excepto regresar la lista misma; mandarte un mensaje de error o tratar el primer símbolo en la lista como un comando para hacer algo.

El apóstrofo sencillo que aparece en algunas de las listas de ejemplo en la sección anterior se llama quote (es una función). Cuando precede a una lista, le dice a elisp que no haga nada con ella excepto tomarla tal y como está escrita. En cambio, si no aparece un apóstrofo delante de la lista,

entonces elisp interpreta el primer elemento en la lista como un comando que tiene que obedecer. En elisp estos comandos se llaman funciones. La lista (+ 2 2) arriba no tenía un apóstrofo, por lo que elisp entiende el + como una función que hace algo con el resto de los elementos en la lista, en este caso, sumar los elementos en la lista.

Actividad 9.23 *Teclea* (+ 2 2) *en el buffer* ****scratch****, *pon el cursor justo después del paréntesis que cierra y luego teclea* C-x C-e.

Lo que acabas de hacer se conoce como *evaluar la lista*.

Actividad 9.24 *Ahora haz lo mismo, pero con una lista que esté precedida con un apóstrofo, por ejemplo,* '(que bonito). *¿Qué pasa?*

Cuando una lista está precedida por un apóstrofo, entonces el intérprete de elisp **no** interpreta la lista, la lee de manera textual. Nos referiremos a este tipo de listas como listas textuales, citadas, inmutables o constantes.

Mensajes de error

A continuación te pediremos que evalúes una expresión que hará que el intérprete de elisp marque un error. Lo hacemos por varias razones: la primera y más importante, para mostrarte que no pasa nada malo, es una actividad inofensiva; haremos esto varias veces a lo largo de la sección. La segunda razón es para que te acostumbres al lenguaje y la forma de los mensajes de error, ya que una vez que los entiendas te darás cuenta que son muy informativos. Puedes pensar en estos mensajes de error como señalización para el viajero de las carreteras de elisp: son crípticos al principio (como casi cualquier otro tipo de señalización), pero una vez que los entiendes, te llevan de la mano a tu destino final.

Lo que haremos, y ese es tu siguiente ejercicio, será evaluar una lista que no está citada y que no tiene un nombre de función como su primer elemento. Hazlo:

Actividad 9.25 *Evalúa la siguiente lista:* (esta lista marca un error).

Después de hacer eso, debiste recibir el mensaje: Symbol's function definition is void: esta, ¿cierto? Dado que este tipo de mensajes es muy frecuente, a continuación te presentamos una explicación, aunque ya debes tener una buena idea de lo que el mensaje significa.

Sabemos que el problema fue ocasionado por el símbolo esta; también se menciona la palabra function, una palabra muy importante. Para efectos prácticos, valga la siguiente definición de función: *un conjunto de instrucciones para la computadora que le indican lo que debe hacer*. En realidad, el símbolo le dice a la computadora dónde encontrar las instrucciones a ejecutar.

Ahora sí, estamos en posición de entender en su totalidad el mensaje de error: el símbolo esta no contiene la definición de conjunto alguno de instrucciones para la computadora.

La extraña frase Symbol's function definition is void tiene que ver con la forma en que Emacs está implementado, ya que cuando un símbolo no tiene un conjunto de instrucciones asociado, el lugar donde tal conjunto (de existir) debería estar, es ocupado por la función void, que no hace nada.

Por otro lado, dado que sí pudimos sumar 2 y 2 evaluando (+ 2 2), podemos inferir que el símbolo + debe tener un conjunto de instrucciones que la computadora obedece; esta instrucción suman los dos números.

Finalmente, debes saber que los símbolos también se usan para nombrar variables o funciones. Esto no quiere decir que el símbolo, por ejemplo +, sea la función, sino simplemente una referencia *al lugar* donde están las instrucciones que hacen a la función. Su papel es similar al de los nombres que tenemos las personas, nuestro amigo Juan Pérez, puede ser referido como Juan, pero **no es** las letras J, u, a, n. De la misma manera, las instrucciones que le dicen a la computadora que sume una lista de números (y cómo hacerlo) no son +.

Más aún, elisp nos permite poner varios nombres a un mismo conjunto de instrucciones. Para sumar dos números podríamos hacer que un nuevo nombre, digamos suma, haga referencia a las mismas instrucciones que +.

El intérprete de elisp

Ya debes tener una buena idea de qué hace el intérprete de elisp cuando encuentra una lista: si es inmutable, regresa la lista; si el primer elemento es un símbolo, busca su conjunto de instrucciones asociado y los ejecuta usando el resto de los elementos de la lista como parámetros; si el conjunto de instrucciones asociado es void, marca un error. Muy sencillo, ¿no? Sin embargo, hay varias complicaciones que necesitas entender para seguir aprendiendo elisp y comenzar a programar:

- Además de listas, elisp puede interpretar también símbolos (sin paréntesis alrededor y sin quote), en cuyo caso el intérprete intentará determinar el valor del símbolo como variable. Regresaremos a esto más tarde.

- Algunas funciones no trabajan igual que la evaluación de listas. Se llaman *formas especiales* y se usan para cosas muy particulares, como definir funciones. No hay muchas y las iremos revisando conforme avancemos.

- Si lo que el intérprete está tratando de evaluar no es una forma especial, y sí es parte de una lista, el intérprete revisa si la lista tiene otra lista adentro. Si hay una lista adentro, primero ve qué debe hacer con ella y si a su vez tiene otra lista adentro, primero hace ésa y así sucesivamente. Una vez que concluye su trabajo (de evaluación) con las listas internas, hace lo que tiene que hacer con la lista actual.

En cualquier otro caso, el intérprete va evaluando de izquierda a derecha y de expresión en expresión.

La última de las complicaciones listadas arriba es la más interesante de todas y la más abstracta también. ¿Por qué evalúa de adentro hacia afuera? La respuesta tiene dos partes: primero debes saber que toda evaluación en elisp regresa un valor (o bien termina en error); lo segundo que te interesa saber es que el valor lo regresa al punto de llamada. Entonces, las expresiones más externas esperan, por decirlo de alguna forma, a los valores que regresan las expresiones anidadas para usarlos. Por ejemplo, veamos que sucede cuando evaluamos (+ 2 (+ 3 4)); hazlo: evalúa esa expresión.

Verás 9 aparecer en el área de eco. Y eso sólo puede suceder si primero se evalúa (+ 3 4) a 7, y ese valor es visto y usado por la otra aplicación de +, como si en realidad estuvieras evaluando: (+ 2 7).

Ahora estás en posición de saber que C-x C-e evalúa el comando eval-last-sexp[4]

Actividad 9.26 *Evalúa la expresión* (+ 2 (+ 3 4))*, en distintas partes de la misma. Por ejemplo, pon el cursor al final y evalúa (debes obtener 9 otra vez); coloca el cursor sobre el último paréntesis – o detrás del paréntesis que cierra* (+ 3 4) *– y atrás de un número, etc. ¿Qué pasa? ¿Puedes explicar por qué pasa eso?*

Variables

En elisp un símbolo puede tener un valor asociado a él, al igual que una función, como ya vimos. Nota que son cosas distintas: una función es un conjunto de instrucciones, mientras que un valor es algo, como un número o un nombre que puede cambiar; de este hecho se desprende el nombre variable, que es el nombre con el que conocemos a los símbolos que representan un valor.

Algo muy importante es que un símbolo puede tener tanto un valor como una función asociados. Puedes pensar en un símbolo como una cajonera, en una cajón se guarda el valor y en otro las instrucciones que conforman la función. Entonces puedes cambiar el contenido de uno de los cajones sin perturbar al otro.

Por ejemplo, fill-column es una variable que existe en todos los buffers de Emacs. Para saber su valor puedes evaluarla y verás que su valor es usualmente 70 o 72.

En el caló de elisp, decimos que una variable es o está ligada a un valor: un número, como 70, una cadena "como esta" o a una lista (como esta otra). Incluso podemos enlazar una variable a la definición de una función.

Otra forma muy común de evaluación es M-: (eval-expression). Esta función viene desactivada por omisión, así que la primera vez que la ejecutes, en lugar de un prompt del estilo Eval: en el área de eco, Emacs te hará preguntas que te permiten decidir qué hacer: activar el comando sólo por esta ocasión, pero dejarlo desactivado; activarlo (permanentemente); o no hacer nada. Se considera un comando peligroso para novatos en Emacs, pero tú ya no eres un novato.

Recapitulando, ahora sabes que una opción para evaluar, que hemos usado mucho, es poner la expresión en un buffer de Emacs, poner el cursor al final de la expresión y luego ejecutar C-x C-e. Ahora tienes otra opción: M-: y después del prompt Eval: escribir la expresión y presionar Enter.

Actividad 9.27 *Prueba evaluar algunos de los ejemplos que hemos revisado hasta el momento usando* M-:*. Nota los aspectos buenos y malos de ambos enfoques. Tener las dos formas es muy útil y dependiendo del contexto puedes escoger la que más te convenga.*

¿Notaste que en la evaluación de fill-column *no usaste paréntesis? Prueba evaluar* (fill −column)*, ¿por qué marca un error?*

[4] El *sexp* abrevia *symbolic expression* y se lee "sex pee". En español es poco usual llamarle "evaluación de expresiones simbólicas" a la acción de eval-last-sexp y nos referimos a dicha acción simplemente como evaluación.

Ahora pon el cursor en el espacio en blanco después del símbolo +, *en la expresión* (+ 2 2), *y evalúa. ¿Qué error marca? ¿Es igual a los errores anteriores? Si no son iguales, ¿cuál es la diferencia entre ellos? Explica.*

Argumentos y tipos de argumentos

Para redondear la discusión sobre evaluación en elisp, también tenemos que hablar de la forma en que el intérprete reconoce y pasa los operandos a las funciones. Verbigracia, en nuestro ejemplo favorito, (+ 2 2), el intérprete regresa 4 y decimos que los dos números (ambos 2) que siguen al + son los *argumentos* de la función. Ellos constituyen la información que el intérprete pasa a la función.

En elisp, los argumentos son los símbolos o listas que siguen al nombre de la función. Distintas funciones requieren distintos números de argumentos: algunas no reciben ningún argumento y otras reciben un número variable de argumentos.

¿De qué tipo debe ser el argumento que pasamos a una función? Depende del tipo de información que requiera la función para llevar a cabo sus acciones. +, por ejemplo, sólo recibe números (¡o carácteres!), mientras que concat, por mencionar otra función, recibe cadenas. Por ejemplo, si evaluamos

```
(concat "abc" "def")
```

obtendremos "abcdef", es decir concat concatena todos sus argumentos (cadenas) en una sola cadena.

Una función como substring recibe tanto una cadena como números. Lo que hace es regresar una subcadena del primer argumento (cadena). Por ejemplo,

```
(substring "que bonito es Emacs elisp" 14 19)
```

regresa "Emacs". El primer argumento, como puedes notar, es una cadena; el segundo es la posición (dentro de esa cadena, empezando a contar desde 0) desde donde será tomada la subcadena resultado; el segundo número (tercer argumento) es la posición a la derecha donde debe terminar la subcadena.

Los valores que regresan las funciones en elisp también pueden ser usados como argumentos.

Actividad 9.28 *Evalúa las siguientes expresiones y explica cada uno de los resultados. Si no entiendes algún resultado, evalúa las subexpresiones hasta que encuentres cuál es el truco.*

```
(+ 2 fill-column)

(concat "wow, veo  " (+ 2 fill-column) " lindos gatitos.")

(concat "wow, veo  " (number-to-string (+ 2 fill-column))
                     " lindos gatitos.")
(+)
(*)
```

```
(+ 1 2 3 4 5)
(+ 1 (* 1 2) 4 (+ 2 fill-column) (* 1 2 -10))
(+ 2 'hola)
```

¿Qué hace la función number-to-string*? ¿Qué tipo de argumentos recibe y cuántos?*

La función *message*

Al igual que la función +, message recibe un número variable de argumentos. Se usa para enviar mensajes al usuario y es tan útil que incluso tiene su propia sección.

Con esta función se consigue que un mensaje se despliegue en el área de eco. Por ejemplo, puedes desplegar un mensaje evaluando lo siguiente:

```
(message "Esto es un bonito mensaje de prueba")
```

Nota que el mensaje aparece en el área de eco incluso con las comillas. Eso sucede porque ése es el valor regresado por la función message. Usualmente, cuando tú usas message en tus programas, esto no sucederá y el texto que verás se imprimirá como un efecto secundario, sin las comillas.

Sin embargo, si hay un %s en el argumento entrecomillado de message, no imprimirá un %s en el área de eco, sino que buscará un argumento después del argumento entrecomillado, lo evaluará y pondrá su valor en la posición del %s en la cadena. Por ejemplo,

```
(message "El nombre de este buffer es: %s" (buffer-name))
```

imprimirá algo como: "El nombre de este buffer es: *scratch*" al ser evaluada, suponiendo que sí estés en un buffer llamado *scratch* – cuando ejecutamos esto, sí estábamos en uno llamado así –.

Si lo que quieres es imprimir un número en notación decimal, entonces cambia el %s por un %d, como por ejemplo:

```
(message "El valor de fill-column es: %d" fill-column)
```

Si hay más de un %s en la cadena entre comillas, entonces el primer argumento que siga a la cadena será puesto en el lugar del primer %s, el segundo argumento sustituirá al segundo %s, y así sucesivamente. Por ejemplo:

```
(message "Hay %d %s en el laboratorio"
         (- fill-column 15) "elefantes rosas")
```

imprimirá "Hay 55 elefantes rosas en el laboratorio".

Lo valioso de message es que si lo usas con formato (%), le puedes pedir que evalúe funciones al momento de evaluarse ella misma.

Asignar valores a una variable

Hay varias formas por medio de las cuales podemos darle un valor a una variable. Aquí veremos setq. Más adelante hablaremos de let. En elisp decimos que estamos *ligando* una variable a un valor.

En esta sección veremos cómo funciona setq[5] y cómo se pasan los argumentos.

setq

Para asignarle a la variable flores la lista '(rosa violeta clavel azucena) puedes evaluar la siguiente expresión:

```
(setq flores '(rosa violeta clavel azucena))
```

Verás la lista (rosa violeta clavel azucena) aparecer en el área de eco. Esta lista es el valor de regreso de la función y, como efecto secundario, la variable (el símbolo) flores está ahora ligado a la lista. Es decir, que después de evaluar la expresión de arriba puedes evaluar el símbolo y verás la lista aparecer en el área de eco.

Con setq también podemos asignar valores a varias variables al mismo tiempo. Por ejemplo, para asignar una lista de flores a la variable flores y una lista de animales a la variable animales, podemos hacer:

```
(setq flores '(rosa violeta clavel azucena)
      animales '(león sapo oso perro))
```

Nota que la expresión bien podría ir toda en una línea, pero tal vez no hubiese cabido en la página y además los humanos podemos leer con mayor facilidad listas con un formato más agradable.

A pesar de que hemos usado hasta el momento el término ligar, hay otra forma de referirnos al trabajo que realiza setq y es decir que setq hace que el símbolo *apunte* a la lista. Esto también se llama *referencia* y es muy común, así que a partir de este momento nos referiremos a este tipo de asignaciones así.

Contador

En esta sección construiremos un contador (muy sencillo) que usa setq. Dicho contador puede ser usado para contar cuántas veces se repite una cierta parte de tu programa. Primero necesitamos inicializar una variable a cero; luego añadir uno a esa variable cada vez que se repita esa parte en el programa. Para hacer esto, necesitas una variable que sirva de contador y dos expresiones: un setq que asigne cero al contador; una segunda expresión setq que incremente el contador cada vez que es evaluado.

```
(setq contador 0)                ; Inicialización

(setq contador (+ contador 1))   ; Incremento

contador
```

El texto que sigue a un carácter ; y hasta el fin de línea, es un comentario que es ignorado por el intérprete de elisp.

[5] setq en realidad es una abreviatura de la combinación de las funciones set y quote.

Si evalúas la primera de estas expresiones, la inicialización, y luego evalúas la tercer expresión, el contador, el número 0 aparecerá en el área de eco. Si evalúas entonces la segunda expresión, el incremento, el contador aumentará en 1 su valor; así, si evalúas nuevamente la tercera expresión, entonces verás el número 1 aparecer en el área de eco. Cada vez que evalúes la segunda expresión, el valor de contador aumentará en 1.

Actividad 9.29 *Evalúa varias veces la segunda y tercera expresiones de manera alternada; después explica con detalle cómo evalúa la segunda expresión el intérprete de* elisp.

9.2.3. Resumen

Aprender elisp es como escalar una montaña en la que la primera parte es la más difícil. Tú ya subiste esa parte, así que lo que sigue se irá haciendo más sencillo conforme avancemos.

En resumen:

- Los programas en elisp están formados por expresiones, que son listas o átomos.
- Las listas están formadas por cero o más átomos y listas interiores, separadas por espacios en blanco y rodeadas por paréntesis. Una lista puede ser vacía.
- Los átomos son símbolos (palabras) como forward−paragraph, símbolos de un sólo carácter como +, cadenas de carácteres o números.
- Un número se evalúa a sí mismo.
- Una cadena también se evalúa a sí misma.
- Cuando evalúas un símbolo, regresa su valor.
- Cuando evalúas una lista, el intérprete de elisp checa el primer símbolo en la lista y luego a la función ligada a ese símbolo, luego las instrucciones en la función son ejecutadas.
- Un sólo apóstrofo le indica al intérprete que debe regresar la siguiente expresión tal y como está escrita y no evaluarla.
- Los argumentos son la información pasada a una función. Los argumentos de una función se calculan evaluando el resto de los elementos de la lista, en la cual la función aparece como el primer elemento.
- Una función siempre regresa un valor cuando se evalúa (a menos que genere un error); además, también puede llevar a cabo alguna otra acción, llamada “efecto secundario”. En muchos casos, el propósito principal de una función es crear un efecto secundario.

Actividad 9.30 *Crea un contador que incremente de 2 en 2. Además, escribe una expresión de tal forma que cuando se evalúe despliegue un mensaje que diga “El valor del contador es X”, donde X deberá ser el valor actual del contador.*

9.3. Practicando evaluaciones

Antes de que aprendas a escribir definiciones de funciones en elisp, es importante que practiques evaluando funciones que ya fueron escritas. Estas expresiones serán listas con los nombres de las funciones como su primer (y en muchos casos único) elemento. Comenzaremos con algunas funciones relacionadas con buffers, ya que son sencillas e interesantes.

> Siempre que das un comando de edición a elisp, tal como el comando para mover el cursor o recorrer la pantalla, estás evaluando una expresión, cuyo primer elemento es una función. Así funciona Emacs.

Cuando tecleas carácteres, provocas que el intérprete de elisp evalúe expresiones y así es como obtienes resultados. Incluso teclear simple texto involucra evaluación de funciones de elisp. En este caso, veremos una que usa self-insert-command, que simplemente inserta el carácter que tecleaste. Las funciones que evalúas tecleando alguna combinación de teclas se llaman funciones *interactivas* o *comandos*; cómo crear este tipo de funciones lo veremos más adelante, en la sección 9.4.1.

9.3.1. Nombres de buffers

Las dos funciones, buffer-name y buffer-file-name, muestran la diferencia entre un archivo y un buffer. Cuando evalúas (buffer−name), el nombre del buffer se despliega en el área de eco; cuando evalúas (buffer−file−name), el nombre del archivo visitado en el buffer se despliega. Usualmente el nombre del buffer es el mismo nombre del archivo que visita, mientras que el nombre del archivo viene con su ruta completa al archivo.

Un buffer y un archivo son entidades completamente distintas. Un archivo es información almacenada permanentemente en la computadora (a menos que tú lo borres), mientras que un buffer es información dentro de Emacs que desaparecerá cuando acabes tu sesión (o cuando mates el buffer). La distinción es importante porque nos permite distinguir entre acciones útiles cuando hacemos programas. Sin embargo, muchas veces escucharás a gente decir “estoy editando un archivo” (en Emacs), en lugar de algo más apropiado como “estoy editando un buffer que después salvaré a un archivo”. Usualmente este tipo de ambigüedades son comprensibles por contexto para una persona, pero la computadora no es tan inteligente y cuando programas en Emacs elisp es importante que le digas *exactamente* lo que deseas.

No todos los buffers de Emacs están asociados a un archivo. Por ejemplo, cuando empiezas tu sesión con Emacs, éste arranca con un buffer llamado *scratch*; este buffer no está visitando ningún archivo, lo mismo que el buffer llamado *Help*.

Actividad 9.31 *Ve al buffer* ∗*scratch*∗ *y evalúa las dos expresiones:*

```
(buffer-name)
(buffer-file-name)
```

también evalúa las expresiones usando C-u C-x C-e.

Obteniendo buffers

La función buffer-name regresa el nombre del buffer, pero si lo que tú quieres es el contenido del buffer, entonces debes usar current-buffer. Es decir, con current-buffer tú obtienes el buffer mismo, no sólo su nombre. Sin embargo, hay una pequeña complicación: si tú evalúas (current-buffer), lo que obtienes es el buffer pero su forma impresa se verá como: #<buffer *scratch*>.

Otra función relacionada es other-buffer que te regresa el buffer más recientemente seleccionado que no sea en el cual estás actualmente.

Actividad 9.32 *Prueba las funciones* buffer-name *y* other-buffer *haciendo lo siguiente. Ve a algún otro buffer en tu sesión actual de Emacs y de regreso a* *scratch*; *después evalúas las dos funciones (¿recuerdas* C-x b *?).*

Cambiando de buffer

La función other-buffer es útil cuando se utiliza con switch-to-buffer. De hecho, esta última es una de las funciones que debes usar con mayor frecuencia en Emacs, cuando ejecutas C-x b; esa combinación de teclas está ligada al comando switch-to-buffer, que recibe un buffer como parámetro y lo muestra en la ventana actual.
Por ejemplo, si evalúas lo siguiente:

```
(switch-to-buffer (other-buffer))
```

verás que hace justo lo mismo que C-x b, ya que estás ejecutando (de otra forma) la misma función con los mismos argumentos.

En general, en programas preferimos set-buffer sobre switch-to-buffer; esto es porque los humanos necesitamos ver lo que hacemos, pero los programas no tienen ojos[6] y no necesitan que el buffer en el que operan sea visible en una ventana de Emacs.

Tamaño de un buffer y la localización del punto

Veamos algunas funciones muy simples, buffer-size, point, point-min, point-max. Estas funciones dan información sobre el buffer, su tamaño, la posición del punto y las posiciones inicial y final del buffer.

buffer-size regresa el tamaño del buffer, que no es otra cosa sino la cuenta de todos los carácteres en el buffer. point regresa la cuenta de todos los carácteres desde el inicio hasta el punto donde actualmente se encuentra el cursor (uno antes, para ser exactos; o mejor dicho, el punto está entre el carácter sobre el cual está el cursor y el anterior); point-min es la primera posición permisible del buffer (o la cuenta de carácteres desde el principio del buffer hasta la primera posición donde podemos trabajar en el buffer) y point-max la cuenta de caracteres hasta la última posición permisible del buffer.

[6] Sí tienen cerebros y bocas, por eso uno puede hablar con sus programas, incluso discutir, aplicarles un correctivo o simplemente platicar con ellos.

Actividad 9.33 *Evalúa las cuatro funciones nombradas en esta sección. Después mueve el cursor en el buffer y repite este proceso.*

Parece tonto tener una función que siempre regresa 1, como point-min, ¿cierto? Pues no, no es cierto. Cuando programamos en elisp, tenemos la facultad de "reducir" nuestro espacio de operación a cualquier subconjunto del buffer actual; esto que se llama *narrowing* ni siquiera lo voy a intentar traducir. Desgraciadamente no nos da tiempo de ver en este curso cómo y para qué puedes usarlo, pero debe ser claro que si estamos trabajando en una sección del buffer tiene sentido preguntar cuál es la primera posición de esa sección, y eso es lo que hace point-min. point-max hace algo similar.

No desprecies estas funciones, ya que a pesar de su aparente simplicidad son sumamente útiles para hacer un gran número de funciones interesantes. De hecho, un gran número de funciones de Emacs las utilizan y algunos de los ejemplos que veremos más adelante también.

9.4. Cómo escribir definiciones de funciones

En elisp, todas las funciones están definidas en términos de otras funciones, excepto algunas primitivas que están escritas en el lenguaje de programación C. De cualquier manera, las funciones primitivas y no primitivas que vienen con Emacs y las nuevas funciones que tú escribas, todas se ejecutan igual.

Vamos a enfatizar más esto: las funciones que tú escribas y las que ya están integradas a Emacs, se ejecutan de la misma forma. Entonces te preguntarás, ¿si la diferencia no es importante, para qué me dices? La respuesta es porque es un dato interesante.

9.4.1. La forma especial defun

En elisp, un símbolo tal como mark-whole-buffer tiene cierto código asociado a él que le indica a la computadora qué hacer cuando la función sea llamada (o ejecutada). Dicho código se conoce como la *definición de la función* y se crea evaluando una expresión que comienza con la palabra reservada defun (que abrevia a *define function*). Dado que defun no evalúa sus argumentos en la forma usual, se conoce como una forma especial.

En subsecuentes secciones veremos cómo están hechas (i.e. el código) de algunas funciones de Emacs; en ésta veremos una función muy simple, para explicar cómo funciona defun. Una definición de función tiene *hasta* cinco partes después de la palabra defun:

1. El nombre del símbolo al cual la función será ligada, pegada, asociada[7].
2. Una lista de los argumentos que serán pasados a la función. Si no se pasarán argumentos a la función, entonces usamos la lista vacía, ( ).
3. Documentación que describe a la función (técnicamente este campo es opcional, pero en la práctica muy recomendable).

[7] Del verbo en inglés *attach* y que significa algo así como ligar, pegar o adjuntar.

4. Opcionalmente, una expresión para hacer que la función sea interactiva, para que puedas usarla tecleando M-x y el nombre de la función; o tecleando una combinación de teclas apropiada.
5. El código que le dice a la computadora qué hacer: el cuerpo de la función.

Es más fácil pensar en una función como un esqueleto de la forma:

```
(defun nombre (argumentos...)
       "Documentación opcional..."
       (interactive paso-de-información) ; opcional
        cuerpo...)
```

Por ejemplo, a continuación te presentamos una función que multiplica su argumento (un número) por 7:

```
(defun multiplica-por-7 (num)
  "Multiplica NUM por 7."
  (* 7 num))
```

Este ejemplo no es interactivo, eso lo veremos más adelante; por el momento nota que las otras partes están todas ahí. Su nombre, multiplica-por-7; sus argumentos, (num); su documentación, "Multiplica NUM por 7."; y su cuerpo, (* 7 num).

En este caso, la función sólo recibe un argumento, que aquí decidimos llamar num, pero bien pudo ser cualquier otro nombre, por ejemplo, parangaricutirimicuaro , pero ese hubiera sido un muy mal nombre, ya que no le dice nada a un humano. En cambio, num, suena como "número" y eso es más o menos lo que esperamos sea pasado a la función, un número. Por ejemplo, evalúa la definición de la función aquí arriba y luego evalúa las siguientes:

```
(multiplica-por-7 6)
(multiplica-por-7 10)
```

deberás ver 42 y 70, respectivamente.

Los nombres que uses para tus argumentos en la definición de una función se convierten automáticamente en privados, esto es, sólo son vistos (y por ende, sólo pueden ser usados) dentro del cuerpo de la función. En nuestro caso, podrás notar que num se multiplica por 7, pero si tú tratas de usar num fuera de la función verás que no está ligado a nada; pon num en el buffer `*scratch*` y luego ejecuta C-u C-x C-e para convencerte.

Algo que debes notar es que los nombres que uses para tus argumentos pueden ser usados en el exterior de la función (i.e. fuera de ella) o como nombres de argumentos en otra función o en muchas; en realidad no importa, porque no afecta el curso del cómputo de tu función.

La documentación de tu función es lo que aparece cuando usas C-h f y un nombre de función. Por ejemplo, teclea C-h f y luego pon multiplica-por-7 y verás un nuevo buffer de ayuda con la documentación que usamos para describir nuestra función.

Actividad 9.34 *Haz una función que saque el promedio de dos números. Llama a sus parámetros* num *y* num-2*. Vuelve a probar* multiplica-por-7 *y comprueba que sigue funcionando, a pesar de que ya hay al menos una función más (la que acabas de hacer que calcula el promedio) que utiliza el mismo nombre para un argumento,* num*.*

Instalar nuevas definiciones

Aquí arriba, cuando te pedimos que evaluaras la definición de la función multiplica−por−7, te hicimos trampa porque no te explicamos lo que iba a suceder. Para efectos didácticos era mejor así, pero aquí te va la explicación completa.

Para que tú puedas ejecutar una función con ciertos argumentos, primero Emacs debe conocer dicha función. Para lograr esto, debes instalar la definición en Emacs. Cuando pones el cursor al final de la definición de una función, como multiplica−por−7, y le picas C-x C-e, verás el nombre de la función en el área de eco (eso es lo que regresa una definición de función, el nombre de la función). En ese momento, la función ha sido instalada en Emacs y es tan parte de Emacs como forward-word o cualquier otra función de edición que tú uses (multiplica-por-7 permanecerá instalada en Emacs hasta que termines tu sesión de Emacs; para instalar código permanentemente puedes introducirlo en tu ∼/.emacs).

Cómo cambiar la definición de una función

Si quieres cambiar el código en multiplica-por-7, sólo tienes que reescribirlo y volver a instalar la función en Emacs. Así modificas cualquier código de función en Emacs, es muy sencillo.

A manera de ejemplo, puedes cambiar el código de la función multiplica-por-7 por una versión que no utilice multiplicación ('*'), por ejemplo:

```
(defun multiplica-por-7 (num)
  "Multiplica NUM por 7."
  (+ num num num num num num num))
```

¡Vamos, pruébalo! Evalúa esta nueva versión y vuelve a probar la función. De hecho, ésta es la forma de hacer funciones pequeñas en Emacs elisp: haces una primera versión, la instalas, la pruebas y luego haces correcciones y vuelves a instalar, probar, en un ciclo hasta que quede como la deseas.

Cómo hacer una función interactiva

Haces una función interactiva poniendo una lista que empieza con la forma especial interactive justo después de la documentación. Un usuario puede invocar una función interactiva tecleando M-x función, o tecleando la combinación de teclas a las cuáles está ligada la función, como por ejemplo, C-n para ejecutar next-line o C-x h para ejecutar mark-whole-buffer.

Algo muy interesante, que escapa al programador de Emacs elisp novato, es que cuando llamas a una función interactiva, su resultado no aparece en la línea de eco. La razón es que muchas de estas funciones son hechas por sus efectos secundarios, como avanzar una línea o borrar una palabra, y no por el valor que regresan. Si el valor regresado apareciera en el área de eco distraería mucho al usuario.

Una versión de nuestra multi-citada función multiplica-por-7 puede ser interactiva y aun así imprimir su resultado en el área de eco. Por ejemplo:

```
(defun multiplica-por-7 (num)
```

```
"Multiplica NUM por 7."
(interactive "p")
(message "El resultado es: %d" (* 7 num)))
```

Ahora, cuando evalúes esta nueva versión de multiplica-por-7 podrás usar la función de manera interactiva: teclea C-u 6 M-x multiplica-por-7 Enter y verás el mensaje "El resultado es: 42" aparecer en el área de eco.

En general, puedes invocar una función como ésta en una de dos formas:

1. Tecleando un argumento prefijo que será num para la función, seguido de M-x y el nombre de la función. Por ejemplo, C-u 3 M-x forward−sentence, o
2. tecleando cualquier tecla o combinación de ellas a las cuales esté ligada la definición, como en C-u 3 M-e.

Ambos ejemplos mencionados hacen lo mismo: mueven el punto tres oraciones hacia adelante (dado que multiplica-por-7 no está ligado a ninguna tecla, entonces no podemos usar la segunda forma, aunque debiste haber aprendido a ligar funciones a teclas hace un par de capítulos dentro del marco de este material).

Un argumento prefijo N puede ser pasado a una función usando C-u N o bien M–N. Si sólo tecleas C-u sin argumentos, el valor por omisión es 4; compruébalo, ejecutando

C-u M-x multiplica −por−7

En nuestro ejemplo, la expresión (interactive "p") le dice a Emacs que pase el argumento prefijo a la función y que use su valor (el del argumento prefijo) como argumento de la función.

Diferentes opciones para *interactive*

En nuestro ejemplo, "p" le indica a Emacs que pase el argumento prefijo (un número) como parámetro a la función. Pero ¿qué sucede si la función no recibe un número? La forma especial interactive tiene un gran número de opciones, cerca de 20 carácteres, y muy probablemente uno de ellos te permitirá pasar la información que requieres a tu función. No los revisaremos todos aquí, sólo veremos unos cuantos, pero puedes buscar la documentación de interactive en el manual en línea de Emacs elisp.

Por ejemplo, la letra "r" le dice a Emacs que pase a la función los valores inicial y final de la región (el valor del punto y el valor de la marca en el buffer actual) como dos argumentos separados a la función. Se usa como sigue:

```
(interactive "r")
```

La letra "B" le dirá a Emacs que pida un nombre de buffer que será pasado a la función. No sólo eso, sino que puedes usar una etiqueta para preguntar por el buffer y, además, Emacs te ayudará a completar el nombre. Se usa así:

```
(interactive "BDame un buffer: ")
```

Una función con varios argumentos también puede usar interactive; la forma en que separamos los argumentos es usando un símbolo de nueva línea, \n. Por ejemplo,

```
(interactive "BDame un buffer: \nr")
```

le dice a Emacs que pida un buffer, que será el primer argumento de la función, y que además pase el valor de la marca y el punto (la región) como segundo y tercer valor (no necesariamente en ese orden) a la función.

Si la función no recibe ningún argumento, eso no es motivo para que no puedas usar interactive; de hecho, interactive siempre recibe el mismo número de argumentos que la función, así que a una función sin argumentos le corresponde una forma interactive sin argumentos, (interactive).

9.4.2. *let*

Una expresión let también es una forma especial que usarás en muchas definiciones de funciones. Su uso es útil para ligar un símbolo a un valor, de forma tal que le permite al intérprete distinguir esta variable de otra con el mismo nombre y que no forme parte del cuerpo de la función.

Para entender por qué let es necesaria, piensa en tu hogar, al cual tú te refieres como "mi casa"; ahora imagina que estás hablando con Foo, y éste está hablando de "mi casa". Muy probablemente Foo está hablando de su casa y no de tu casa, pero si cada vez que tú oyes "mi casa" piensas en tu casa, te confundirás sin lugar a dudas. Ahora imagina que dos funciones usan una variable y que dicha variable se llama igual, pero debe tener distintos valores para cada función.

let evita este tipo de confusiones. Lo que hace es crear un nombre que "opaca", o mejor dicho "oculta", a cualquier referencia externa a ese mismo nombre en el cuerpo del let.

Una expresión let tiene tres partes: (let varlist cuerpo...). La primera parte es el símbolo let, el segundo es una lista, identificada como *varlist*; cada elemento de la lista es un símbolo o una lista de dos elementos, el primero de los cuales es un símbolo y el segundo una expresión; el tercer elemento de la lista es el cuerpo.

Los símbolos en la lista varlist, son las variables a las que se les asigna un valor inicial por let. Los símbolos que aparecen solos son asignados el valor nil; el intérprete evalúa los segundos elementos de cada lista y asigna al primer elemento (un símbolo) el resultado.

Por lo tanto, una varlist puede lucir así: (num (sex f)), en cuyo caso Emacs asignará a la primera variable, num, el valor nil y la variable sex valdrá f.

Si la lista de variables, varlist, está compuesta de parejas (el caso más usual), let luce así:

```
(let ((var value)
      (var value)
      ...
      (var value))
  cuerpo ... )
```

Ejemplo de *let*

La siguiente expresión let le da valor a dos variables y luego imprime sus valores. El cuerpo consiste sólo de la llamada a message:

```
(let ((zebra 'rayas)
      (tiger 'feroz))
```

```
(message "Un animal tiene %s y el otro es %s" zebra tiger))
```

Puedes evaluar esta expresión. El resultado lucirá como: “Un animal tiene rayas y el otro es feroz”.

Actividad 9.35 *Haz una función llamada* suma-exotica*, que recibe tres parámetros numéricos,* a, b, c *y regresa el resultado de la operación:* $a^2 + b^3 + c^4$*. Usa* let *para almacenar los resultados intermedios. También puedes hacer una función auxiliar que reciba dos números como parámetros,* n *y* m *y regrese* n^m*.*

9.4.3. La forma especial *if*

La tercera forma especial que veremos, además de defun y let, es la condicional if. Esta forma se usa para enseñar a la computadora a tomar decisiones. No es indispensable para escribir funciones, pero su uso es muy común y por eso está aquí. La función beginning-of-buffer la usa, por ejemplo.

La idea básica detrás de if es que “si (if) algo es verdadero, entonces algo se hace”, por ejemplo, “si el cielo está despejado y hace calor, entonces ve a la playa”. En elisp no usamos ninguna palabra para designar la parte del “entonces”, por lo que las construcciones if lucen así:

```
(if prueba-falsa-o-verdadera
    realizar-esto-si-es-verdadera)
```

La primera parte es conocida como la parte *if* y la segunda es conocida como la parte *then*. Por ejemplo, evalúa la siguiente expresión:

```
(if (> 5 4)
    (message "Wow, 5 es mayor que 4."))
```

La función $>$ prueba si el primer argumento es mayor que el segundo y regresa verdadero si lo es.

En la vida real, claro está, la prueba no es fija, usualmente depende de una variable. Por ejemplo,

```
1  (defun tipo-de-animal (carácter)
2    "Imprime un mensaje en el área de eco dependiendo de CARÁCTER.
3     Si es el símbolo 'feroz, entonces avisa que es un tigre."
4    (if (equal carácter 'feroz)
5        (message "Es un tigre")))
```

imprime el mensaje “Es un tigre” dependiendo de si su parámetro es el símbolo 'feroz o no.

Las expresiones *if* reciben opcionalmente un tercer parámetro, conocido como la parte *else* y se evalúa cuando la prueba es falsa. Así, podemos completar nuestra función tipo-de-animal para que regrese algo más creativo que nil cuando no encuentra la característica feroz.

```
1 (defun tipo-de-animal (carácter)
2   "Imprime un mensaje en el área de eco dependiendo de carácter.
3 Si el el símbolo 'feroz, entonces avisa que es un tigre."
4   (if (equal caract 'feroz)
5       (message "Es un tigre")
6     (message "Ummm no, lo siento, no es feroz.")))
```

Vamos no seas tímido, evalúa nuevamente la función y pruébala con 'feroz, 'amistoso, y otros símbolos que se te ocurran.

Valores de verdad en elisp

Hasta el momento hemos hablado de falso y verdadero como si fueran nuevos tipos de datos en elisp. La realidad es que falso está representado por nuestro viejo amigo, nil; cualquier otra cosa – en serio, cualquier valor distinto de nil – es verdadero.

La expresión que prueba si algo es verdadero se interpreta como verdadera si el resultado de la evaluación no es nil. Como ya habíamos mencionado, nil también significa la lista vacía, ( ). Es decir, para el intérprete de elisp, nil y ( ) son la misma cosa, pero los humanos tendemos a referirnos a nil como falso y a ( ) como la lista vacía.

Cabe notar que si el intérprete de elisp no puede regresar algún valor útil para representar verdadero, entonces regresará el símbolo t, que significa verdadero (la "t" es de true).

9.4.4. *save-excursion*

La función save-excursion es la cuarta y última forma especial que revisaremos en este material.

En elisp es común hacer programas para editar; la función save-excursion es comúnmente usada, ya que guarda las posiciones del punto y la marca, ejecuta el cuerpo de la función y luego regresa el punto y la marca a sus valores previos, si es que fueron cambiados por la acción de la función. Su único propósito es evitar darle sorpresas inesperadas o molestas al usuario, cambiando a sus espaldas la posición del punto o la marca.

En Emacs es común que una función cambie la posición del punto y la marca como parte integral de su funcionamiento, aun cuando el usuario pueda o no esperar tal comportamiento. Por ejemplo, la función count-lines-region mueve el punto. El uso de save-excursion es una buena práctica de programación.

save-excursion incluso recupera los valores de la marca y el punto, aun en el eventual caso de que algo salga mal, o en el caló usado en elisp, incluso cuando la *función termine de manera anormal*. Asimismo, esta función también regresa al buffer de partida en caso de que la función cambie de buffer internamente para trabajar.

La forma general de save-excursion es:

```
(save-excursion
  cuerpo ...)
```

aunque es usual que save-excursion ocurra dentro de una expresión let, como en:

```
(let varlist
     (save-excursion
       cuerpo ...))
```

9.4.5. Resumen

En las últimas secciones hemos mostrado algunas funciones y formas especiales. Aquí las describiremos brevemente, para que te acuerdes rápidamente cómo se usan y también mencionaremos algunas funciones más que son parecidas.

eval-last-sexp Evalúa la última expresión simbólica anterior a la posición actual del punto. Su valor aparece en el área de eco, a menos que la función sea llamada con un argumento, en cuyo caso el resultado se inserta en el buffer actual. Este comando está usualmente ligado a C-x C-e.

defun *Define function*. Esta forma especial tiene hasta cinco partes: el nombre, un esqueleto para los argumentos que serán pasados a la función, documentación, una declaración interactive opcional y el cuerpo de la definición.

Por ejemplo:

```
(defun back-to-indentation ()
  "Apunta al primer char visible en la línea."
  (interactive)
  (beginning-of-line 1)
  (skip-chars-forward " \t"))
```

interactive Le declara al intérprete que la función puede ser usada interactivamente. Esta forma especial puede ir seguida de una cadena con una o más partes que pasan información a los argumentos de una función, en secuencia. Estas partes también le pueden indicar al intérprete que solicite información del usuario. Las partes de la cadena se separan por medio de nuevas líneas, \n.

Algunos carácteres de control comunes son:

- **b** El nombre de un buffer existente.
- **f** El nombre de un archivo existente.
- **p** El argumento (numérico) prefijo. (Nota que esta "p" es minúscula).
- **r** El punto y la marca, pasados como dos argumentos numéricos, el más pequeño (cualquiera que éste sea) primero. Éste es el único carácter de control que especifica dos argumentos sucesivos en lugar de uno solo.

let Declara que una lista de variables será usada dentro del cuerpo del let y les da un valor inicial; por omisión, dicho valor es nil; después evalúa las demás expresiones en el cuerpo del let y regresa el valor de regreso de la última. Dentro del cuerpo del let, el intérprete de elisp no ve los valores de las variables cuyos nombres ya estaban ligados fuera del let[8].

[8] Esto representa un concepto muy importante en lenguajes de programación, llamado *alcance léxico*, pero

Por ejemplo,

```
(let ((foo (buffer-name))
      (bar (buffer-size)))
  (message "Este buffer es %s y tiene %d carácteres."
           foo bar))
```

save-excursion Almacena los valores del punto, la marca y el buffer actual antes de evaluar el cuerpo de la forma. Los restaura una vez terminada la ejecución de la última expresión en su cuerpo.

Por ejemplo,

```
(message "Somos %d carácteres hasta este punto (en este buffer.)"
         (- (point)
            (save-excursion
              (goto-char (point-min)) (point))))
```

if Evalúa el primer argumento de la función; si es verdadero, evalúa el segundo argumento; en caso contrario, evalúa el tercero (si existe).

La forma especial if se llama *condicional*. Hay otras formas condicionales en elisp, pero if es probablemente la más usada.

Por ejemplo,

```
(if   (string= (int-to-string 21)
               (substring (emacs-version) 7 9))
    (message "Esta es la versión 21 de Emacs")
  (message "Esta NO es la versión 21 de Emacs"))
```

equal Prueba si dos objetos son el mismo. equal regresa verdadero si los objetos
eq tienen la misma estructura y contenidos. eq regresa verdadero si los dos objetos son en realidad el mismo objeto.

< <= La función **<** verifica si su primer argumento es menor que el segundo; las demás revisan
> >= respectivamente menor o igual, mayor y por último mayor o igual. Las cuatro funciones reciben argumentos numéricos.

message Imprime un valor en el área de eco. El mensaje sólo puede ser de una línea de longitud. El primer argumento es una cadena que puede contener '%s', '%d', o '%c' para imprimir el valor de argumentos que sigan a la cadena. El argumento usado por '%s' debe ser una cadena. El argumento usado por '%d' debe ser un número. El argumento usado por '%c' debe ser un número y será impreso como el carácter con ese valor en el código ASCII.

setq La función setq asigna el valor de su primer argumento al valor del segundo.
set El primer argumento es citado automáticamente por setq. Hace lo mismo para subsecuentes parejas de valores. Otra función set sólo toma dos argumentos y los evalúa ambos antes de asignar el valor del segundo al primero.

buffer-name Sin argumentos, regresa el nombre del buffer actual, como cadena.

---

ya tendrás muchos semestres para dominar toda esa terminología; por el momento tendrás que conformarte – y nosotros también – con entender el concepto.

buffer-file-name Sin argumentos, regresa el nombre del archivo que el buffer actual está visitando.

current-buffer Regresa el buffer en el cual Emacs está activo; puede ser algún buffer distinto al que está visible en la ventana de Emacs.

other-buffer Regresa el buffer más recientemente seleccionado (uno distinto al que se pasa como parámetro a other-buffer y al buffer actual).

switch-to-buffer Selecciona un buffer para que Emacs esté activo en él, y lo despliega en la ventana para que los usuarios puedan verlo. Usualmente está ligada a C-x b.

set-buffer Cambia la atención de Emacs a un buffer donde los programas correrán. No altera lo que se está viendo en la ventana.

buffer-size Regresa el número de carácteres en el buffer actual.

point Regresa el valor de la posición actual de punto, como entero; esto es igual al número de carácteres desde el inicio del buffer.

point-min Regresa el mínimo valor permisible para el punto en el buffer actual. Esto es 1, a menos que *narrowing* esté activo.

point-max Regresa el máximo valor permisible para el punto en el buffer actual. Esto es el final del buffer, a menos que *narrowing* esté activo.

Actividad 9.36 *Escribe una función no interactiva que duplique el valor de su argumento. Después haz la función interactiva.*

Actividad 9.37 *Escribe una función que pruebe si el valor actual de* fill-column *es mayor que el argumento pasado a la función, y si lo es, que mande el mensaje apropiado al área de eco.*

9.5. Algunas funciones relacionadas con buffers

En esta sección estudiaremos varias funciones de Emacs. Esto usualmente se llama "walk through", que sería más o menos equivalente a "un paseo por". La idea es mostrarte ejemplos *de verdad* de todo lo que hemos visto. Aprender viendo es una técnica muy usada en programación y estamos seguros que te servirá mucho, así que siéntate cómodamente, respira hondo y pon atención.

Las funciones que analizaremos aquí están todas relacionadas con buffers; más adelante veremos otro tipo de funciones. Describiremos cada función, en su momento, de manera breve; si te interesa ver la documentación completa de la función, puedes usar C-h f *< nombre de la función>* Enter; o si es una variable C-h v *<nombre de la variable>* Enter.

9.5.1. Definición (simplificada) de *beginning-of-buffer*

La función beginning-of-buffer es una buena función para comenzar nuestro paseo por las funciones de Emacs, ya que es sencilla y a estas alturas ya debes estar familiarizado con ella. Cuando

se usa interactivamente, beginning-of-buffer mueve el cursor al inicio del buffer, dejando la marca en la posición previa. Usualmente está ligada a M-<.

Sin más, aquí está nuestra definición simplificada (es simplificada porque la versión real maneja una opción compleja además del comportamiento que hemos descrito):

```
1     (defun simplified-beginning-of-buffer ()
2       "Mueve el punto al inicio del buffer;
3     dejando la marca en la posicion actual."
4       (interactive)
5       (push-mark)
6       (goto-char (point-min)))
```

Nota que tiene sus cinco partes como toda buena definición. La parte que no conoces aún, es el cuerpo de la función, que en este caso consta de dos expresiones:

```
5       (push-mark)
6       (goto-char (point-min))
```

La primera de ellas, (push-mark), al ser evaluada por el intérprete lo que hace es poner la marca en la posición actual del cursor, en donde sea que éste esté. Dicha posición se guarda en el anillo de marcas (Emacs tiene varios tipos de anillos, el de marcas, el de la muerte y otros y todos son muy útiles).

La siguiente línea, (goto-char (point-min)), mueve el cursor a la primera posición del buffer. ¿Por qué poner la marca en el lugar anterior? A primera vista no parece obvio, pero en la práctica uno usualmente desea regresar al inicio del buffer *temporalmente*, como por ejemplo para hacer un pequeño ajuste en las líneas iniciales, para revisar algún dato o alguna otra actividad. Sin embargo, el punto más importante es usualmente el de edición más reciente, así que poniendo la marca en él es fácil regresar a ese punto (prueba, por ejemplo, C-x C-x).

Cuando veamos funciones que desconozcas, por ejemplo goto-char, recuerda que puedes usar C-h f, seguido del nombre de la función, para terminar con Enter para obtener su documentación.

Actividad 9.38 *Escribe la función simplified-end-of-buffer.*

La definición de *mark-whole-buffer*

Esta función, mark-whole-buffer, no es más difícil que nuestra versión simplificada de beginning-of-buffer y, a diferencia de aquélla, ésta no está simplificada; veremos la versión real.

Lo que hace mark-whole-buffer es poner el punto al inicio del buffer y la marca al final; no es tan usada como las anteriores, pero es importante. Usualmente está ligada a C-x h. Luce así:

```
1 (defun mark-whole-buffer ()
2   "Pone el punto al inicio y la marca al final del buffer."
3   (interactive)
4   (push-mark (point))
5   (push-mark (point-max))
6   (goto-char (point-min)))
```

En esta función, las últimas dos expresiones ponen la marca al final del buffer y la otra mueve el punto a la primera posición. Pero, ¿por qué aparece (push-mark (point)) como primera línea del cuerpo? Nuevamente, la idea detrás de esto es que la posición actual (antes de llamar a mark-whole-buffer) es la posición de edición, la más importante actualmente en el buffer; la práctica indica que cuando uno marca todo el buffer, lo hace con la intención de reformatearlo, re-indentarlo, copiarlo, o algo similar, pero después de eso, uno desea regresar a la posición previa de edición.

¿Cómo regresamos a esa posición? Pues es muy sencillo. Recuerda que dijimosq que push-mark empuja la marca a un anillo que puedes recorrer; por omisión C-x C-x te lleva a la posición previa de la marca, y ya sabes que C–Space pone la marca. Pues C–Space, que en realidad es un ligado de la función set-mark-command, hace más que eso: si le pasas un argumento prefijo, *brinca* a esa marca en el anillo local de marcas; ¡vamos, inténtalo! ¡Ejecuta dos veces C-u C–Space y ve lo que sucede!

La definición de *append-to-buffer*

La función append-to-buffer copia una región (la región actual en el buffer actual) a un buffer especificado. Usaremos la función insert-buffer-substring para copiar la región; como es de esperarse, esta última recibe dos posiciones, la inicial y la final, y el buffer del cual debe extraer la subcadena comprendida entre dichas posiciones.

Esas posiciones no son otras que el punto y la marca (que rodean la región) en el buffer actual; aquí está nuestra definición:

```
(defun append-to-buffer (buffer start end)
"Agrega el texto de la región al BUFFER.
Es insertado en BUFFER antes de su punto.
Cuando es llamada desde un programa recibe tres argumentos:
un buffer o su nombre, y dos numeros especificando el area
a copiar."
  (interactive "BAgregar al buffer: \nr")
  (let ((oldbuf (current-buffer)))
    (save-excursion
      (set-buffer (get-buffer-create buffer))
      (insert-buffer-substring oldbuf start end))))
```

Analiza parte por parte la función y verás que no hay muchas cosas nuevas; lo que si tiene es una mezcla funcional de varias cosas que vimos muy por encima. Primero te conviene probarla para que veas cómo funciona y eso aclarará mucho. Visita un archivo de prueba en un buffer, marca una región (pon la marca y mueve el punto un par de líneas abajo, por ejemplo). Ahora regresa al buffer *scratch*, copia y evalúa esta función y pon el punto al final del buffer.

A continuación, regresa al buffer del archivo que acabas de abrir (donde pusiste la marca y el punto anteriormente) y llama a la función: M-x append-to-buffer. Te pide un buffer, te propondrá el más reciente (si hiciste lo que acabo de decir, debe ser *scratch*); presiona Enter y verás aparecer la región en el buffer *scratch*.

Muy bien. Ahora, ¿qué es lo que pasó? Vamos por partes:

1. Sólo necesitamos tres argumentos, el nombre del buffer al que vamos a copiar (destino) y las posiciones del punto y la marca en el buffer fuente. Nota que no necesitamos también el nombre del buffer fuente, porque para que la función haga lo que esperas que haga, tienes – necesariamente – que estar en dicho buffer.
2. La forma `( interactive "BAgregar al buffer: \nr")`, solicita un buffer (el destino) y ya sabes que `r` pasa como dos argumentos consecutivos las posiciones del punto y la marca. No sólo eso, `r` siempre pasa primero el más pequeño, así que cuando llamemos a insert-buffer-substring, que espera su segundo argumento sea menor o igual que su tercer argumento, no tendremos ningún problema.
3. El cuerpo consta de una expresión let, que es la primera que vemos en acción. La usamos para guardar el buffer actual (el fuente) en una variable local llamada oldbuf. ¿Por qué? Porque tenemos que cambiar de buffer (al destino) para poder insertar la subcadena.
4. Nota también el uso de save-excursion, que con la explicación del punto anterior ya debe ser obvio. Lo usamos aquí (antes de cambiar de buffer) para que guarde la información adecuada y nos regrese aquí cuando acabe la función.
5. set-buffer es nueva, aunque ya la habíamos explicado en el la sección 9.4.5. Lo que hace es cambiar la atención de Emacs (si no está en la ventana, no aparecerá por la acción de esta función) al buffer que recibe como parámetro. En este caso, le dice a Emacs, `( set-buffer ( get-buffer-creat` que se cambie al buffer destino, pero ¡oh sorpresa! aquí nos encontramos con otra función nueva: get-buffer-create, y, para no variar con los nombres largos y auto-explicativos de Emacs, esta función recibe una cadena o un buffer como argumento. Si es un buffer se mueve a ese buffer; si es una cadena, busca un buffer con ese nombre y se mueve a él; finalmente, si es una cadena pero no hay ningún buffer en Emacs con ese nombre, crea uno nuevo y se mueve a él.
6. Por último, tenemos `( insert-buffer-substring oldbuf start end)` que lo único que hace es cortar la subcadena comprendida entre start y end en nuestro buffer origen (recuerda que guardamos su contenido en la variable oldbuf) y luego insertarla en la posición actual.

9.5.2. Resumen

Aquí está un resumen de las funciones que acabamos de revisar:

describe-function Imprimen la documentación de una función o una variable.
describe-variable Usualmente están ligadas a C-h f y C-h v, respectivamente.

push-mark Pone la marca en una localidad particular y almacena el valor de la marca anterior en el anillo de marcas. La marca es una posición en el buffer que mantendrá su posición relativa aun cuando sea agregado o eliminado texto del buffer.

goto-char Pone el punto en la posición especificada por el valor de su argumento, que puede ser un número, un marcador, o una expresión que regresa el número de una posición, tal como (point-min).

insert-buffer-substring Copia una región de texto de un buffer, que se pasa como argumento a la función en el buffer actual.

mark-whole-buffer Marca todo el buffer como una región. Usualmente está ligado a C-x h.

set-buffer Cambia la atención de Emacs a otro buffer, pero no cambia la ventana que está siendo desplegada. Se usa cuando un programa (a diferencia de un humano) va a trabajar en un buffer distinto.

get-buffer-create **get-buffer** Encuentra un buffer nombrado o crea uno si un buffer con ese nombre no existe. La función get-buffer regresa nil si el buffer nombrado no existe.

Actividad 9.39 *Usa* if *y la función* get-buffer *para construir una función interactiva que imprima un mensaje diciendo si un buffer dado existe o no.*

9.6. Funciones fundamentales: *car, cdr, cons*

En elisp, car, cdr y cons son funciones fundamentales. La función cons se usa para construir listas, mientras que las funciones car y cdr se usan para descomponerlas.

Retrasamos la discusión de estas funciones hasta este momento, porque para hacer cosas interesantes usando estas funciones fundamentales era necesario que conocieras las formas especiales que revisamos y que manejaras de mejor forma buffers, cadenas, etc.

La función cons es una abreviatura de la palabra *construct* y sirve justamente para construir listas. Las otras dos funciones, desgraciadamente, tienen nombres esotéricos, y desafortunados porque no significan absolutamente nada útil; por razones históricas se siguen usando y en funciones de Emacs siguen apareciendo, así que aquí también los usaremos. Pero puedes pensar en ellas como *primer elemento*, el car, y el *resto de la lista*, el cdr (que por cierto, se pronuncia *could-er* en inglés).

También presentaremos rápidamente varias funciones utilitarias más, como length, setcar, nthcdr y nthcdr. Culminaremos la sección y este breve curso de Emacs elisp revisando copy-region-as-kill, una función de Emacs muy útil y relativamente más compleja que todas las anteriores.

9.6.1. *car* y *cdr*

El car de la lista es simplemente el primer elemento de la lista. El car de la lista (rosa violeta clavel azucena) es rosa. Intenta evaluar lo siguiente:

```
(car '(rosa violeta clavel azucena))
```

Verás que el símbolo rosa aparece en el área de eco. Un mejor nombre sería first (y de hecho este nombre también existe en Emacs elisp; está definido como un alias de car en el paquete cl.el).

El car no remueve el primer elemento de la lista, esto es, después de aplicar car a una lista, la lista sigue siendo la misma. En el caló de elisp, esto significa que car es una función *no destructiva*.

cdr de una lista es el resto de la lista, esto es, cdr regresa la parte de la lista que sigue al primer elemento. De tal forma que (cdr '(rosa violeta clavel azucena)) regresa (violeta clavel azucena).

Al igual que car, cdr no remueve ningún elemento de la lista. Claramente, un mejor nombre para cdr sería rest y al igual que first, rest también existe (en el paquete cl.el) y es un alias de cdr.

Cuando car y cdr se aplican a listas de símbolos (como en los ejemplos anteriores) el elemento de la lista regresado por car es rosa sin paréntesis, como símbolo. Sin embargo, el cdr de la lista es siempre otra lista.

Por otro lado, en una lista de listas, el primer elemento es una lista. car regresa este primer elemento como lo que es: una lista. Por ejemplo, evalúa el siguiente ejemplo:

```
(car '((león tigre chita)
       (gazela antílope zebra)
       (ballena delfín tiburón)))
```

Esta función regresa la lista de animales carnívoros, (león tigre chita), mientras que el cdr de esa lista es la lista: ((gazela antílope zebra) (ballena delfín tiburón)).

cons

La función cons construye una lista; es la inversa de car y cdr. Por ejemplo, cons puede usarse para hacer una lista de cuatro elementos a partir de una lista de sólo tres elementos, agregando uno al principio así:

```
(cons 'rosa '(violeta clavel azucena))
```

Después de evaluar esa expresión verás a nuestra querida y vieja amiga, la lista (rosa violeta clavel azucena) aparecer en el área de eco.

cons siempre requiere una lista como segundo argumento y lo que regresa es una nueva lista cuyo car es el primer argumento y cuyo cdr es su segundo argumento (la lista). No puedes empezar una lista a partir de nada, necesitas empezarla a partir de la lista vacía. Por ejemplo, evalúa las siguientes expresiones en secuencia:

```
(cons 'azucena ())
(cons 'clavel '(azucena))
(cons 'violeta '(clavel azucena))
(cons 'rosa '(violeta clavel azucena))
```

Nota que en la primera expresión estamos usando a () como la lista vacía, pero recuerda que es exactamente lo mismo que usar nil.

Encontrar la longitud de una lista: *length*

Puedes usar la función length de elisp para saber cuántos elementos tiene una lista, como en los siguientes ejemplos:

```
(length '(azucena))
(length '(daisy buttercup))
```

```
(length ())
```

Los valores que obtendrás una vez que evalúes las expresiones, representan el número de elementos en cada lista, 1, 2 y 0 respectivamente.

nthcdr

La función nthcdr está asociada con la función cdr. Lo que hace es aplicar cdr repetidamente. Si sacas el cdr de la lista (rosa violeta clavel azucena) obtendrás la lista (violeta clavel azucena).

Si ahora sacas el cdr de esta nueva lista, obtendrás (clavel azucena); (nota que si sacas dos veces el cdr de la misma lista, obtendrás lo mismo, ya que es no destructiva). Es decir, si evalúas (cdr (cdr (rosa violeta clavel azucena))) obtendrás (clavel azucena).

La función nthcdr hace lo mismo que repetidas llamadas a cdr anidadas. En el siguiente ejemplo, el argumento 2 se pasa a la función nthcdr, junto con la lista, y el valor que regresa es la lista sin los dos primeros elementos, que es lo mismo que aplicar dos veces cdr en cadena.

```
(nthcdr 2 '(rosa violeta clavel azucena))
```

Actividad 9.40 *Usando la lista original,* (rosa violeta clavel azucena)*, aplica* nthcdr *pasándole argumentos numéricos del 0 al 5. ¿Puedes explicarlo? Si no puedes, entonces aplica el mismo número de veces (de manera anidada)* cdr *a la misma lista. Ahora sí, explícalo.*

setcar

Como te puedes imaginar por sus nombres, las funciones setcar y setcdr (que veremos en un momento) asignan nuevos valores al car y al cdr de una lista. En realidad cambian la lista original, a diferencia de car y cdr que son no destructivas. Una forma de entender cómo funciona esto es experimentando. Empezaremos con la función setcar.

Primero, podemos hacer una lista y luego asignar a una variable dicha lista, utilizando la función setq. Aquí está una lista de animales:

```
(setq animales '(girafa antilope tigre leon)).
```

Evalúa las siguientes expresiones. Una vez evaluadas, puedes confirmar que la variable animales ahora apunta a la lista (girafa antilope tigre leon). A continuación, evalúa la función setcar pasándole dos argumento, la variable animales y el símbolo citado hipo; es decir, evalúa:

```
(setcar animales 'hipo)
```

Verás el símbolo hipo aparecer en el área de eco. Ahora evalúa nuevamente la variable animales y notarás que la lista ha cambiado.

setcdr

La función setcdr es similar a la función setcar, excepto que esta función cambia el segundo y subsecuentes elementos de una lista en lugar del primero.

Por ejemplo, asigna el valor de una variable animales−domesticables, evaluando la siguiente expresión:

```
(setq animales-domesticables '(caballo vaca oveja perro))
```

Ahora, evalúa:

```
(setcdr animales-domesticables '(gato cocodrilo))
```

Verás la lista (gato cocodrilo) aparecer en el área de eco. Ése es el valor de regreso de la función; ahora verifica su efecto secundario, evaluando animales−domesticables.

Actividad 9.41 *Construye una lista de aves evaluando varias expresiones* cons*. Guarda esa lista en una variable, con* setq*. Investiga qué pasa cuando haces un* cons *de una lista con ella misma; por ejemplo, qué pasa cuando haces* (cons aves aves)*, suponiendo, claro, que la variable donde guardaste tu lista se llama* aves*. Reemplaza la primera ave de tu lista con un pez. Reemplaza el resto de las aves en la lista con peces.*

9.6.2. Cortando y guardando texto

Cuando matas texto en un buffer, con alguno de los comandos que matan texto, Emacs guarda ese texto en una lista para que puedas traerlo de nuevo a la vida con alguno de los comandos que hacen *yank*. Por lo que la lista puede lucir más o menos así:

```
("un pedazo de texto" "texto final")
```

La función cons puede usarse para añadir una pieza de texto a esta lista, como en:

```
(cons "otra pieza de texto" '("un pedazo de texto" "texto final"))
```

Y después, con un uso apropiado de car y nthcdr puedes obtener de vuelta todas las secciones de texto almacenadas en la lista. Por ejemplo, si quisieras el segundo elemento de la lista, podrías evaluar:

```
(car (nthcdr 1 '("otra pieza de texto"
                 "un pedazo de texto"
                 "texto final")))
```

que debe mostrar "un pedazo de texto".

Las funciones de Emacs que hacen kill y yank en realidad son más complejas. Para empezar, el anillo de la muerte (equivalente a nuestra lista en el ejemplo anterior) es circular (de ahí el nombre de anillo) por lo que una vez que llegas al último elemento, si quieres seguir buscando partes de texto en el anillo, te debe llevar al primero de la lista.

La función principal de este capítulo tiene que ver con el anillo de la muerte y cómo se usa. Para que te vayas aclimatando, primero te mostraremos una función llamada zap–to–char.

zap-to-char

La función zap-to-char elimina el texto en la región comprendida entre la localización del cursor (i.e. el punto) y la siguiente aparición de un carácter (incluido este último) dado. El texto que zap-to-char elimina se guarda en el anillo de la muerte; y puede sacarse de ahí con C-y (yank). Si se le pasa un argumento prefijo, entonces elimina el texto hasta encontrar ese número de presencias del carácter.

Entonces, si el cursor estuviera al inicio de esta oración y el carácter fuera *s*, **Entonces**, sería eliminada. Si el argumento prefijo fuese 2, '**Entonces, s**' sería eliminado.

Para determinar cuánto texto tiene que ser eliminado, zap-to-char utiliza una función de búsqueda. Las búsquedas se usan extensivamente en el código que manipula texto en Emacs, por lo que mientras revisamos estas funciones te rogamos pongas especial atención en la función de búsqueda y en la de borrado.

```
1  (defun zap-to-char (arg char)
2    "Mata el texto hasta la ARG-ésima aparición de CHAR.
3     Va hacia atrás si ARG es negativo. Marca un error si CHAR
4     no es encontrado."
5    (interactive "*p\ncZap to char: ")
6    (kill-region (point)
7                 (progn
8                   (search-forward (char-to-string char) nil nil arg)
9                   (point))))
```

Analicemos ahora la función por partes:

1. La sección interactiva, `(interactive "*p\ncZap to char: ")`, nos indica tres cosas:
   a) El '*' le dice a Emacs que marque un error si el buffer está protegido contra escritura. O en otras palabras, si es de sólo lectura. Esto es obvio, porque si no podemos editarlo, no podemos borrar texto alguno.
   b) La 'p' le dice a Emacs que la función recibirá como primer argumento el argumento prefijo. Éste ya lo conocías.
   c) El \n separa los argumentos y la c le dice a Emacs que debe pedir un carácter, utilizando para ello el prompt: `Zap to char: `.
2. El cuerpo de la función contiene la función que mata (esto es, que elimina) el texto de la región desde la posición actual (la primera presencia de `(point)`) hasta la posición (incluida ella) donde se encuentra el carácter. Esto es la primera parte:

   ```
   6      (kill-region (point) ...
   ```

   La siguiente parte del código utiliza la función progn. El cuerpo de progn consiste de llamadas a search-forward y nuevamente a point. Es más fácil entender el funcionamiento de progn una vez que entiendas cómo funciona la búsqueda hacia adelante, search-forward, así que primero veremos esta última y luego regresamos a progn.
3. La función search-forward se usa para localizar el carácter que marca la posición hasta la que queremos cortar el texto. Si la búsqueda es exitosa, deja el punto inmediatamente después del último carácter en la cadena objetivo (en este caso, la cadena objetivo es sólo de un carácter de largo). Si la búsqueda es hacia atrás, deja el punto justo antes del primer carácter en la cadena objetivo. Además, search-forward regresa t para señalar verdadero, por lo que mover el punto es un efecto secundario.
   En zap-to-char, la llamada a la función search-forward luce como:

   ```
   8    (search-forward (char-to-string char) nil nil arg)
   ```

   La función search-forward toma cuatro argumentos:
   a) El primer argumento es el objetivo, aquello que estamos buscando. Éste debe ser una cadena, tal como "s".
   Nota que en elisp, un carácter no es visto (automáticamente) como una cadena de un solo carácter. Por eso usamos `(char-to-string char)` para transformar el carácter a una cadena, como search-forward lo quiere.

b) El segundo argumento limita la búsqueda; es una posición específica en el buffer. En este caso, la búsqueda puede ir hasta el final del buffer, por lo que no estamos limitando la búsqueda y entonces usamos `nil`.

c) El tercer argumento le dice a la función qué debe hacer si la búsqueda falla – puede señalar un error (e imprimir un mensaje) o puede regresar `nil`. Usar `nil` como tercer argumento provoca que la función marque un error.

d) El cuarto argumento para search-forward es una cuenta de repetición – cuántas presencias de la cadena debe buscar. Este argumento es opcional y si la función se llama sin una cuenta de repetición, la función recibe por omisión 1. Si este argumento es negativo, la búsqueda va hacia atrás.
En su forma general, search-forward se ve así:

```
(search-forward   "cadena objetivo"
                  límite-de-la-búsqueda
                  que-hacer-si-la-búsqueda-falla
                  cuenta-de-repetición)
```

4. La función progn es tal que provoca que todos sus argumentos sean evaluados en secuencia y luego regresa el valor del último. Las expresiones previas (a la última) se evalúan sólo por sus efectos secundarios, ya que sus valores de regreso se descartan.
Su forma general es:

```
(progn
  cuerpo...)
```

En zap-to-char, la expresión progn tiene que hacer dos cosas: poner el punto en la posición correcta y luego regresar la posición de ese punto para que kill-region pueda hacer su trabajo. El primer argumento de progn es search-forward. El segundo argumento es (point). Esta expresión regresa el valor del punto, que en este caso será la localización a la que se ha movido search-forward.

De acuerdo, de acuerdo, fue una explicación muy larga. Lo sentimos, pero no conocías search-forward y progn, que son dos funciones muy útiles y muy usadas en elisp. Así que aquí te va un resumen de cómo funciona zap-to-char ahora que sabes cómo funcionan todas las funciones auxiliares:

El primer argumento pasado a kill-region es la posición del punto en el momento que ejecutas zap-to-char. Dentro del progn, la función de búsqueda mueve el punto justo después del carácter hasta el que queremos cortar; point regresa el punto de esta nueva posición. La función kill-region pone estos dos valores juntos, el primero como inicio de la región y el segundo como el punto final de ésta y elimina la región.

La función progn es necesaria porque kill-region recibe sólo dos argumentos; y fallaría si search-forward y point fueran escritas en secuencia, en lugar de como argumentos de progn. Es decir que progn es el segundo argumento de kill-region y su valor de regreso es el valor que regresa `(point)`.

Obviamente, ésta no es la única manera de hacer esta función: bien pudimos haber guardado el valor regresado por search-forward en una variable local (con let) y luego usarlo en kill-region.

Pero esta forma que hemos revisado, es la que se usa desde la versión 19 de Emacs y es más elegante, además de que muestra el uso de una nueva función que te servirá mucho: progn.

delete-region: nota cultural

La función zap-to-region que acabamos de ver utiliza a la función kill-region; ésta a su vez utiliza otras dos funciones, copy-region-as-kill, que veremos en un minuto, y delete-region. La función delete-region es interesante y muy sencilla; lo único que hace es borrar texto de forma tal que no puedas recuperarlo.

Sin embargo, delete-region no está escrita en Emacs-elisp sino en C. Es una de las primitivas de Emacs.

9.6.3. Inicialización de variables: *defvar*

Ésta es la última parada antes de poder explicar copy-region-as-kill y es importante, porque la variable kill−ring, el lugar donde se guardan todas las secciones de texto que se matan, se crea con defvar.

La función defvar viene de *define variable* y es una forma especial, similar a setq ya que asigna un valor a una variable. Sin embargo, es distinta a setq en dos aspectos: primero, sólo asigna un valor a una variable si la variable aún no tiene uno; si la variable ya tiene un valor, defvar no sobreescribe su valor actual. Segundo, defvar tiene una cadena que sirve para documentar la variable.

Para ver el valor actual de una variable cualquiera, usamos describe-variable, usualmente ligada a C-h v. Por ejemplo, checa lo que dice C-h v cuando le pasas kill-ring. Verás dos cosas importantes: primero su valor, que puede ser muy pequeño o muy grande, dependiendo de cuánto hayas matado en la sesión de Emacs hasta el momento; verás también su documentación, que comienza más o menos así:

```
Documentation:
List of killed text sequences.
```

La definición de kill-ring usando defvar puede ser:

```
(defvar kill-ring nil
  "List of killed text sequences.")
```

Como puedes notar, su valor inicial es nil, que tiene sentido, ya que si no has guardado nada en el anillo no esperarías que cuando quieras pegar (recuperar del anillo, o revivir)[9], algo apareciera, ¿o sí?

[9] No hay traducción exacta para *yank*, pero en notación de muchos otros editores de texto, equivaldría a pegar, sólo que es mucho más poderoso porque tiene memoria.

9.6.4. *copy-region-as-kill*

La función copy-region-as-kill copia una región de texto del buffer en una variable que se llama kill −ring. Si llamas a kill-region-as-kill inmediatamente después de haber ejecutado un kill-region, Emacs junta la sección de texto recién copiada con la anterior. Esto significa que si haces un yank después, obtendrás todo el texto de las últimas dos operaciones de muerte. Por otro lado, si algún otro comando precede a la acción de copy-region-as-kill, la función copia el texto en una entrada separada en el anillo de la muerte.

Aquí está el código de la función. Probablemente en versiones muy recientes de Emacs sea distinto; de cualquier forma esta versión es lo suficientemente compleja e interesante para que la analicemos aquí.

```
1  (defun copy-region-as-kill (beg end)
2  "Salva la región como si hubiese sido matada, pero no la mata."
3  (interactive "r")
4  (if (eq last-command 'kill-region)
5      (kill-append (buffer-substring beg end) (< end beg))
6    (setq kill-ring
7          (cons (buffer-substring beg end) kill-ring))
8    (if (> (length kill-ring) kill-ring-max)
9        (setcdr (nthcdr (1- kill-ring-max) kill-ring) nil)))
10 (setq this-command 'kill-region)
11 (setq kill-ring-yank-pointer kill-ring))
```

Si revisas con cuidado la definición notarás que tiene sus cinco partes. Los dos argumentos, beg y end y el código "r" de interactive (que regresa la región, como dos argumentos numéricos consecutivos) deben darte una idea respecto a qué hacen referente al inicio (beg, por beginning) y fin (end) de la región. Los parámetros están en inglés, porque cuando comiences a ver el código de distintas funciones de Emacs por tu cuenta, verás que la región se usa exhaustivamente y curiosamente éstos son los nombre favoritos para nombres de parámetros en funciones que manejan la región.

El cuerpo de la función comienza con un if – línea 4 –. Nos sirve para distinguir dos tipos de situaciones: saber si el comando ejecutado inmediatamente antes que éste fue (o no) un kill-region. En caso de que haya sido, la nueva región muerta se pega a la anterior. En caso contrario, se inserta (la región muerta) al inicio del anillo de la muerte.

Las últimas expresiones son setq. Una asigna a la variable this −command el valor kill −region y la otra hace que la variable kill −ring−yank−pointer apunte al anillo de la muerte.

El cuerpo del copy-region-as-kill amerita una discusión más detallada y la mostramos a continuación:

El cuerpo

La función copy-region-as-kill fue escrita con sumo cuidado para poder:

- Fusionar uno o más comandos que matan en una sola pieza de texto que pudiera ser pegada nuevamente con un solo comando. No tienes idea lo útil que es esto, pero eventualmente lo sabrás.
- Todo lo que matas (o los comandos que matan hacia adelante) se añaden al final del texto copiado previamente, y los comandos que copian o matan texto en reversa se copian al principio del texto copiado previamente. De esta manera, las palabras en el texto (que se va pegando) se mantienen en el orden correcto.

La función hace uso de dos variables para mantener el rastro de los comandos actual y anterior que ejecutó Emacs. Las dos variables son this−command y last−command.

Normalmente, siempre que una función se ejecuta Emacs asigna esa función a this−command, en este ejemplo copy-region-as-kill. Al mismo tiempo, Emacs asigna el valor de last−command al valor anterior de this−command. Sin embargo, el comando copy-region-as-kill es diferente; asigna el valor de this−command a kill−region, que es el nombre de la función que llama a copy-region-as-kill.

En la primera parte del cuerpo de copy-region-as-kill, una expresión if determina si el valor de last−command es kill-region. Si lo es, la parte "entonces" del if se evalúa; usa la función kill-append para concatenar el texto copiado con esta llamada al texto que ya se encuentra en la primera posición del anillo de la muerte. Por otro lado, si el valor de last−command no es kill-region, entonces la función copy-region-as-kill agrega un nuevo elemento al anillo de la muerte.

La función *kill-append*

La función kill-append que se usa en la definición de copy-region-as-kill se muestra a continuación:

```
(defun kill-append (string before-p)
  (setcar kill-ring
          (if before-p
              (concat string (car kill-ring))
            (concat (car kill-ring) string))))
```

Como siempre, podemos analizarla por partes (nota que no tiene documentación). La función setcar utiliza concat para concatenar el nuevo texto al car del anillo de la muerte. Si pre-agrega o post-agrega, depende del resultado de la expresión if:

```
          (if before-p
              (concat string (car kill-ring))
            (concat (car kill-ring) string))
```

Si la región que se está matando está antes que la región que mató el último comando, entonces debemos agregar antes (pre-agregar) el texto; contrariamente, si el texto muerto sigue a lo que se mató con anterioridad, entonces debemos agregarlo después. La expresión if depende del predicado before−p para decidir si el texto nuevo debe ir antes o después de lo que está en el car del anillo de la muerte.

El símbolo before−p es el nombre de uno de los argumentos de kill-append. En la función copy-region-as-kill su valor se calcula con (< end beg), lo que es verdadero cuando el texto que

estamos cortando va antes que los que están en el car del anillo de la muerte (y por ende será pre-agregado).

Pre-agregar está representado por:

```
(concat string (car kill-ring))
```

Post-agregar por:

```
(concat (car kill-ring) string)
```

Para entender cómo funciona esto, primero necesitamos entender cómo funciona concat. La función concat liga o une dos cadenas de texto. El resultado es una cadena. Por ejemplo,

```
(concat "abc" "def")
```

regresa "abcdef".

Ahora sí, podemos entender rápidamente cómo funciona kill-append: modifica el contenido del anillo de la muerte. La función setcar cambia en realidad el primer elemento de la lista. Logra esto usando concat para reemplazar el viejo contenido del primer elemento del anillo con un nuevo primer elemento que se obtiene concatenando el viejo texto con el nuevo (el texto recién muerto, esto es). El nuevo texto se pone antes que el viejo o al revés, dependiendo de si estaba antes o después en el buffer, respectivamente.

La parte final de *copy-region-as-kill*

Muy bien; después de haber entendido cómo funciona copy-region-as-kill cuando el comando previo fue también un kill-region, ahora tenemos que ver la última parte: ¿qué pasa cuando el comando previo no fue kill-region?

Lo que hace es asignar como valor al anillo de la muerte una nueva lista que es igual a la anterior, pero ahora como primer elemento tiene el texto recién cortado. Esto se logra con la sección:

```
6     (setq kill-ring
7           (cons (buffer-substring beg end) kill-ring))
```

La siguiente parte es nuevamente una expresión if. Esta expresión sirve únicamente para mantener acotado el tamaño del anillo de la muerte; usualmente su valor es 30. Es decir, evita que el anillo de la muerte guarde más allá del valor que tenga kill-ring-max. Esto se logra con la sección de código:

```
8     (if (> (length kill-ring) kill-ring-max)
9         (setcdr (nthcdr (1- kill-ring-max) kill-ring) nil))
```

Lo que dice esta expresión if es que si pasa del tamaño permitido, entonces convierte en nil el último elemento del anillo de la muerte. Ya vimos las funciones setcdr y nthcdr así que esto no debe parecerte tan extraño. Salvo quizá por el uso de la función llamada 1-, que resta uno a su argumento.

A continuación siguen dos asignaciones en el código de copy-region-as-kill, la primera:

```
10  (setq this-command 'kill-region)
```

Nota que estas dos expresiones no son parte del if anterior, por lo que se ejecutan siempre que se ejecute la función. Lo único que hace es guardar el símbolo kill-region en la variable this-command, por lo que si volvemos a copiar algún texto inmediatamente después, éste será pegado al car del anillo de la muerte.

Finalmente, la última línea

```
11  (setq kill-ring-yank-pointer kill-ring)
```

actualiza el apuntador global kill-ring-yank-pointer que dice en todo momento cuál es el siguiente texto a revivir. Esta variable es usada por yank y yank-pop, pero, por cuestiones de espacio, ya no ceremos estas dos últimas funciones veremos aquí.

9.6.5. Resumen

Aquí está un resumen de las funciones que introdujimos recientemente:

car Regresa el primer elemento de una lista.

cdr Regresa el resto de la lista.

nthcdr Regresa el resultado de sacar el cdr 'n' veces a una lista.

cons Construye una lista agregando su primer argumento como primer elemento a la lista que recibe como segundo argumento. Si no recibe una lista como segundo argumento, entonces crea lo que se conoce como una lista impropia, aunque todo lo que se puede hacer con listas impropias se puede hacer con listas normales, así que no debería preocuparte.

setcar Cambia el primer elemento de una lista.

setcdr Cambia el resto de una lista.

progn Evalúa cada uno de sus argumentos en secuencia y regresa el valor del último, descartando todos los valores intermedios.

search-forward Busca una cadena y si la encuentra, mueve el punto. Recibe cuatro argumentos:

1. La cadena a buscar (objetivo).
2. Opcional, un límite para la búsqueda.
3. Opcional, qué hacer si falla, regresa nil o un mensaje de error.
4. Opcional, cuántas veces repetir la búsqueda; si el argumento es negativo, busca hacia atrás.

kill-region Corta el texto entre el punto y la marca del buffer y lo guarda en el anillo de la muerte, para que puedas revivirlo con yank.

delete-region Elimina el texto entre el punto y la marca del buffer y lo descarta. No puedes recuperarlo.

copy-region-as-kill Copia el texto entre el punto y la marca al anillo de la muerte para que puedas recuperarlo. La función ni corta ni mueve el texto del buffer.

Actividad 9.42 *Escribe una función interactiva que busque una cadena. Si la búsqueda encuentra la cadena, deja el punto después de ésta y despliega el mensaje "La encontré". (NO utilices* search-forward *como el nombre de tu función; si lo haces, sobre-escribirás la función existente que viene con Emacs. Usa un nombre como* busca.*)*

Actividad 9.43 *Escribe una función que imprima el tercer elemento del anillo de la muerte en el área de eco, si existe, claro. Si el anillo de la muerte no tiene un tercer elemento, manda un mensaje apropiado.*

9.7. Lluvia de funciones útiles de Emacs-elisp

Ponte cómodo y relájate, ya que en esta sección no hay ejercicios, sólo funciones y explicaciones breves. Sin embargo, si las tecleas y pruebas, estamos seguros que te divertirás, las entenderás mejor y te serán útiles como patrones para funciones futuras.

9.7.1. ¿Quién o qué es ASCII?

Es común imprimir la tabla de carácteres ASCII[10], pero obviamente hacerlo tecleando carácter por carácter no es precisamente una idea brillante. Por un lado son 128 carácteres (2^7) en el ASCII estándar y 256 en el extendido (2^8). Aquí está una función en elisp que lo hace por nosotros.

```
(defun show-ascii ()
  "Muestra el ASCII como una lista de parejas."
  (interactive)
    (let ((counter 0))
      (dotimes (i 256)
        (insert (format "(%d,%c) " i i))
        (when (eq counter 10)
          (insert "\n")
          (setq counter 0))
        (setq counter (incf counter)))))
```

Sólo evalúa la función; después muévete a un buffer que puedas editar y ejecuta M-x show-ascii y verás aparecer varias líneas, cada una con diez o menos de tales parejas. Por ejemplo, la séptima línea debe lucir así:

```
(61,=) (62,>) (63,?) (64,@) (65,A) (66,B) (67,C) (68,D) (69,E) (70,F)
```

y que puedes leer cómo el código ASCII de 'A' es 65, el de 'B' es 66 y así sucesivamente.

[10] ASCII son las siglas de *American Standard Code for Information Interchange*, aunque se utiliza para referirnos al conjunto de codificación que predominó hasta hace un par de años. Hoy día Unicode es el estándar de facto para codificación.

Lo más interesante de esta función es el uso de la función dotimes que, como su nombre indica, hace algo un cierto número de veces. Tiene la siguiente sintaxis:

```
(dotimes (VAR COUNT [RESULT]) BODY...)
```

Evalúa BODY ligando VAR a un entero sucesivo desde 0 (cero) inclusive, hasta COUNT (no incluido). Después evalúa RESULT, para obtener el valor de regreso, o regresa nil por omisión.

Ahora ya debe ser obvio cómo funciona la función show-ascii: primero inicializa el contador lstinline|counter| en 0; utiliza i como índice para recorrer todos los posibles valores ASCII desde 0 hasta 255 (recuerda que el 256 no está incluido) y va imprimiendo cada valor numérico y su correspondiente carácter. Esto lo hace la función format. Lo que sigue, sirve para agregar una nueva línea cada que hemos impreso diez parejas y no tiene mayor complicación.

9.7.2. ¿Cómo numerar una región de líneas?

Algo que puede ser muy útil, aunque sea poco común, es numerar una lista de objetos, digamos la lista de tus libros que casualmente tienes escrita en un archivo. Con lo que sabes de edición lo podrías hacer más o menos así: visitas el archivo en un buffer de Emacs, te mueves al primer carácter de la primera línea y tecleas 1.– ; luego te mueves a la siguiente línea, regresas a la primera posición en la línea y tecleas 2.– ; y así sucesivamente.

No suena muy difícil. Con un poco de práctica podrías llegar a la primera posición de la siguiente línea con teclear sólo dos combinaciones de teclas y agregar un número (uno o más dígitos), un punto, un guión y un espacio; son sólo unos cuantos carácteres más. Y en efecto, es muy sencillo. ¡Desgraciadamente tú tienes 8272 libros en tu lista! Además, como eres muy escrupuloso quieres que todos los títulos de tus libros comiencen en la misma columna, o lo que es lo mismo quieres que la numeración agregue exactamente el mismo número de carácteres a cada línea. No hay problema, aquí esta una función en elisp que lo hace:

```
1  (defun numera (beg end)
2    "Numera las lineas en la región."
3    (interactive "r")
4    (let* ((lines (count-lines beg end))
5           (padding (length (int-to-string lines))))
6      (goto-char beg)
7      (dotimes (line lines)
8        (insert (format
9                 (concat "%" (format "%cd" padding) "d.- ")
10                (1+ line)))
11       (forward-line 1))))
```

Hay pocas cosas nuevas en esta función. Por ejemplo, la función count-lines (línea 4) que regresa el número de líneas que hay en la región; la función int-to-string (línea 5) que convierte un entero a una cadena. Todo lo demás debe ser medianamente obvio, salvo quizá por el detalle de la variable llamada padding (línea 5), que la usamos para saber de cuántas posiciones tiene que ser la cadena que vamos a agregar.

El razonamiento que usamos fue el siguiente: si hay seis líneas en la región, entonces el grueso de la cadena que tenemos que agregar en cada línea es: un carácter (para el dígito de la línea), un carácter para el punto, uno para el guión y un espacio, en total cuatro. Pero si hay 28 líneas, entonces necesitamos un carácter más (los primeros nueve serán de un dígito y tendrán un espacio a la izquierda del dígito) y los siguientes ya son de dos, y así sucesivamente. Pues bien, padding – línea 9 – es justamente el número de carácteres que debe ocupar el número de línea, y format entiende ese parámetro y agrega los espacios en blanco necesarios a la izquierda del número.

9.7.3. Indentar es un placer, borrar espacios en blanco un martirio

Una de los aspectos más interesantes de Emacs es que *sabe* cómo indentar el texto de mis programas. Sin embargo, como casi cualquier otra cosa en esta vida, no es perfecto. Por ejemplo, digamos que al final de una línea pongo un carácter que abre un nuevo bloque (sin importar el lenguaje de programación). Emacs detecta el carácter e inserta una nueva línea y luego, digamos otros siete espacios en blanco para que comiences a poner el cuerpo del bloque. En la mayoría de los casos esto es una bendición. Sin embargo, supongamos que tecleé un carácter de más en la línea anterior (cerca del final de la línea), ¿cómo regreso? Lo más probable es que intente usar BackSpace para borrar el carácter y continuar.

Lo que en realidad sucede es que tienes que teclear ocho o diez veces BackSpace, en lugar de sólo una o dos, porque primero borras todos los espacios blancos que agregó Emacs. Claro, te preguntarás por qué no utilizar las funciones de posicionamiento de cursor y luego borramos y listo. Pues por muchas razones, la primera de ellas que usualmente uno sabe lo que tecleó mal casi instantáneamente. Tus dedos viajan en el teclado más rápido de lo que lees, y mientras detectas el error uno o dos carácteres más ya fueron impresos; lo natural, o instintivo si así lo prefieres, es inmediatamente teclear dos o tres veces (dependiendo de qué tan lejos esté el error) la tecla BackSpace y continuar editando.

Algunos modos ya incluyen variables que modifican el comportamiento de la función que se ejecuta cuando presionamos BackSpace, pero algunos no. Aquí está una versión de una función que borra inteligentemente todos los espacios en blanco hasta llegar a un carácter no blanco y después el código para instalarla en Emacs:

```
1  (defun my-electric-delete (arg)
2    "Borra el carázcter precedente o blancos -- espacios,
3  tabuladores, y nuevas lineas--. Si 'my-hungry-delete-key' no
4  es nil, entonces todos los blancos que preceden la posición
5  actual son borrados."
6    (interactive "P")
7    (if (or (not my-hungry-delete-key)
8            arg)
9        (funcall my-delete-function (prefix-numeric-value arg))
10     (let ((here (point)))
11       (skip-chars-backward " \t\n")
12       (if (/= (point) here)
13           (delete-region (point) here)
14         (funcall my-delete-function 1)))))
```

Primero, nota como el if (linea 7) nos sirve para discernir los casos que maneja la función. Si la variable my-hungry-delete-key, que no aparece en el cuerpo de la función, es nil o si le pasamos un argumento numérico a la función, entonces se ejecuta:

```
8    (funcall my-delete-function (prefix-numeric-value arg))
```

En esta línea hay dos funciones nuevas, pero ambas son triviales: prefix-numeric-value que regresa el valor numérico del argumento pasado a la función y funcall que evalúa su primer argumento como si fuera una función, pasándole el resto de los argumentos. Nota que no es equivalente a:

```
(my-delete-function (prefix-numeric-value arg))
```

porque esto trataría de buscar una función llamada my-delete-function, pero tal función no existe y de existir probablemente no sería lo que nosotros buscamos. Es una variable, cuyo valor es, así es adivinaste: el nombre de una función. Hablemos un poco sobre la razón de hacerlo así; porque si lo piensas dos veces parece una complicación innecesaria; ¿por qué no ponemos ahí el nombre de la función que es el valor de my-delete-function? Porque hay muchas funciones similares para borrar carácteres y quieres darle la oportunidad a los usuarios de utilizar la que mejor les acomode.

Ahora, ¿qué pasa si la condición del if es falsa? Se ejecuta la siguiente parte (lineas 10–14), que sí parece una función como las que hemos visto. Primero guardamos el valor actual del punto en una variable llamada here (linea 10), nos saltamos todos los blancos (hacia atrás) y ahora vemos si en realidad nos movimos o no. Si nos movimos, entonces borramos la región; si no nos movimos (quiere decir que estamos en una literal) entonces sólo borramos un carácter. Nuevamente borramos con la función para borrar que el usuario quiera.

Por ejemplo, para programar en elisp, puedes poner esto en tu ∼/.emacs:

```
(add-hook 'emacs-lisp-mode-hook
  (lambda ()
    (define-key emacs-lisp-mode-map "\C-m" 'newline-and-indent)
    (define-key emacs-lisp-mode-map '(backspace) 'my-electric-delete)
    (setq my-delete-function 'backward-delete-char-untabify
          my-hungry-delete-key t)))
```

que básicamente le dice a Emacs que cada vez que tecleas Enter, debe pasar a la siguiente línea y luego indentar adecuadamente. También le indica que BackSpace debe ejecutar la función que recién analizamos; por último le doy nombre a las dos variables que se usan dentro del cuerpo de la función.

9.7.4. Un ejercicio totalmente inútil, o eso parece

¿Cuántas veces has tenido la necesidad de partir una oración en una lista de las palabras que constituyen la oración? ¿Nunca? En realidad no es común, al menos no en elisp. Sin embargo, es un buen ejemplo para mostrar el manejo de cadenas y listas en elisp.

La función que verás usa dos funciones auxiliares. A decir, las dos funciones auxiliares son las que hacen el trabajo y la función principal sólo "oculta" detalles que el usuario de la función no tiene por qué conocer. Esto es una práctica usual en programación: sólo pide al usuario los parámetros que el usuario conoce.

```
1 (defun string-tokenizer (char str)
2   "Regresa una lista que contiene todos los elementos
3 en la cadena STR que están separados por CHAR."
4   (string-tokenizer-aux char str nil))
5
6 (defun string-tokenizer-aux (char str res)
7   "Regresa una lista, RES, que contiene todos los elementos
8 en la cadena STR que están separados por CHAR."
9   (let* ((index (index-of char str 0)))
10    (if (null index)
11        (if (string-equal str "")
12            (reverse res)
13          (reverse (cons str res)))
14      (string-tokenizer-aux char (substring str (1+ index))
15                            (cons (substring str 0 index)
16                                  res)))))
17
18 (defun index-of (char str count)
19   "Regresa el índice de la primera aparición de CHAR
20  en la cadena STR empezando en COUNT."
21   (cond ((string-equal str "") nil)
22         ((char-equal (aref str 0) char) count)
23         (t (index-of char (substring str 1) (1+ count)))))
```

string-tokenizer es la función interfaz y recibe dos parámetros, la cadena que vamos a seccionar y el carácter que separa las secciones. Por la pregunta con la que iniciamos la sección, podría pensarse que siempre usaríamos un espacio en blanco como separador. Finalmente las palabras en una oración casi siempre están separadas por un espacio (ignorando muchas reglas gramaticales, claro, porque los signos de puntuación y más de un espacio también delimitan palabras). Pero no lo hicimos así, porque esta función es más general y podemos fácilmente hacer una especialización de string-tokenizer que sirva para separar palabras en una oración.

Bien. Nota que esta función no hace más que llamar a string-tokenizer-aux, pero con tres parámetros. Los primeros dos son los que pasó el usuario (una cadena y un carácter separador) y un tercero, nil o la lista vacía. La función auxiliar usará este último parámetro para ir almacenando las secciones (o sub-cadenas).

Veamos entonces cómo funciona string-tokenizer-aux. Primero calcula el índice dentro de la cadena STR en el cuál está la primera presencia del carácter separador – en realidad el cálculo lo hace la segunda función auxiliar, index-of, pero por el momento vas a tener que creerme que sí funciona, es decir, que sí regresa un entero que representa la posición (en la cadena) de la primera presencia del carácter separador – .

A continuación viene una serie de pruebas; primero preguntamos si el índice es nil (linea 10); si lo es, entonces preguntamos si la cadena es la cadena vacía en la linea 11 (i.e. si ya consumimos toda la entrada). Si la respuesta es afirmativa, regresamos el resultado en reversa (más adelante explicaré por qué en reversa). Si la cadena aún no es la cadena vacía, quiere decir que el contenido completo de la cadena debe ser la última parte del resultado (ya no se puede separar más); asi que lo agregamos al resultado y lo regresamos.

Como la función string-tokenizer-aux es una función recursiva, tiene dos partes importantes. La primera, la cláusula de escape, que es la parte que acabamos de revisar en el párrafo anterior. Ahora, como buen ejemplo de función recursiva, también tiene una parte donde se llama a sí misma. Veamos cómo llegamos ahí. Si el índice no es nil (la primera pregunta en la linea 10), entonces sabemos que index-of regresó un número, un índice en la cadena, por lo que sabemos hasta dónde hay que cortar y eso hacemos: cortamos desde el inicio de la cadena hasta el índice (sin incluir al carácter separador) y agregamos esto al inicio de la lista resultado. Ahora tenemos que analizar el resto de la cadena y por eso hacemos una llamada recursiva a la función misma, pero ahora usando como parámetros: (1) el mismo carácter separador; (2) el resto de la cadena (después del primer carácter separador y sin incluirlo) y (3) con la nueva lista resultado que ya tiene en su primera posición la cadena recién cortada.

Esto se repite tantas veces como carácteres separadores contenga la cadena inicial, hasta que fnalmente termina y regresa la lista. Por ejemplo, supón que hacemos la siguiente llamada:

```
(string-tokenizer ? "es una bonita cadena")
```

La notación ? (signo de interrogación y espacio) le dice a Emacs que le estamos pasando un carácter, en este caso un espacio blanco. Lo que regresará es una lista de la forma ("es" "una" "bonita" "cadena" Nota cómo ninguna de las sub-cadenas (palabras) contiene al carácter separador.

La función auxiliar index-of es también una función recursiva. Tiene dos casos base (o cláusulas de escape): la primera se activa cuando se agota la cadena (que se va recortando en un carácter con cada llamada a sí misma); la segunda se activa cuando el carácter que estábamos buscando es encontrado. Su tercer parámetro es un contador, que si le pasamos 0 (cero) nos regresará el índice en

la cadena donde se encuentra el carácter separador por primera vez. Los siguiente ejemplos deben aclarar un poco la situación:

```
(index-of ? "es una bonita cadena" 0)
2
(index-of ? "una bonita cadena" 0)
3
```

que si cuentas desde 0 son los índices del primer espacio en blanco en ambos casos. Para redondear la idea de la recursividad, estos ejemplos siguen el comportamiento de las llamadas recursivas de la función string-tokenizer-aux:

```
(string-tokenizer-aux ? "es una bonita cadena" ())
(string-tokenizer-aux ? "una bonita cadena" ("es"))
(string-tokenizer-aux ? "bonita cadena" ("una" "es"))
(string-tokenizer-aux ? "cadena" ("bonita" "una" "es"))
(string-tokenizer-aux ? "" ("cadena" "bonita" "una" "es"))
(reverse ("cadena" "bonita" "una" "es"))
("es" "una" "bonita" "cadena")
```

Esto también debe aclarar por qué tenemos que regresar el resultado en reversa: porque fuimos metiendo las sub-cadenas al inicio de la lista. Si las fuéramos metiendo al final no habría necesidad, pero luciría más compleja la función.

Por último, decimos que las funciones index-of y string-tokenizer-aux están ambas en forma recursiva de cola, primero porque son recursivas y segundo porque lo *último* que hacen es llamarse a sí mismas. Una función recursiva puede hacer muchas llamadas a sí misma, pero está en forma recursiva de cola si y sólo si el que llama regresa inmediatamente después. Por ejemplo, la siguiente función calcula el n-ésimo término en la serie de Fibonacci:

```
(defun fib (n)
  (if (< n 2)
      1
    (+ (fib (- n 1)) (fib (- n 2)))))
```

pero no está en forma recursiva de cola, porque cuando terminan las llamadas recursivas no regresa la función ya que falta realizar las sumas.

9.7.5. Resumen

Aquí está un resumen de las funciones que acabamos de revisar:

dotimes Tiene la siguiente sintaxis:

```
(dotimes (VAR COUNT [RESULT]) BODY...)
```

Evalúa BODY ligando VAR a un entero sucesivo desde 0 inclusive, hasta COUNT (excluyéndolo). Después evalúa RESULT para obtener el valor de regreso, o regresa nil por omisión.

count-lines Tiene la siguiente sintaxis:

```
(count-lines START END &optional IGNORE-INVISIBLE)
```

Regresa el número de líneas entre START y END. Usualmente esto equivale al número de carácteres de nueva línea entre esas dos posiciones, pero puede ser una más si el mayor de los dos números no está al inicio de una línea.
Si el valor del parámetro opcional es distinto de nil, entonces las líneas ocultas (o colapsadas) con selective-display no se consideran para la cuenta.

int-to-string Recibe un número como parámetro y lo convierte a una cadena en notación decimal. Utiliza un signo negativo si el número es negativo; algo que no es obvio del nombre de la función es que puede recibir tanto enteros como números de punto flotante.

funcall Evalúa el primer argumento como si fuera una función, pasándole el resto de los argumentos como sus parámetros. Por tanto, `( funcall 'cons 'x 'y)` regresa `(x . y)`.

9.8. Para saber más

Emacs-Lisp es un lenguaje muy poderoso; sin embargo, no es un lenguaje utilizado para mucho más que modificar y extender Emacs. Es decir, su uso como lenguaje de producción no es muy amplio, por lo que muy poca gente invierte tiempo programando profesionalmente en elisp. Es importante que sepas hacer cosas relativamente sencillas, como las que vimos aquí y algunas incluso un poco más complicadas, porque de esa manera podrás mejorar tu productividad.

Por otro lado, el número de usuarios de Emacs es tan grande, hablando a nivel mundial, que es difícil que quieras hacer algo que no se le haya antojado a alguien más alguna vez y, más aún, es muy probable que esa *cosa* ya esté implementada; utiliza Google o visita alguno de estos sitios:

- http://www.gnu.org
- http://www.xemacs.org
- USENET o Google groups:
  - gnu.emacs.help
  - comp.emacs
  - comp.emacs.xemacs
  - gnu.emacs.gnus
  - gnu.emacs.sources

Apéndices

Distribuciones de Linux | A

Existen varias formas de instalar Linux en tu computadora. A lo largo de este libro se utilizó Ubuntu, pero hay muchas más opciones. Según www.distrowatch.com, en este momento existen mas de 500 versiones distintas de Linux. Existen algunas de ellas que tienen soporte por parte de grandes empresas como Novell, Oracle, Red Hat o IBM. Además, para empezar a usar Linux no necesitas instalarlo de forma permanente en tu computadora, ya que muchas distribuciones tienen lo que se conoce como un *live cd*, que es un disco que te permite utilizar Linux desde un cd o dvd sin tener que copiarlo a tu disco duro. De esa forma puedes decidir si esa distribución de Linux ofrece lo que tú necesitas sin tener que eliminar el sistema operativo que utilizas actualmente. A lo largo de este apéndice te presentaremos algunas de las distribuciones más comunes de Linux, en dónde puedes obtenerlas y cuáles son sus puntos clave.

A.1. Ubuntu

A.1.1. Filosofía

Desde 2004 Ubuntu ha experimentado un incremento enorme en su número de usuarios, siendo actualmente la distribución más popular. Como es una distribución derivada de Debian (el cual revisaremos en A.2), cuenta con toda la robustez de esa distribución. El principal problema de Debian es los largos periodos de tiempo que pueden pasar entre la liberación de nuevas versiones. Para evitar eso, los desarrolladores de Ubuntu se han comprometido a liberar una nueva versión

cada seis meses. Además los principales objetivos de Ubuntu son:

- Mantener por siempre su gratuidad, y que nunca haya un costo adicional para obtener una "versión profesional".
- Hacer que Ubuntu pueda ser usado por el mayor número de personas.
- Estar siempre comprometido con el software libre.

A pesar de tener a su disposición todos los paquetes de Debian, en Ubuntu la instalación por omisión incluye un número reducido de paquetes, lo cual permite que pueda distribuirse en un solo CD, dejando la instalación de otros paquetes como una decisión posterior. Puedes obtener una copia en http://www.ubuntu.com/

A.1.2. Manejo de software

El manejo de paquetes dentro de Ubuntu se hace utilizando archivos .deb, los cuales pueden ser instalados con la herramienta dpkg. Esta herramienta es la encargada de desempacar los archivos, copiarlos a su destino y ejecutar las operaciones para que el nuevo software opere adecuadamente. Dado que dpkg es una herramienta poderosa pero a la vez poco amigable, se han creado varias interfaces para hacer el trabajo de instalación más sencillo. De dichas interfaces, una de las más comunes es apt. Veamos como se instala un paquete haciendo uso de esta herramienta.

Lo primero que necesitamos es saber cuál es el nombre del paquete que deseamos instalar; para ello haremos uso de la herramienta apt-cache. Mediante ésta podremos consultar la lista de paquetes disponibles y ver cuál de ellos es el que nosotros deseamos. Estas listas se construyen a partir de los repositorios que se encuentran en el archivo /etc/apt/sources.list. Dicho archivo contiene líneas como la siguiente:

```
deb http://mx.archive.ubuntu.com/ubuntu/ main universe
```

Esa línea indica lo siguiente:

- La palabra deb indica que se van a obtener archivos de binarios (también puede aparecer deb-src, para indicar que se van a obtener archivos de código fuente).
- Le sigue el URL del repositorio que indica de dónde se van a obtener los paquetes.
- La siguiente palabra indica qué distribución se utiliza (en este caso *stable*), y qué secciones están disponibles[1] (en este caso main).

Puedes agregar tantas líneas como necesites, para así agregar más repositorios que te permitan tener acceso a más paquetes. La mayor parte del tiempo no necesitarás más que los repositorios oficiales. En caso de que modifiques tus repositorios necesitarás indicarle a apt que reconstruya la lista de paquetes. Para poder hacerlo, ejecuta como root la instrucción:

```
% apt-get update
```

Una vez hecho eso, puedes buscar el paquete que deseas, digamos emacs. Entonces es necesario ejecutar la instrucción:

```
% apt-cache search emacs
```

[1] Entre las distintas secciones están: main universe multiverse restricted

De esa forma le indicamos a apt que debe buscar la lista de paquetes y regresarnos todos los que contengan la palabra emacs en su nombre o su descripción. Esto regresará algo como:

```
dictionary-el - dictionary client for Emacs
drscheme - PLT Scheme Programming Environment
emacs-extra - emacs configuration
emacs-snapshot-gtk - The GNU Emacs editor (with GTK+ 2.x support)
emms - The Emacs MultiMedia System
emacs21 - The GNU Emacs editor
emacs21-bin-common - The GNU Emacs editor's dependent files
emacs21-common - The GNU Emacs editor's
```

Si quieres ver más detalles acerca de un paquete, entonces puedes ejecutar:

```
% apt-cache search -f emacs21-common
```

Con lo cual obtienes un listado como:

```
Package: emacs21-common
Priority: optional
Section: editors
Installed-Size: 35948
Architecture: all
Source: emacs21
Version: 21.4a-6ubuntu2
Replaces: emacs21 (< 21.2-4)
Depends: emacsen-common (>= 1.4.10), dpkg (>= 1.9.0)
Suggests: emacs21-el
Conflicts: emacs21-el (< 21.4a-6), w3-el
Filename: pool/main/e/emacs21/emacs21-common_21.4a-6_all.deb
Size: 10936074
MD5sum: b4a981d12f6a39f43c896ddf924129fd
SHA1: 33f667d8fbd5e18570d405c5c54d9e22643c9d6f
Description: The GNU Emacs editor's shared, architecture
             independent infrastructure
 GNU Emacs is the extensible self-documenting text editor.
 This package contains the architecture independent infrastructure
  that's shared by emacs21 and emacs21-nox.
```

Esa instrucción puedes ejecutarla con más de un paquete para que encuentres cuál es el que deseas instalar. Una vez hecho eso, será necesario ejecutar el comando de instalación de apt:

```
% sudo apt-get install emacs21-common
```

Con lo cual apt te informará qué paquete se va a descargar y cuánto espacio utilizará. En caso de que existan dependencias no resueltas, te informará cuáles son los paquetes que debes instalar para satisfacerlas, y si así lo deseas, apt se encargará de obtenerlas por ti.

A.2. Debian

A.2.1. Filosofía

Debian es un sistema operativo libre, dinámico y gratis. Es distribuido por el Proyecto Debian. Su objetivo principal es distribuir un conjunto de software que sea capaz de satisfacer la mayoría de las necesidades de los usuarios de una computadora. Para lograr eso, Debian actualmente tiene 18,733 paquetes disponibles. Debian mantiene sus distribuciones en tres estados:

stable (estable). Contiene los paquetes oficiales; es la distribución lista para usarse y todo está garantizado que funcionará.

testing (de prueba). Contiene paquetes que aún no han sido aceptados en la distribución estable, pero que en algún momento lo estarán.

unstable (inestable). Contiene paquetes en desarrollo; probablemente habrá paquetes que no funcionarán del todo bien, pero también tiene los paquetes más nuevos.

Conforme va pasando el tiempo, los paquetes pueden pasar de una distribución inestable a una de prueba y de ahí, después de una revisión minuciosa, finalmente llegan a estable. Muchas personas consideran a Debian como la distribución más robusta de Linux[2], siendo esto una consecuencia de la enorme cantidad de paquetes que tiene disponibles en sus repositorios, todas las arquitecturas que soporta y la garantía de que en su versión estable todo está probado. Por desgracia, esto puede ocasionar que la versión estable sea obsoleta, pues revisar todos esos paquetes para garantizar que todo es seguro lleva mucho tiempo. Esto puede no ser un problema si utilizas la versión de prueba o inestable. Puedes descargar Debian desde http://www.debian.org/.

A.2.2. Manejo de software

El manejo de paquetes de Debian, al igual que Ubuntu, es mediante el manejador de paquetes dpkg y su interfaz apt. Para revisar cómo utilizar dichas herramientas, consulta la sección A.1.

A.3. Fedora

A.3.1. Filosofía

Fedora es una colección de proyectos patrocinados por Red Hat, y desarrollados en contacto con la comunidad de software libre. Originalmente Red Hat tenía una sola distribución homónima a la compañía, pero en el 2003 ésta desapareció. Red Hat dividió sus esfuerzos en dos di-

[2] dando esto como resultado varias "guerras santas"

recciones: el proyecto Fedora mantendría lo que antes era la distribución del sistema operativo y Red Hat Enterprise se encargaría de atender a los clientes empresariales. El propósito del proyecto Fedora es y ha sido apoyar el rápido progreso del software libre y de su contenido. Esto incluye tener procesos abiertos al público, transparencia y rápida innovación. Puedes obtenerlo en http://fedoraproject.org/.

A.3.2. Manejo de software

El manejo de paquetes de Fedora es mediante el uso de archivos .rpm, los cuales se instalan haciendo uso de la herramienta rpm[3]. Para resolver las dependencias que puede tener un paquete, e instalarlas automáticamente, se hace uso de la herramienta yum[4]. Si se desea instalar emacs en el sistema, lo primero que se debe hacer es buscar cuál es el paquete que se desea instalar. Para eso se utiliza la instrucción:

```
% yum search emacs
```

Con lo cual se obtiene un listado de los paquetes que contienen la palabra emacs, y que se puede observar a continuación.

```
emacs.i386                    21.4-5                    installed
Matched from:
emacs
The GNU Emacs text editor.
Emacs is a powerful, customizable, self-documenting,
modeless text editor. Emacs contains special code editing
features, a scripting language (elisp), and the capability
to read mail, news, and more without leaving the editor.
http://www.gnu.org/software/emacs/


readline.i386                 5.0-3                     installed
Matched from:
The Readline library provides a set of functions that allow
users to edit command lines. Both Emacs and vi editing modes
are available. The Readline library includes additional
functions for maintaining a list of previously-entered command
lines for recalling or editing those lines, and for performing
csh-like history expansion on previous commands.
```

Una vez que sabes cual es el nombre del paquete que quieres instalar, debes ejecutar como root la instrucción:

```
% yum install emacs.i386
```

[3] Redhat Package Manager
[4] Yellow dog Update Manager

Con lo cual yum te mostrara cuales son los paquetes que va a instalar, sus dependencias y cuanto espacio van a ocupar.

Compilando e instalando a pie

B

B.1. Aplicaciones no disponibles

Hay ocasiones en las que quieres instalar una aplicación, pero no existe un paquete para el sistema de instalación de tu distribución. Algunas veces, alguien más se encarga de distribuirlos y puedes o no conseguir un .deb o un .rpm en su página. En caso de que eso no sea posible, será necesario compilar el paquete e instalarlo a pie. Esto puede sonar más complicado de lo que en realidad es, pero para ilustrar el proceso veamos un ejemplo. Vamos a instalar un cliente de MSN para Linux que se conoce como *amsn*. Lo primero que necesitas es conseguir el código fuente de la aplicación, que en este caso lo puedes bajar de http://sourceforge.net/projects/amsn/. El archivo que vamos a bajar es amsn-0.96.tar.bz2, el número corresponde a la versión que bajamos, por lo que es probable que cuando tú lo consigas sea algo distinto. El siguiente paso es descomprimir el archivo; en este caso, a partir de la extensión, sabemos que es un archivo empacado con tar y luego comprimido con bzip2. Para desempacarlo puedes usar el comando

tar xvfj amsn-0.96.tar.bz2,

o bien hacerlo desde Konqueror. Esto genera el directorio amsn-0.96 En el encontrarás algunos archivos que suelen ser estándar en las distribuciones de código fuente en UNIX. Entre estos se encuentran, generalmente, los siguientes archivos:

- README
- INSTALL
- configure

En el archivo README viene una breve explicación de qué es la aplicación, cómo funciona y a veces instrucciones para instalar. En caso de que las instrucciones para instalar no aparezcan en README, es porque están en INSTALL. La información más importante de este archivo son las dependencias del programa que quieres compilar y las opciones que le puedes pasar al compilarlo. El resto del procedimiento es igual para la mayoría de los programas, y usualmente es:

1. Configurar los parámetros de compilación.
2. Compilar el programa.
3. Instalar el programa.

Configurando la compilación

Veamos cómo se configuran los parámetros de compilación. Para hacer eso utilizamos herramientas de UNIX que se llaman automake y configure. El desarrollador que preparó los fuentes para instalar se encargó de utilizar un guión de instalación con automake para que nosotros podamos compilar con configure. De esa forma configure se encarga de buscar en dónde tenemos nosotros instaladas las dependencias del programa que queremos compilar, y establece en dónde queremos instalar nuestro programa una vez compilado, incluyendo binarios y documentación. Así que realicemos este paso para *amsn*. Es necesario que el archivo configure tenga permisos de ejecución; en caso de que no sea así puedes otorgárselos con chmod +x configure; una vez hecho eso ejecútalo con ./configure. Verás que empieza a revisar si tienes instaladas todas las dependencias que necesitas; si no encuentra algo en ese momento se detendrá y te indicará qué es lo que falta en tu sistema. Supongamos que en este momento configure indica que no puede encontrar el compilador de C, gcc; entonces verás algo como:

```
checking for gcc... no
checking for cc... no
checking for cc... no
checking for cl... no
configure: error: no acceptable C compiler found in $PATH
```

Será necesario instalar este compilador. En este momento, si corres con suerte, bastará con utilizar tu manejador de paquetes mientras que en el peor de los casos tendrás que compilar también ese paquete. Para instalar gcc y de paso g++[1] en Ubuntu basta con ejecutar sudo apt-get install gcc g++. así que si una vez heco esto ultimo vuelves a ejecutar config, deberá seguir con su ejecución normal. En caso de que te hagan falta bibliotecas para seguir compilando también será necesario que las instales. Supongamos que no tienes instalado Tk, el cual es necesario para compilar amsn; cuando configure se detenga y nos indique que no encuentra la biblioteca necesaria, recurrimos a apt para instalarla. En este caso es necesario instalar el paquete de desarrollo, pues queremos compilar código fuente. Los paquetes de desarrollo en Linux tienen la terminación *dev*. Ejecutamos entonces sudo apt-get install tk8.4-dev, y una vez más ./configure. Ahora amsn debería compilar sin problemas[2]. En caso de que quieras modificar algún parámetro de la instalación de amsn, puedes

[1] g++ es el compilador de C++.

[2] Dependiendo de qué tengas instalado en tu máquina, es posible que necesites instalar más paquetes, pero el procedimiento es igual al que acabamos de describir

ejecutra ./ configure −−help, lo cual te mostrará las opciones disponibles.

Compilando

Cuando configure termina exitosamente su ejecución, puedes compilar el programa. Al terminar, configure crea un archivo de nombre Makefile en el archivo en donde se encuentra el código fuente. Dentro de ese archivo se encuentran las opciones y rutas que se prepararon en el sección B.1. La herramienta que lee el archivo Makefile, lleva el mismo nombre, *makefile*, y se invoca con el comando make. En caso de que no esté instalada, nuevamente recurrimos a apt, sudo apt−get install make. Ahora, colocándonos en el directorio en donde se encuentra el archivo Makefile, tecleamos make y vemos cómo se compila todo[3].

Instalando

Ahora viene la última parte. Después de haber compilado el programa es necesario instalarlo en algún lugar que facilite su acceso. De esa forma cualquier usuario del sistema puede usar el programa que compilaste. Para llevar a cabo la instalación una vez más vas a hacer uso de make, pero esta vez es necesario hacerlo con permisos de super usuario, sudo make install . Verás como make despliega las instrucciones que está llevando a cabo.

Listo. Una vez hecho esto, si tecleas amsn desde tu consola, se iniciará el cliente de MSN. Este proceso es casi estándar dentro del mundo UNIX, por lo que puedes repetirlo para la mayoría del código fuente que quieras compilar e instalar. Si existe un paquete para la aplicación que deseas instalar dentro del manejador de paquetes de tu distribución, es mejor idea usarlo y no compilar a pie. Si compilas e instalas pierdes las ventajas del manejador, como saber fácilmente qué aplicaciones tienes instaladas y en qué versión están, o el desinstalar fácilmente[4].

[3] Dependiendo de qué estés instalando, esto puede tardar varios minutos

[4] En algunas aplicaciones puedes usar sudo make uninstall para desinstalar la aplicación

Emacs: archivos de configuración

C

A lo largo de este libro hemos revisado Emacs y muchas extensiones, subsistemas y paquetes para extender la funcionalidad básica de Emacs. Conforme avanzamos, explicamos las distintas opciones y líneas de configuración, pero las integramos aquí para tu comodidad.

C.1. ~/.emacs

```
;; Este archivo de configuración para Emacs, es un anexo del libro:
;; Manual de supervivencia en Linux

;; ****************************************
;; ************ font-lock *****************
;; ****************************************
(require 'font-lock)
(add-hook 'font-lock-mode-hook 'turn-on-fast-lock)
(global-font-lock-mode 1)

;; Color-theme:
(add-to-list 'load-path (expand-file-name "~/elisp/color-theme"))
(require 'color-theme)
(color-theme-initialize)
(color-theme-charcoal-black)
```

```
;; ****************************************
;; *********** Codificación: UTF-8 ***********
;; ****************************************
(prefer-coding-system 'utf-8)

;; AUCTeX, RefTeX y preview-emacs:
;; Si AUCTeX y preview-emacs están instalados de manera global en tu
;; sitio, no requieres las siguientes dos líneas:
(add-to-list 'load-path (expand-file-name "~/elisp/auctex"))
(add-to-list 'load-path (expand-file-name "~/elisp/auctex/preview"))
(require 'tex-site)
(require 'tex)
(require 'latex)
(require 'reftex)
(load "preview-latex.el" nil t t)
(setq TeX-auto-save t)
(setq TeX-parse-self t)
(setq-default TeX-master nil)
(setq LaTeX-section-hook
      '(LaTeX-section-heading
        LaTeX-section-title
        LaTeX-section-toc
        LaTeX-section-section
        LaTeX-section-label)
      )


;; Ispell hablando español.
(setq ispell-program-name "aspell"
      ispell-extra-args '("-W" "3")
      ispell-dictionary-alist
      (cons
       '("spanish" "[[:alpha:]]" "[^[:alpha:]]" "['.-]" nil ("-B" "-d"
          "es") nil utf-8)
       ispell-dictionary-alist)
      )
(set-default 'ispell-local-dictionary "spanish")

;; BBDB
;; Si BBDB está instalado en un sitio estándar de Emacs, no requieres
   la primera línea:
(add-to-list 'load-path (expand-file-name "~/elisp/bbdb/lisp"))
(require 'bbdb)
(bbdb-initialize 'vm)

;; w3m
(require 'w3m-load)

;; Dictionary (dict client)
```

```
(load "dictionary-init")
(global-set-key [(control c) ?s] 'dictionary-search)
(global-set-key [(control c) ?m] 'dictionary-match-words)

;; TRAMP
(require 'tramp)
(setq tramp-default-method "scp")

;; ERC
(require 'erc)
(require 'erc-spelling)

;; JDE y amigos:
(load-file "~/elisp/cedet/common/cedet.el")
(semantic-load-enable-code-helpers)
(add-to-list 'load-path (expand-file-name "~/elisp/jde/lisp"))
(add-to-list 'load-path (expand-file-name "~/elisp/elib"))
(require 'jde)
(setq jde-enable-abbrev-mode t
      jde-electric-return-p t
      jde-build-function '(jde-ant-build)
      jde-debugger '("JDEbug")
      tempo-interactive t
      )

;; Subversion Dependiendo de tu configuración, puede ser que necesites
;; la primera línea para enviar comentarios en UTF-8 al repositorio:
(setenv "LCCTYPEenUS.utf-8")
(require 'psvn)
```

C.2. ~/.vm

```
;; -*- emacs-lisp -*-
(setq vm-spool-files
      '(("~/INBOX"
         "imap-ssl:galadriel.fciencias.unam.mx:993:inbox:login:Juan.Doe
             :*"
         "~/INBOX.CRASH"))
      )

(setq vm-imap-auto-expunge-alist
      '(
        ;; Dejamos una copia en el servidor.
        ("imap-ssl:galadriel.fciencias.unam.mx:993:inbox:login:Juan.Doe
            :*" . nil)
```

```
        )
      vm-skip-read-messages t
      vm-circular-folders   t
      vm-virtual-folder-alist
      '(
        ("Amigos"
         (("~/INBOX")
          (and (author "Elisa Viso.*") (author "Francisco Solsona.*"))
          ))
        )
      vm-summary-show-threads t
      )

(setq vm-mail-header-from "Juan Doe <Juan.Doe@ciencias.unam.mx>"
      send-mail-function 'smtpmail-send-it
      smtpmail-smtp-server "galadriel.fciencias.unam.mx"
      smtpmail-local-domain "ciencias.unam.mx"
      smtpmail-sendto-domain "ciencias.unam.mx"
      )

;; BBDB insinuation:
(bbdb-insinuate-vm)
```

C.3. ~/.emacs-w3m

```
;; -*- emacs-lisp -*-

(setq w3m-home-page "http://www.fciencias.unam.mx"
      w3m-default-display-inline-images t
      )
```

Bibliografía

[1] DIVERSOS AUTORES The tex catalogue on line.
http://www.tex.ac.uk/tex-archive/help/Catalogue/catalogue.html. Contiene muchísimos paquetes de LaTeX con su documentación, listos para ser descargados.

[2] ETTRICH, M. Kde. http://www.kde.org.

[3] HILBRICH, T. Emacs: Dictionary. http://www.myrkr.in-berlin.de/dictionary/.

[4] ITO, A. Emacs: w3m. http://w3m.sourceforge.net.

[5] KINNUCAN, P. Emacs: Jdee - java development environment for emacs.
http://jdee.sunsite.dk/.

[6] LUDLAM, E. Emacs: Cedet - collection of emacs development environment tools. http://cedet.sourceforge.net/.

[7] MITTELBACH, F., AND GOOSENS, M. *The LaTeX Companion, Second Edition*. Addison-Wesley, 2004.

[8] REICHOER, S. Emacs: psvn. http://www.xsteve.at/prg/vc_svn/.

[9] STALLMAN, R. Emacs, gcc y gdb. http://www.gnu.org. Presidente de la Free Software Foundation. Fundador GNU.

[10] TORVALDS, L. El sistema operativo linux. http://www.linux.org.

[11] TSUCHIYA, M. Emacs: emacs-w3m. http://emacs-w3m.namazu.org.

[12] WALLIN, I., AND CEDEQVIST, P. Emacs: Elib - emacs lisp library.
http://elib.sourceforge.net.

[13] WANG, P. S. *An Introduction to Unix with X and the Internet*. PWS Publishing Company, 1997.

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CIENCIAS

Manual de supervivencia en Linux
se terminó de imprimir en octubre de 2007
en los talleres de Publidisa Mexicana, S. A. de C. V.
Calz. de Chabacano 69, Col. Asturias
México 06850, D. F.

El tiro fue de 500 ejemplares

Está impreso en papel Bond de 90 gramos
En su composición se emplearon tipos Times New Roman y Courier
de 11:13.5, 12:14 y 18:20 puntos de pica.