Herramientas para sobrevivir en Ciencias de la Computación

Canek Peláez, editor
Facultad de Ciencias, UNAM

AUTORES:

Verónica Arriola Ríos
José Galaviz Casas
Emiliano Galeana Araujo
Karla García Alcántara
Luz de María Gasca Soto
Salvador Hernández López
Alejandro Hernández Mora
Canek Peláez Valdés
Luis Soto Martínez
Elisa Viso Gurovich
Víctor Zamora Gutiérrez

BASADO EN EL MANUAL ORIGINAL DE:

Editores: Elisa Viso y Francisco Solsona
Autores: Mauricio Aldazosa, José Galaviz, Karla García, Iván Hernández, Canek Peláez, Karla Ramírez, Fernanda Sánchez Puig, Francisco Solsona, Manuel Sugawara, Arturo Vázquez y Elisa Viso

Obra financiada parcialmente por el proyecto PAPIME **PE108620**
DGAPA, UNAM
2022

Índice general

Prefacio IX

1. Introducción a Unix **1**
1.1. Historia . . . 1
1.1.1. Sistemas UNIX libres . . . 2
1.1.2. El proyecto GNU . . . 3
1.2. Sistemas de tiempo compartido . . . 4
1.2.1. Los sistemas multiusuario . . . 4
1.3. Inicio de sesión . . . 5
1.4. Emacs . . . 6
1.4.1. Introducción a Emacs . . . 6
1.4.2. Emacs y tú . . . 7

2. Ambientes gráficos en Linux **9**
2.1. Ambientes de escritorio . . . 9
2.2. GNOME . . . 10
2.2.1. El escritorio GNOME . . . 10
2.2.2. Actividades . . . 12
2.3. Emacs . . . 16
2.4. Comenzando con Emacs . . . 16

3. Sistema de Archivos **17**
3.1. El sistema de archivos . . . 17
3.1.1. Rutas absolutas y relativas . . . 20
3.2. Moviéndose en el árbol del sistema de archivos . . . 21
3.2.1. Permisos de archivos . . . 24

4. La Terminal **29**
4.1. Sintaxis estándar de comandos . . . 29
4.2. Agrupando nombres de archivos . . . 31
4.3. Archivos estándar y redireccionamiento . . . 32
4.4. Entubamientos y filtros . . . 34

4.5. Citas (*quotations*) . . . 35
4.6. Procesos y Tareas . . . 35
4.7. Utilidades de la terminal . . . 37
4.8. Script para la terminal . . . 39
4.9. Variables de entorno y alias . . . 41

5. Edición **49**
5.1. La guerra de los editores de texto . . . 49
5.2. Emacs . . . 50
5.2.1. Comandos . . . 50
5.2.2. Movimiento . . . 52
5.2.3. Matando y borrando . . . 53
5.2.4. Reencarnación de texto . . . 53
5.2.5. Regiones . . . 54
5.2.6. Rectángulos . . . 54
5.2.7. Registros . . . 55
5.2.8. Archivos . . . 55
5.2.9. Buscar . . . 56
5.2.10. Reemplazar . . . 56
5.2.11. Guardar . . . 56
5.2.12. Ventanas . . . 57
5.2.13. Marcos (*frames)* . . . 57
5.2.14. Ayuda en línea . . . 57
5.2.15. Lista de buffers . . . 57
5.2.16. Anzuelos (*Hooks*), *setq* y otros . . . 58
5.2.17. Ortografía . . . 58
5.2.18. Extendiendo Emacs . . . 59

6. Control de versiones **63**
6.1. Control de versiones . . . 63
6.1.1. Sistemas centralizados . . . 64
6.1.2. Sistemas distribuidos . . . 65
6.2. Git . . . 66
6.2.1. Bifurcaciones . . . 71
6.2.2. Deshacer cambios . . . 74
6.2.3. Usos avanzados . . . 75
6.3. GitHub . . . 75

7. LaTeX **81**
7.1. Introducción . . . 81
7.2. Componentes de un texto . . . 82
7.3. Ambientes para formato . . . 83

7.3.1. Posibles errores . . . 85
7.4. Formato general de un documento . . . 87
7.4.1. Cartas . . . 90
7.5. Tipos y tamaños de letras . . . 91
7.6. Listas de enunciados o párrafos . . . 94
7.6.1. Numeraciones . . . 95
7.6.2. Listas marcadas . . . 98
7.6.3. Descripciones . . . 99
7.6.4. Listas más compactas . . . 101
7.7. Tablas . . . 102
7.7.1. Multicolumnas . . . 104
7.7.2. Separación de renglones y columnas . . . 106
7.7.3. Líneas "incompletas" verticales y horizontales . . . 110
7.7.4. Espacio entre los renglones . . . 111
7.8. Matemáticas en LaTeX . . . 112
7.8.1. Fórmulas matemáticas . . . 114
7.9. Emacs y LaTeX . . . 116
7.9.1. Anotaciones en los símbolos . . . 130
7.9.2. Arreglos de fórmulas . . . 131
7.10. Imágenes y figuras . . . 135
7.10.1. Tablas, figuras, etc. . . . 135
7.10.2. Cortando figuras . . . 142
7.10.3. Colores . . . 143
7.10.4. Dibujando objetos geométricos . . . 146
7.10.5. Líneas . . . 146
7.10.6. Acomodando objetos . . . 166
7.11. Referencias e índices . . . 171
7.11.1. Referencias bibliográficas . . . 171
7.11.2. Índices . . . 177
7.12. Emacs . . . 177
7.12.1. AUCTeX . . . 178
7.12.2. RefTeX . . . 181
7.12.3. Preview . . . 182

8. Lenguajes de marcado **185**
8.1. XML . . . 186
8.2. HTML . . . 189
8.2.1. XHTML . . . 189
8.3. CSS . . . 190
8.3.1. Javascript . . . 193
8.4. Emacs . . . 195

A. Distribuciones de Linux **197**
A.1. Fedora 197
A.1.1. Filosofía 197
A.1.2. Manejo de software 198
A.2. Ubuntu 199
A.2.1. Filosofía 199
A.2.2. Manejo de software 199
A.3. Debian 201
A.3.1. Filosofía 201
A.3.2. Manejo de software 201

B. Compilando e instalando a pie **203**
B.1. Aplicaciones no disponibles 203

C. Emacs: archivos de configuración **207**
C.1. ∼/.emacs 207

Índice de figuras

1.1. Acceso al sistema. 5
1.2. Inicio de una sesión en un sistema UNIX. 6
1.3. Equivocación al teclear la contraseña. 6

2.1. GNOME 3 10
2.2. Indicadores 11
2.3. Configuraciones 11
2.4. Calendario 12
2.5. Actividades 13
2.6. Buscando Emacs 13
2.7. Aplicaciones 14
2.8. Agregando Emacs al tablero 14
2.9. Varias aplicaciones 15
2.10. Escritorios virtuales 15

3.1. Sistema de archivos en un sistema Unix 19
3.2. Resultado de entubar ls -al con less 23
3.2. Resultado de entubar ls -al con more *Continúa de la página anterior* . . . 24

6.1. Asignamos un título mnemónico a la llave y la agregamos. 78
6.2. Personalizamos la creación del nuevo repositorio. 79
6.3. Opciones para inicializar un repositorio. 80
6.4. Vista final del contenido del archivo en línea. 80

7.1. Gráfico introducido con `includegraphics` 139
7.2. Escalando objetos con graphics. 140
7.3. Rotaciones y reflejos con `graphics` 141
7.4. Tomando pedazos de figuras 142
7.5. Ejemplo de opciones para dibujar 152
7.6. Graficación de funciones arbitrarias 161
7.7. Impresión de la retícula 162
7.8. Colocación de los ejes 162

7.9. Segunda forma de pintar la retícula 163
7.10. Divertimento con sólo trazo de líneas 164
7.11. Dibujo de una gráfica en LaTeX 169
7.12. Actividad 7.52 171
7.13. Ejemplo de LaTeX usando BibTeX (1/2) 173
7.13. Ejemplo de LaTeX usando BibTeX (2/2) 174
7.14. Salida de LaTeX al ejecutar BibTeX 174
7.15. Ejemplo de una base de datos BibTeX 175
7.16. Ejemplo usando los estilos BibTeX 176
7.17. Preview en acción 183

8.1. Página sin hoja de estilo 191
8.2. Página con hoja de estilo 192

Índice de tablas

1. Convenciones tipográficas IX
1. Convenciones tipográficas *Continúa de la página anterior* X

4.1. Variables de entorno básicas 42
4.2. Comandos de Unix importantes 42
4.2. Comandos de Unix importantes *Continúa de la página anterior* 43
4.2. Comandos de Unix importantes *Continúa de la página anterior* 44
4.2. Comandos de Unix importantes *Continúa de la página anterior* 45
4.2. Comandos de Unix importantes *Continúa de la página anterior* 46

5.1. Emacs: comandos de movimiento 52
5.2. Emacs: movimiento a lugares especiales 53
5.3. Emacs: comandos para matar y borrar 53
5.4. Emacs: comandos para manejar rectángulos 55
5.5. Emacs: comandos para manejar registros 55
5.6. Emacs: comandos para revisar ortografía 59

7.1. LaTeX: ambientes 83
7.2. LaTeX: comandos para modificar espacios 88
7.2. LaTeX: comandos para modificar espacios *Continúa de la página anterior* . . 89
7.3. Tamaños disponibles para letras 92
7.4. LaTeX: comandos para cambios de tipo de letra 93
7.5. Modificadores para la definición de columnas 103
7.6. Acentos 113
7.7. Símbolos no ingleses 113
7.8. Símbolos de puntuación y especiales 113
7.9. Símbolos griegos 117
7.10. Símbolos con flechas 118
7.11. Operadores y símbolos relacionales 118
7.11. Operadores y símbolos relacionales *Continúa de la página anterior* . . . 119
7.12. Símbolos varios 120
7.13. Nombres de funciones comunes 120

7.14. Símbolos agrandables 122
7.15. Delimitadores 124
7.16. Tipos de letras en modo matemático 125
7.16. Tipos de letras en modo matemático *Continúa de la página anterior* . . . 126
7.17. Acentos en modo matemático 130
7.18. Comandos de espacio en modo matemático 134
7.19. Posiciones posibles para los flotantes 136
7.19. Posiciones posibles para los flotantes *Continúa de la página anterior* . . . 137
7.20. Conversión de grados Farenheit a Centígrados 137
7.21. Conversión de Farenheit a Centígrados 138
7.22. Representaciones numéricas 138
7.23. Comparación de tamaños 142
7.24. Paquetes de pgf para colores y dibujos 143
7.25. Colores básicos con los que se cuenta 145
7.26. Modificadores para el trazo de líneas y figuras 151
7.27. Opciones de grosor y estilos de líneas 152
7.28. Ejemplos de figuras en TikZ 153
7.29. Líneas dobles 153
7.30. Tipos de puntas de flechas en TikZ 154
7.31. Figuras geométricas que se pueden lograr en TikZ 155
7.31. Figuras geométricas que se pueden lograr en TikZ *Continúa de la página anterior* 156
7.31. Figuras geométricas que se pueden lograr en TikZ *Continúa de la página anterior* 157
7.32. Graficación de funciones comunes en TikZ 157
7.32. Graficación de funciones comunes en TikZ *Continúa de la página anterior* . 158
7.33. Símbolos para marcar curvas que se grafican 159
7.33. Símbolos para marcar curvas que se grafican *Continúa de la página anterior* . 160
7.34. Emacs: comandos de AUCTEX 179

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 Fedora y como administrador de paquetes utilizamos yum. Por ejemplo, verás en notas al margen algo parecido a esto:

```
% yum install emacs
```

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 Emacs, 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 caracteres 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 caracteres 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.

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:

- GNOME
- KDE

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

- Xfce
- Enlightenment

2.2. GNOME

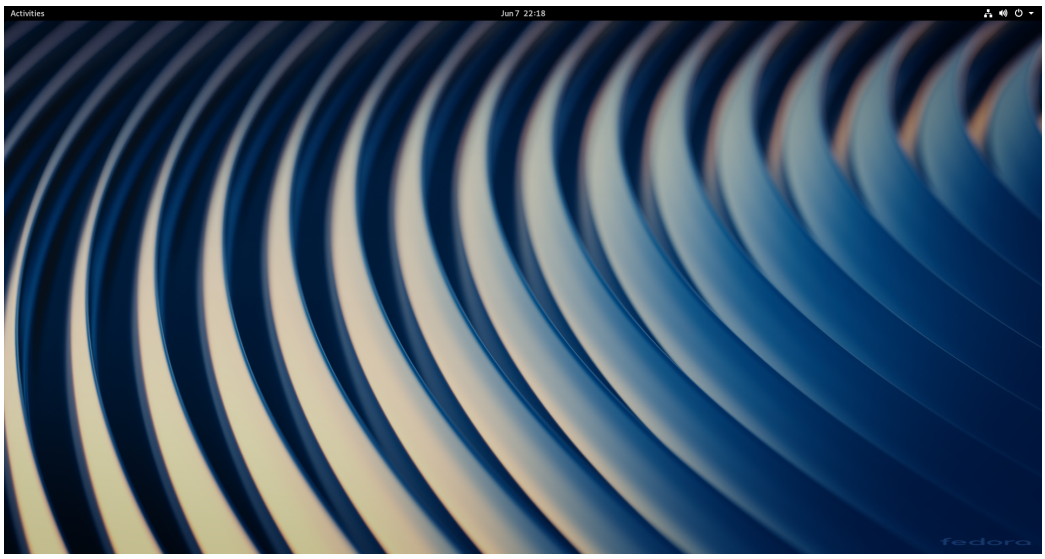
El sitio del proyecto GNOME[3], describe a la versión 3 del mismo como “una manera fácil y elegante de usar la computadora”.

La filosofía de GNOME es poder proporcionarle al usuario un escritorio fácil de usar para que pueda realizar su trabajo. En ese sentido, GNOME en general evita ofrecer muchas opciones de configuración, bajo la idea de que un ambiente de trabajo debe de funcionar sin que el usuario tenga que configurar nada: las cosas deben funcionar de manera automática.

2.2.1. El escritorio GNOME

El ambiente de trabajo de GNOME (conocido como el *shell*) consiste en únicamente un fondo de pantalla, y una barra superior donde están las actividades, el calendario y reloj, e indicadores distintos del sistema, como se muestra en la figura 2.1.

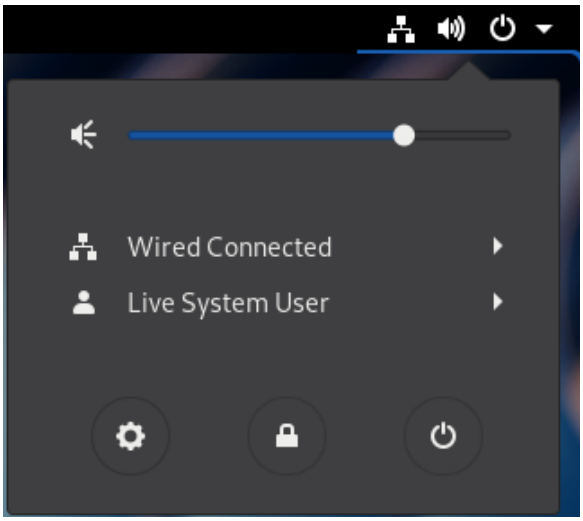
Figura 2.1 GNOME 3



[3] http://www.gnome.org/

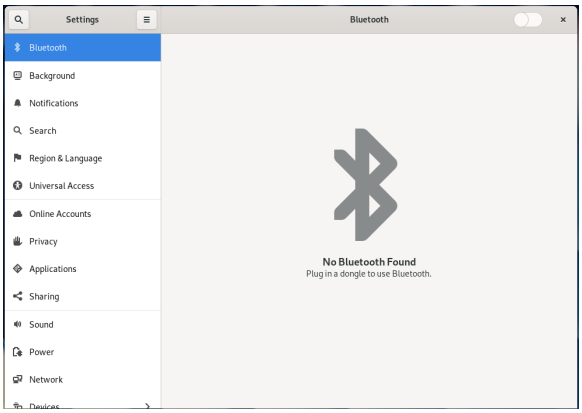
Cada una de las zonas de la barra superior cumple una función distinta: los indicadores nos permiten verificar y configurar cosas como el idioma del teclado, el volumen del sonido, el tipo de conexión de red que estamos usando, así como opciones para cerrar la sesión o apagar la computadora, como se ve en la figura 2.2.

Figura 2.2 Indicadores



El icono de abajo a la izquierda nos permite acceder a las configuraciones del sistema (figura 2.3).

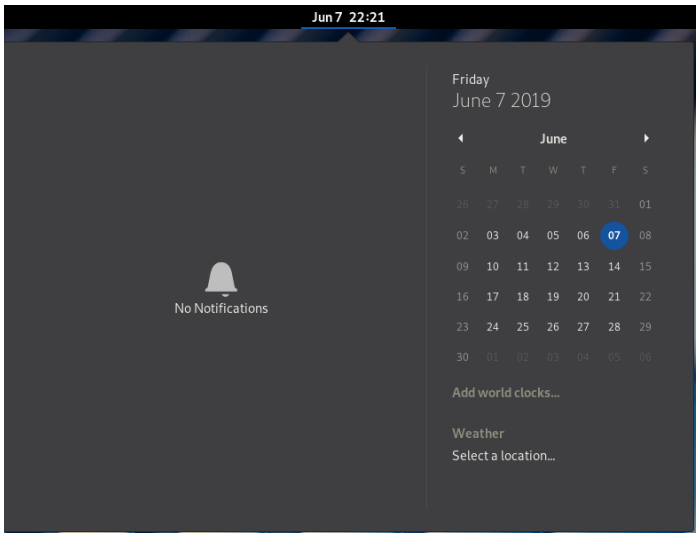
Figura 2.3 Configuraciones



El calendario y reloj nos permiten ver rápidamente la fecha y hora, así como citas o

eventos que tengamos apuntados en nuestro calendario (figura 2.4).

Figura 2.4 Calendario



Y las actividades serán lo que más usaremos, así que se describiran en la siguiente sección:

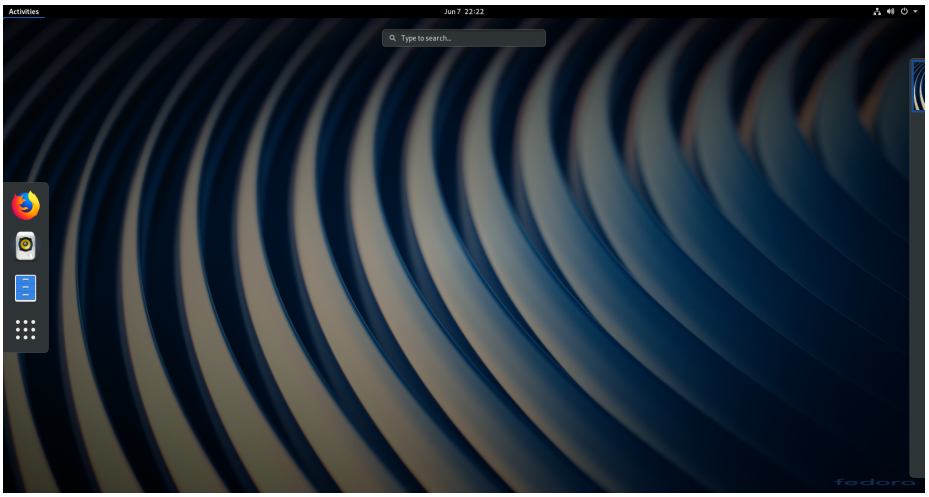
2.2.2. Actividades

Las actividades nos permiten organizar, lanzar, e interactuar con las aplicaciones con las estemos trabajando en nuestra computadora. Se puede acceder a las actividades de varias maneras: una es haciendo click con el ratón en la barra superior donde dice “Actividades”; otra es moviendo el ratón a la esquina superior izquierda; y una tercera más es presionando la tecla Super [4].

No importa como entremos a las actividades, se nos presentará una pantalla parecida a la figura 2.5.

[4] En los teclados de casi todas las PCs, es la tecla con el logo de Windows.

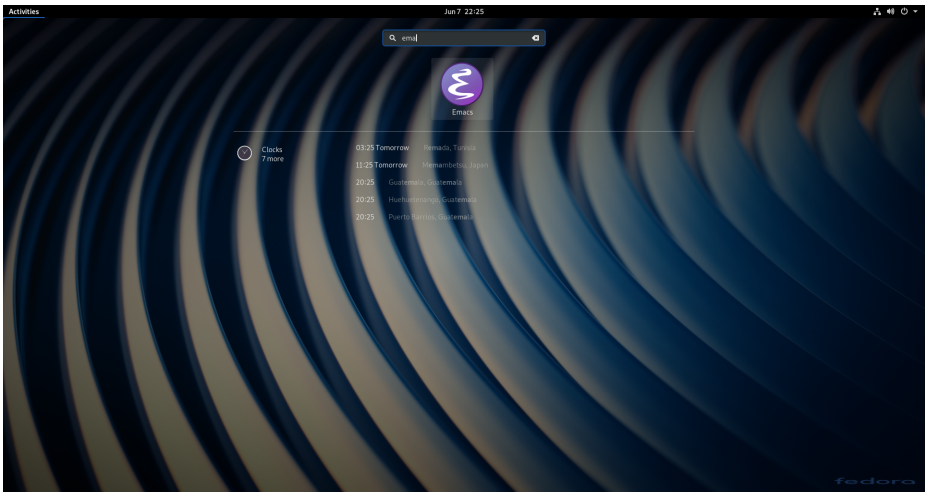
Figura 2.5 Actividades



La barra vertical a la izquierda es el tablero, y ahí podremos poner las aplicaciones que más usemos. Veremos esto en un momento.

Para lanzar una aplicación, si sabemos el nombre de la misma podemos sencillamente escribir su nombre; GNOME shell comenzará a buscar todas las aplicaciones, contactos y documentos que cacen con la cadena que escribamos (figura 2.6).

Figura 2.6 Buscando Emacs



Si no queremos escribir el nombre de la aplicación, o si no lo conocemos, podemos

buscarla en las aplicaciones, que es el último icono de arriba hacia abajo en el tablero (figura 2.5). Esto nos presentará todas las aplicaciones instaladas en el sistema (figura 2.7).

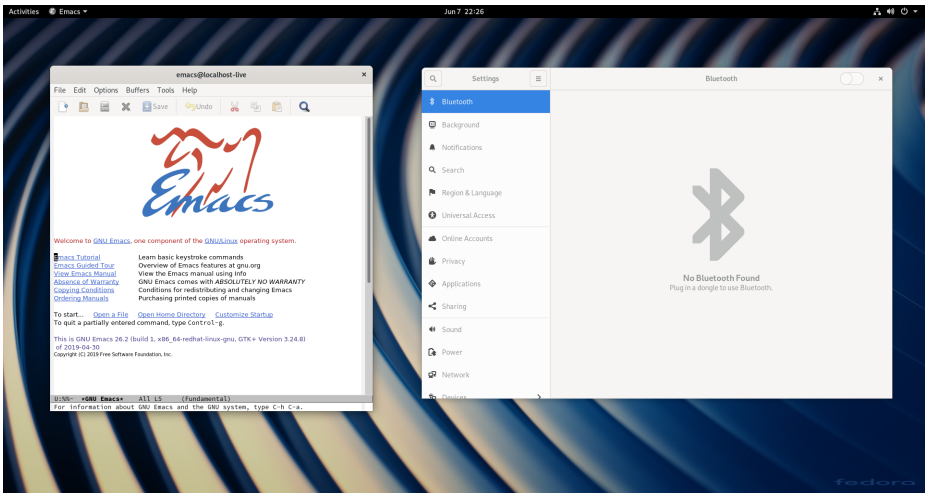
Figura 2.7 Aplicaciones

De cualquiera de estas maneras, podemos arrastrar el icono de Emacs al tablero, para poder lanzarlo de forma más rápida (figura 2.8).

Figura 2.8 Agregando Emacs al tablero

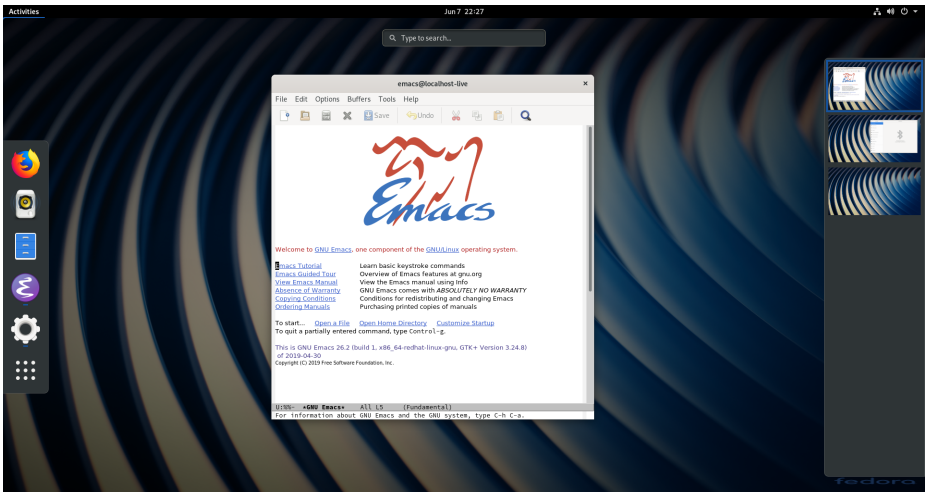
Ya que tengamos varias aplicaciones abiertas, en las actividades podemos verlas todas si están en el mismo escritorio virtual (figura 2.9).

Figura 2.9 Varias aplicaciones



El shell de GNOME soporta múltiples escritorios virtuales dinámicos, lo que nos permite organizar las aplicaciones que tengamos abiertas (figura 2.10).

Figura 2.10 Escritorios virtuales



Debe quedar claro que para usar GNOME 3 de forma eficiente, se prefiere el uso del teclado por encima del uso del ratón; se recomienda que consultes el sitio de atajos para el GNOME shell[5], pero aquí te damos los más comúnmente usados:

[5] `https://wiki.gnome.org/Projects/GnomeShell/CheatSheet`

Atajo	Descripción
System (Windows)	Cambia entre el escritorio y las actividades.
Alt + F1	Cambia entre el escritorio y las actividades.
Alt + F2	Lanza diálogo para ejecutar comando
Alt + Tab	Cambia aplicaciones
Alt + Shift + Tab	Cambia aplicaciones en el orden inverso
Alt + [tecla arriba de Tab]	Cambia entre ventanas de la misma apliación
Ctrl + Alt + Tab	Cambia entre elementos de shell
Ctrl + Shift + Alt + R	Inicia y detiene grabar el escritorio en un video
Ctrl + Alt + ↑/↓	Cambia escritorios virtuales
Ctrl + Alt + Shift + ↑/↓	Mueve la ventana actual a un escritorio virtual distinto

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 5.2); o cómo ejecutar comandos; o un intérprete de comandos de Unix dentro de Emacs (Sección **??**). En la sección anterior viste cómo lanzar Emacs desde GNOME.

2.4. Comenzando con Emacs

$ yum 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[6], la cual, junto con XEmacs[7] – 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.

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

[7] 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*.

3 Sistema de Archivos

3.1. El sistema de archivos

Antes hablar sobre el sistema de archivos, debemos establecer lo que es un archivo. Un archivo puede entenderse como una secuencia de bytes con un significado concreto, que almacena cierta información, se guarda en un dispositivo de memoria(como un disco duro, un CD, un DVD o una unidad de almacenamiento USB) y tiene un nombre que lo identifica. Dentro de Unix podemos encontrar 3 tipos de archivos:

- Archivos regulares. Son todos aquellos que guardan información en si mismos, y pueden tener una extensión en el nombre que indica de que clase es la información que este almacena. 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

- 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.

- Directorios[1], pueden contener archivos, incluso otros directorios y archivos especiales. El nombre de un directorio identifica a los archivos que este agrupa.

Dentro de la computadora, toda la información que se guarda es vista como archivo. Debido a esto, es necesario definir una forma de organizarlos de manera que podamos trabajar con ellos. De no existir una organización, la memoria sería un caos donde se guarda cualquier cosa donde sea y como sea, además de ser necesario conocer el "idioma" de cada dispositivo de memoria que se quiera usar.

El sistema de archivos es un componente muy importante en un sistema operativo. Nos permite hacer uso de los dispositivos de almacenamiento de manera transparente y sin lidiar con elementos específicos del hardware. Establece como se administran los dispositivos de memoria, la forma en que los archivos se guardan en dichos dispositivos, entre otras cosas que van mas allá del alcance de este manual. Para fines prácticos, cuando hablemos del sistema de archivos nos referiremos a la estructura en que se organizan los archivos para ser manipulados por el usuario.

Unix organiza su sistema de archivos como una jerarquía de directorios. La jerarquía de directorios es un árbol (*tree*), pero dibujado de cabeza o de lado. Un directorio especial, `root`, es la raíz de la jerarquía y se denota por el carácter '`/`'. Dicho directorio almacena todos los archivos del sistema.

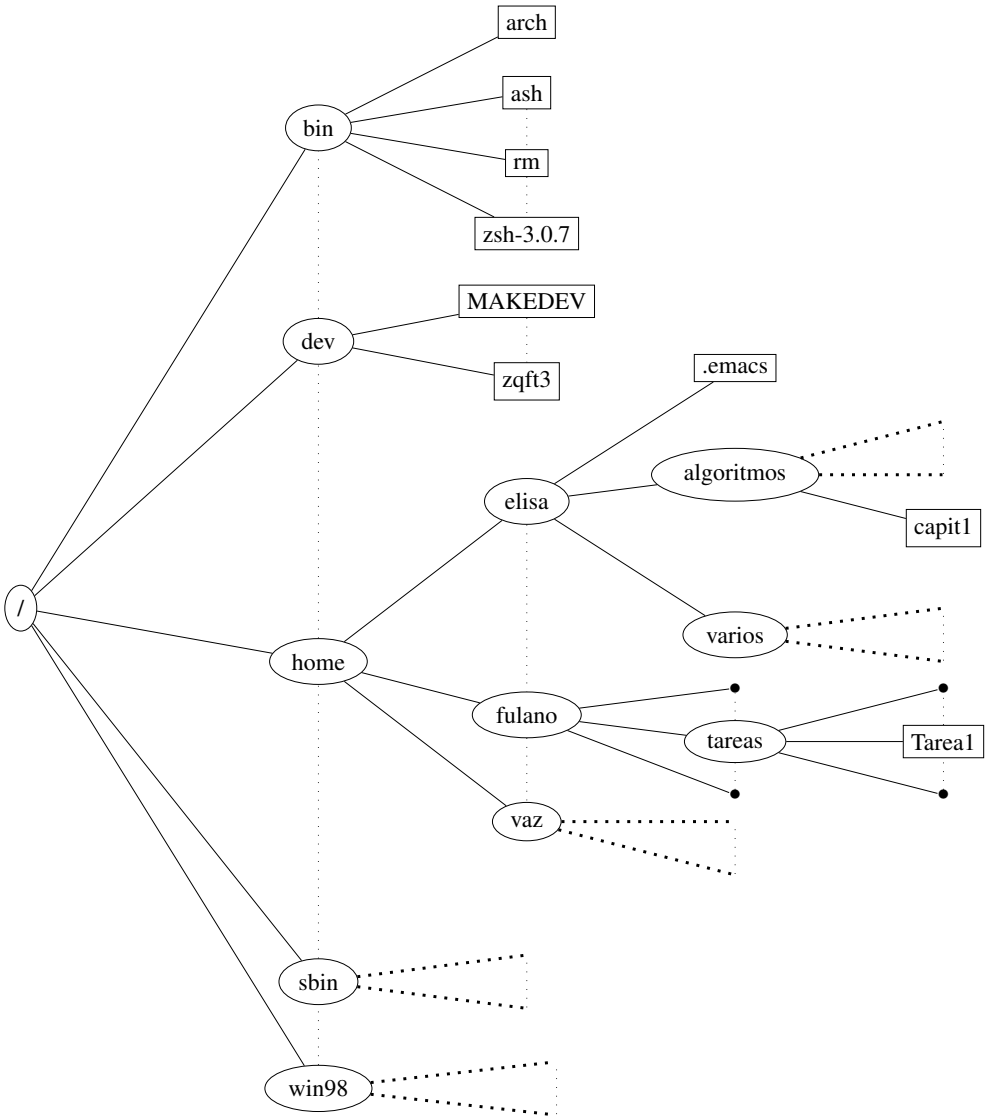
En los sistemas Unix, cada usuario tiene un directorio personal, en el cual almacena sus archivos. Dicho directorio se conoce como el *hogar (home)*. Una vez que entres en sesión, cada programa que se ejecuta se encuentra situado en un *directorio de trabajo*. El shell inicia, por omisión, en el directorio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

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

[1] Los directorios también se conocen como carpetas.

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

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

Para las siguientes secciones nos apoyaremos en una herramienta fundamental en Linux, la terminal. Se mencionarán conseptos que detallaremos en el respectivo capítulo.

Por omisión, la terminal 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

La forma de manipular los archivos en Linux es a través de tres operaciones básicas: *lectura* (*read*), *escritura* (*write*) y *ejecución* (*execute*). Todo lo que necesites de los archivos lo puedes hacer con estas operaciones. Ahora bien, cómo dijimos antes, Linux es un sistema multiusuario, por lo que sería potencialmente caótico que todos los usuarios tengan libertad para operar cualquier archivo de cualquier usuario. Para evitar esto contamos con los permisos.

Los *permisos de acceso* a un archivo, o simplemente *permisos*, especifican quién puede hacer qué a un archivo. Ya sea para evitar que un archivo importante sea borrado o que un archivo confidencial sea visto. Dependiendo de si se trata de un archivo normal o directorio, los permisos de comportan un tanto diferente. 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.

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.

La Terminal | 4

En este capítulo exploraremos una de las herramientas más útiles e importantes en un sistema Unix y algunas de las razones por las cuales resulta extremadamente conveniente familiarizarce con ella. Comenzaremos por definir lo que es una terminal, luego explicaremos algunos detalles para entenderla mejor, también algunos trucos que facilitan su uso y ayudan a realizar las tareas en unos cuantos comandos. Finalmente, mostraremos una alternativa para sustituir todos los comandos por una sola instrucción que teclear.

La terminal, línea de comandos, o consola es un interprete de comandos que nos permite interactuar con la computadora en modo de texto, es decir sin la necesidad de un entorno gráfico. Por otro lado, los comandos son instrucciones que se le dan a la computadora y ejecutan uno o varios programas. Normalmente veremos la terminal como una ventana negra que solo puede desplegar texto y muestra un *prompt* que indica que está en espera de recibir comandos.

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.

4.1. Sintaxis estándar de comandos

POSIX[1] 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[2]). 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 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.

[1]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.

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

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

4.2. 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 caracteres.

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 4.1 *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 caracteres, incluso la secuencia vacía, que es aquella que no tiene ningún carácter.

Actividad 4.2 *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 caracteres dentro de los corchetes proporcionan una lista para que se trate de casar con alguno (y sólo uno) de ellos.

Actividad 4.3 *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 4.4 *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/.

4.3. 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

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

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.

Actividad 4.5 *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 4.6 *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.*

4.4. 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.

4.5. Citas (*quotations*)

Todos los shells estándar asignan significados especiales a ciertos caracteres, llamados *meta-caracteres*. Para usar estos caracteres como caracteres 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-caracteres 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 caracteres que le sucedan, lo cual le da a esos caracteres 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 caracteres, generalmente llamados caracteres 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 caracteres 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 caracteres normales.

4.6. Procesos y Tareas

En términos simples, un proceso es la instancia de un programa en ejecución. El sistema operativo asigna un número entero positivo como identificador a cada proceso.

Ver procesos

Para poder ver los procesos en ejecución, se cuenta con el comando ps

```
fulanito@[estnum fulanito]% ps
 PID TTY          TIME CMD
 3976 pts/2    00:00:00 bash
 4272 pts/2    00:00:00 ps
```

Cuando se usa como se muestra arriba, muestra solo los procesos asociados a la terminal donde se ejecutó el comando. Para ver todos los procesos de la computadora se tiene la opción $-a$

Por otro lado, cuando un proceso es asociado a una terminal, es decir que esa terminal lo creo, se dice que es una tarea de esa terminal. Las tareas tienen su propio identificador a nivel terminal, independiente de el identificador de proceso. La forma de ocupar este identificador es precedido de %.Para ver las tareas se usa el comando jobs

```
fulanito@[estnum fulanito]%jobs
[1]-  Ejecutando              evince moderno.pdf &
[2]+  Ejecutando              emacs &
```

terminar procesos

En caso de que necesites terminar un proceso es posible usar el comando kill. Para usarlo se debe pasar como argumento el número identificador del proceso, o de tarea, y la opción -KILL

```
fulanito@[estnum fulanito]% kill -KILL 3976
```

Niveles de ejecución

En la terminal contamos con dos niveles de ejecución para los programas. Cuando se ejecuta un comando sin ninguna alteración, este se lanza en primer plano y es por esto que la terminal se mantiene ocupada hasta que dicho proceso termina. Sin embargo, es posible ejecutar más de un programa en una terminal y para eso contamos con el segundo plano.

Para pasar un proceso en ejecución a segundo plano, usamos la combinación de teclas C-z . Inmediatamente después, la terminal se liberará para ejecutar mas programas o comandos y el proceso que se ejecutaba pasa a estar suspendido y en segundo plano.

```
fulanito@[estnum fulanito]% emacs
^Z
[1]+  Detenido                emacs
```

Para que este se reanude en segundo plano utilizamos el comando bg.

```
fulanito@[estnum fulanito]% bg
[1]+ emacs &
```

Es posible lanzar un proceso en segundo plano, y se ejecute desde que se crea. Esto se hace colocando el símbolo & después del comando.

```
fulanito@[estnum fulanito]% emacs &
[1] 3984
```

Los valores que podemos ver son el identificador del proceso en el sistema y un identificador a nivel de la terminal.

Para traer a primer plano el proceso se usa el comando fg

En caso de que se quiera trabajar con varios programas, es necesario especificar que proceso se quiere pasar a primer o segundo plano. el solo usar fg no funcionará puesto que solo regresa un proceso. Con el comando jobs podemos ver los procesos que se ejecutan en la terminal y además nos muestra un número. Si tecleamos fg %N regresará a primer plano ese proceso que tenga el numero N. Para cerrar un proceso que está en primer plano se puede usar C-d.

4.7. Utilidades de la terminal

A lo largo de la sección, podríamos notar que lo que hicimos con la terminal, se puede realizar igualmente con el entorno gráfico. La ventaja es que la terminal nos permite hacerlo mucho más rápido. En el tiempo que localizas el cursor, lo posicionas en el icono para abrir el explorador de archivos, y das clic; ya pudiste acceder a casi cualquier directorio del árbol. Más aún, puedes tener cosas muchos mas cómodas con la terminal, que resultan engorrosas en el entorno gráfico.

Por ejemplo, entrar al directorio Escritorio y ver los archivos que tiene y los permisos de los mismos puede hacerse con

```
ls ~/Documentos/
```

Durante tu paso por el mundo de Linux te encontrarás conviviendo de forma continua con la terminal de Linux, por lo cual es bueno conocer ciertas características de esta que pueden hacer que tal convivencia sea mucho mas amena. A continuación te presentamos algunos “trucos” que son desde muy útiles hasta fundamentales en la terminal.

Manual de los programas

Muchos programas vienen con una descripción de su funcionamiento y las opciones con las que cuentan. Para poder visualizar esta descripción puedes usar el comando "man"luego del nombre del programa. Para desplazarte en la descripción de un programa puedes usar las flechas del teclado o la tecla Enter y para salir debes presionar la tecla q.

```
man ls
```

Historial de comandos

Los comandos que ingresas en la terminal se guardan en un archivo en el directorio home. El archivo se llama .bash_history y puedes revisar su contenido con cualquier editor de texto.

```
emacs .bash\_history
```

Historial de comandos

Otra forma de visualizar el historial, es mediante el comando history, el cual te permite, entre otras cosas, ver una cierta cantidad de comandos.

```
history 20
```

Comando anterior

Si lo que necesitas es reutilizar el último comando que escribiste, puedes presionar ↑ para pasar al comando anterior. Puedes repetir esta acción para recorrer uno por uno los comandos que hayas ingresado a la terminal.

Búsqueda por coincidencia

En caso de que recuerdes parte del comando que quieras buscar, puedes presionar C-r. Esto despliega una entrada de texto que muestra comandos según encuentre coincidencias con el texto escrito. Esto es especialmente útil cuando buscas un comando largo, no recuerdas como deben usarse las opciones, o requieres una invocación con ciertos argumentos que ya usaste previamente.

```
(reverse-i-search)`ls': ls|grep correos.txt
```

Y para seleccionar el comando, presiona C-a

Auto completar

Siguiendo con la búsqueda de la comodidad, la terminal puede terminar de escribir un comando o nombre de archivo por ti. Luego de escribir algo en la terminal, si presionas Tab, se buscaran nombres de archivos o programas que empiecen con el texto previo. En caso de existir solo una coincidencia, la completará automáticamente, si existen mas, te mostrará las diversas opciones.

Copiar y pegar

Como ya mencionamos antes, la combinación de teclas C-c termina un proceso, por lo que no podemos usarla para copiar texto. Las teclas que se deben presionar para copiar texto de la terminal son Shift + C-c. De manera similar si se quiere pegar texto en la terminal se debe teclear Shift + C-v.

4.8. Script para la terminal

Es posible que te encuentres en situaciones en que requieras de la ejecución de la misma secuencia de comandos de forma repetida. Para evitar escribir varias veces lo mismo o en su defecto navegar por tu historial de comandos, es posible juntar toda la secuencia en un archivo de modo que este se ejecute como si fuera un guión de lo que la terminal debe hacer. A estos archivos que contienen comandos para la terminal se les conoce como scripts

Como crear un script

Para hacer un script, basta con escribir en un archivo de texto plano los comandos que quieras ejecutar. Este archivo debe llevar la extensión .sh, además, dado que la idea es que el archivo sea ejecutado, es necesario agregar el permiso de ejecución. También es altamente recomendable que la primer línea sea !#bin/bash. Esto indica que programa interpretará el archivo. No es necesario si el script se ejecuta desde la terminal, pero en caso contrario no será posible.

La ejecución del script se realiza escribiendo la ruta del archivo que contiene las instrucciones. En caso de que el archivo esté guardado en el directorio de trabajo, es necesario agregar la referencia al directorio actual, es decir ./, seguido del nombre del archivo.

```
./guion.sh
```

Argumentos

Aún cuando ya sea posible usar un solo script para englobar todos los comandos que necesitamos, si estos necesitan una variación en los argumentos, el guión que tenemos no funcionará por si solo. Para evitar la creación de un nuevo script para cada posible entrada, podemos manejar argumentos. Para usar un argumento en el script, solo debes poner el símbolo $ seguido del numero de argumento que quieres usar.

```
#!bin/bash
mkdir $1
ls $2 > $3.txt
```

Básicamente, la instrucción recibe al menos tres argumentos, crea un directorio que lleva por nombre el primer argumento, obtiene los archivos que contenga el directorio cuya ruta es el segundo argumento y guarda el resultado en un archivo que llevará por nombre lo que se reciba como tercer argumento con extensión ".txt".

Variables

Las variables son espacios de memoria que guardan información y tienen un identificador. Para definir una variable se debe escribir el identificador, el signo de igualdad, sin espacios, y la información que guardará la variable. Por ejemplo podemos hacer una variable que guarde una ruta

```
ruta=~/Documentos/propedeutico
```

Y para poder acceder al valor de la variable, al igual que con los argumentos, usamos $.

```
ls $ruta
```

En caso de que el valor que se desea guardar contenga espacios, es necesario hacer un ajuste, ya que la terminal considera los espacios como limite de expresiones. Para poder guardar un valor con espacios es necesario ponerlo entre comillas. En el caso de la terminal, tanto las comillas simples como dobles funcionan pero hay una diferencia importante entre ambas que hay que resaltar: Cuando se usan comillas dobles, es posible agregar variables y que están sustituyan su valor por el que guardan. En caso de usar comillas simples, no se hará esta sustitución y dicho valor se mantendrá tal cual se escribió.

Por ejemplo:

```
msg1='Esta es la ruta $ruta'
msg2="Esta es la ruta $ruta"
echo $msg1
echo $msg2
```

Dará como resultado

```
Esta es la ruta ~/Documentos/propedeutico
Esta es la ruta $ruta
```

Algo interesante que podemos guardar en una variable, es el resultado de la ejecución de un comando. Esto se hace poniendo dicho comando entre paréntesis y todo precedido por el símbolo $

```
lista=$(ls -al $ruta)
```

4.9. Variables de entorno y alias

Para este punto, resulta ya muy cómodo haber pasado de pelearnos con los clics y selección de texto, a solo uno cuantos comandos y finalmente a solo ejecutar un programa. El único inconveniente que puede surgir es la necesidad de saber la ruta del archivo con el script y escribirla cada vez que usamos el programa. Sin embargo, es posible resumir todo en una sola línea a través de los alias.

Un alias es un nombre alternativo que le puedes dar a una cadena de texto. Esta cadena puede ser una ruta, un comando, un valor, etc. Para poder crear un alias usamos el comando alias seguido del nombre alternativo que queremos dar, luego el símbolo = sin dejar espacios, y finalmente, entre comillas ‘ ‘ la cadena a la que queremos poner el alias.

```
fulanito@[estnum fulanito]% alias lal='ls -al'
```

El resultado de lo anterior es que al escribir lal en la terminal, se ejecutará el comando ls con las opciones que se indicaron.

Los alias funcionan únicamente en la terminal en que se crearon, lo que significa que es necesario crearlos cada vez que se inicia a menos que hagamos una pequeña modificación en el lugar adecuando.

En Linux existen una serie de archivos de configuración que se ejecutan cada vez que se inicia sesión. Uno de esto es el archivo es .bashrc que se encuentra en el directorio home y configura características de la terminal. Es altamente recomendable no borrar o modificar esta clase de archivos a menos que se tenga plena conciencia de lo que se hará. Sin embargo, la modificación necesaria es muy pequeña e inofensiva(siempre y cuando no se borre nada durante el proceso). Si agregas la definición de un alias al final de este archivo, podrás usar el alias en cualquier terminal sin necesidad de volver a definirlo.

Variables de entorno

Como mencionamos antes, para ejecutar un programa, es necesario escribir la ruta en que se encuentra el archivo que contiene dicho programa. Por otro lado, los comandos de la computadora también son programas y por tanto requieren saber donde está guardado el archivo donde se ejecutarán.
Resultaría muy complicado y molesto tener que buscar, aprender y escribir la ruta de cada comando por lo que esto no suena una buena opción. Tampoco es adecuando pensar en alias pues, además de que es necesario crear uno para cada caso, no es posible pasar argumentos. Para solucionar este problema, en Linux contamos una variable que guarda las rutas donde están los posibles comandos y con esto ya no es necesario que el usuario lidie con eso. Esta variable es llama PATH y es lo que se conoce como una variable de entorno.

Las variables de entorno son aquellas que guardan información importante para el sistema operativo. Para poder verlas vasta con escribir su nombre precedido del símbolo $. Por ejemplo $PATH.

Otras variables importantes son

Tabla 4.1 Variables de entorno básicas

Nombre	Descripción
$HOME	Guarda la ruta del directorio hogar del usuario
$SHELL	Guarda el programa que se usa como terminal por omisión
$PATH	Guarda las ruta de los directorios que tienen los programas que son comandos. Si un directorio aparece en esta variable, sus programas pueden ser invocados desde la terminal sin necesidad de especificar una ruta
$PS1	Guarda la cadena que forma el promt principal que se muestra en la terminal
$USER	Guarda el nombre de usuario actual

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

Tabla 4.2 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 4.2 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 4.2 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 4.2 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 4.2 Comandos de Unix importantes *Continúa de la página anterior*

Comando	Descripción
wc	El comando wc cuenta el número de caracteres, palabras y líneas en un archivo o conjunto de éstos. Una palabra es considerada como una secuencia (no vacía) de caracteres 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.

Actividad 4.7 *Escribe las rutas relativas para referirte a los mismos archivos anteriores, desde los directorios* /, /usr/share/, /facultad/calculo/ayudantia/ *y* /usr/local/bin.

Actividad 4.8 *Crea un directorio llamado* miPractica *que contenga dos subdirectorios llamados* usuarios *y* archivo *y que estén en tu directorio base.*

Actividad 4.9 *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 4.10 *A todos los directorios que haz creado asígnales permisos de lectura y escritura para ti y sólo de lectura a los demás.*

Actividad 4.11 *A todos los archivos que haz creado asígnales todos los permisos para ti y ningún permiso para los demás.*

Actividad 4.12 *Crea una copia del directorio* miPractica *y todo su contenido, llámale* ejercicios.

Actividad 4.13 *Elimina el directorio* miPractica *y todo su contenido.*

Edición 5

5.1. 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 referiremos a él por su nombre completo: vi vi vi (666, en romano), el editor de la bestia.

[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.

Hablaremos de Emacs más adelante, en la sección 5.2. Linux, y de manera más general el software libre, te ofrece una única cosa y hay que valorarla: la facultad de elegir.

5.2. 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 5.1).

En el capítulo anterior (sección **??**) 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.

5.2.1. 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 **??** 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 5.2.11); recuerda teclear C-x C-c para terminar la ejecución de Emacs.

5.2.2. 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 5.1 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 5.2 Emacs: movimiento a lugares especiales

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

5.2.3. 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 caracteres.

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

Tabla 5.3 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

5.2.4. Reencarnación de texto

Cuando matas alguna sección, los caracteres 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 caracteres que han muerto en este buffer (o en cualquier otro buffer); a esta operación 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.

5.2.5. 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 caracteres 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 caracteres 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.

5.2.6. Rectángulos

En Emacs, además de poder hacer selecciones de caracteres 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 5.4 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

5.2.7. 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 5.5 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.

5.2.8. Archivos

Como mencionamos en el capítulo anterior (sección **??**), 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).

5.2.9. 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.

5.2.10. 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–%.

5.2.11. 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*.

5.2.12. 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.

5.2.13. 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.

5.2.14. 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.

5.2.15. 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.

5.2.16. 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.

5.2.17. 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.

```
$ yum 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 caracteres 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.

```
(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")
```

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 5.6 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.
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.

5.2.18. 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)
```

Control de versiones

6

Cuando se edita un archivo de texto, especialmente si está relacionado con tareas computacionales (código fuente de un programa, *scripts*, archivos de configuración en Unix), esto suele ser una tarea *destructiva*. No nada más se agregan nuevas líneas de texto a un archivo; muchas veces se destruyen (eliminan) o modifican muchas de las líneas que ya existían.

Esto es completamente normal; pero a veces nos gustaría mantener una versión que ya funciona (o con varias partes avanzadas), antes de comprometernos a modificarla.

Es en este tipo de situaciones donde entra el concepto de control de versiones.

6.1. Control de versiones

La necesidá de mantener un control de versiones fue reconocida en el ámbito de la programación básicamente desde el inicio de la misma. Lo que hacen los sistemas de control de versiones es, como su nombre indica, controlar versiones: un grupo de programadores puede decidir que los archivos de un proyecto han llegado a un punto que vale la pena preservar (o asignar versiones), así se lo hacen saber al sistema de control de versiones y a partir de ese momento ese estado (o *versión*) del proyecto se preserva para toda la posteridad. Se pueden modificar los archivos del proyecto (o agregar e incluso elimiar algunos), pero si así se desea se puede recuperar (de preferencia *fácilmente*) la versión anterior.

Esto además es un proceso acumulativo; si después de haber preservado una versión se

decide preservar una nueva, ambas estarán disponibles siempre[1], así como cualquier otra versión futura que se quiera preservar. Un sistema de control de versiones entonces mantiene un historial de *todas* las versiones preservadas de un proyecto, con la única limitación siendo el espacio en disco duro disponible en la computadora: el kernel de Linux utiliza 7 gigabytes, a pesar de que su sistema de control de versiones utiliza varias optimizaciones para ahorrar espacio en disco. Sin embargo es un ejemplo extremo: el kernel de Linux consiste de aproximadamente 25 millones de líneas de código desarrolladas a lo largo de 26 años por miles de autores.

El control de versiones es útil en sí mismo; sin embargo se vuelve *particularmente* útil al trabajar en equipo: cada integrante del equipo puede tener una copia del proyecto y preservar los cambios que le parezcan convenientes. Si hay conflictos o se introducen errores, es (relativamente) sencillo regresar a una versión que se sabe que funciona.

6.1.1. Sistemas centralizados

En 1972 Marc Rochkind creó en Bell Labs el Source Code Control System (SCCS), pero no estuvo disponible al "público" hasta 1977 (entre comillas porque el uso de las computadoras era muy limitado en ese entonces para el público en general). En 1982 SCCS fue en general reemplazado por el Revision Control System (RCS) escrito por Walter Tichy.

En ese entonces tanto SCCS como RCS funcionaban a nivel de archivo, no de proyecto; eso quiere decir que uno mantenía versionados archivos individuales, no conjuntos de los mismos. No era extraño en esa época que muchos proyectos consistieran de un puñado de archivos, si no es que uno solo.

En 1986 Dick Grune comenzó a trabajar en una serie de *scripts* de shell que eventualmente se convertirían en el Concurrent Versions System (CVS), liberado oficialmente en 1990. CVS ya podía trabajar con directorios (rudimentariamente), lo que permitió mantener control de versiones de proyectos mucho más complejos. A un directorio o conjunto de directorios que estén bajo control de versiones se le suele denominar un *repositorio*.

El proyecto GNU que mencionamos en el capítulo 1 fue anunciado en 1983 por Richard Stallman; el control de versiones es *fundamental* para el desarrollo del Software Libre, por razones tanto técnicas como sociales (el control de versiones permite, entre otras cosas, determinar de manera clara quién es el autor de qué modificaciones). CVS se convirtió en miembro del proyecto GNU.

CVS es un sistema de control de versiones *centralizado*; esto quiere decir que existe una copia "central", que es realmente la que está bajo el contro de versiones, y múltiples usuarios pueden bajar una copia de la última versión del proyecto. Cuando un usuario a realizado cambios a su copia, puede enviar los mismos al repositorio central para que el resto a su vez los tengan disponibles; sin embargo, el historial de versiones anteriores únicamente se encuentra en el repositorio central.

[1] Siempre y cuando no se elimine el proyecto mismo, obviamente.

CVS fue el sistema de control de versiones utilizado por básicamente todos los proyectos de software libre desarrollados en y para Linux en la década de los noventas, a pesar de todas las limitaciones que tiene. Sin embargo, una excepción muy importante fue el mismo kernel de Linux: a pesar de que siempre estuvo disponible un repositorio CVS para el mismo, no es lo que utilizaba Linus Torvalds, el creador y dictador benevolente de Linux que es el que tiene la decisión final de qué se incluye o no en el kernel.

Inicialmente Linus utilizaba su correo electrónico para recibir *parches* que después aplicaba manualmente. Un parche o archivo diferencial (*diff*) es un archivo de texto con un formato especial que permite ver las diferencias entre dos archivos de texto. En lo general en un sistema de control de versiones, las distintas versiones de un archivo pueden describirse con una secuencia de parches. Se les denomina parches porque justamente "parchan" un proyecto, con la idea de que un parche repare o alivie un error en el código.

El aplicar parches manualmente es, por supuesto, increíblemente ineficiente, pero Linus se negaba a utilizar sistemas de control de versiones porque (él aducía) ninguna hacía lo que él necesitaba. Esto continuó durante todo el final del siglo XX.

6.1.2. Sistemas distribuidos

En 1998 la compañía BitMover comenzó a trabajar en un sistema de control de versiones particularmente orientado a las necesidades de Linus y, por extensión, Linux. En el año 2000 BitKeeper fue liberado y dos años después comenzó a ser utilizado por Linus y, de manera gradual y a regañadientes, por el resto de la comunidad de desarrolladores del kernel de Linux.

La razón de que muchos desarrolladores se resistieran a usar BitKeeper es que era un sistema de control versiones que no es Software Libre; la compañía BitMover lo vendía, aunque a los desarrolladores del kernel de Linux se les proporcionaba automáticamente una licencia gratuita.

BitKeeper no es un sistema de control de versiones centralizado, como CVS o Subversion (que es una mejora incremental sobre CVS); es un sistema de control de versiones *distribuido*. Esto quiere decir que no existe un repositorio central; cada copia del mismo es un repositorio en sí misma, con toda la historia del proyecto incluida en ella.

La resistencia a BitKeeper y el hecho de que no fuera Software Libre resultó (junto con otras razones) en la creación de varios sistemas de control de versiones distribuidos, probablemente los más famosos siendo Mercurial y Bazaar.

En 2005 BitMover decidió retirar la versión gratuita de BitKeeper que proporcionaba a los desarrolladores del kernel de Linux, supuestamente porque algunos de ellos habían tratado de aplicar ingeniería reversa al protocolo de comunicación del mismo, así que Linus decidió tomarse (literalmente) unas semanas para escribir su propio sistema de control de versiones distribuido, porque ninguno en existencia hacía lo que BitKeeper.

Después de unas cuantas semanas de desarrollo, Linus liberó Git, que doce años des-

pués es básicamente lo que utilizan todos los proyectos de Software Libre en existencia (con algunas excepciones).

6.2. Git

Git es un sistema de control de versiones distribuido con dos características muy particulares: está pensado para hacer bifurcaciones (que explicaremos más adelante) de manera muy rápida y además utiliza criptografía para toda la historia de un repositorio.

Lamentablemente la facilidad de uso nunca fue una prioridad para Linus ni para ninguno de los siguientes desarrolladores de Git, así que el programa tiene la en general muy merecida fama de ser algo difícil de aprender a usar, especialmente sus características más oscuras. Sin embargo para cosas sencillas sólo es necesario aprender un puñado de comandos que serán los que cubriremos en esta sección.

Git es de hecho una colección de programas, los cuales en general serán invocados a través del programa principal `git`. Los comandos de Git generalmente son de la siguiente forma:

```
~ $ git comando [opciones]
```

Lo que hace el programa `git` es tomar el comando que recibe como parámetro, `comando` en el ejemplo de arriba, y ejecutar `git-comando`. En una instalación normal de Git, esto resulta en cientos de programas para los comandos de Git.

Para crear un repositorio podemos comenzar con un directorio vacío o que ya contenga archivos, es lo mismo. Nosotros crearemos un nuevo repositorio desde cero en nuestros ejemplos:

```
~ $ mkdir ejemplo
~ $ cd ejemplo
~/ejemplo $ git init .
```

Recordemos que usar "." sirve para referirnos al directorio actual, así que el comando que acabamos de usar se traduce en pedir a git que cree un repositorio asociado al directorio que pasamos como parámetro (en este caso .); si el directorio no existe, se crea uno nuevo vacío con el nombre que hemos pasado, por lo tanto una forma de inicializar un repositorio desde cero es:

```
~ $ git init ejemplo
~ $ cd ejemplo
```

El comando `init` de Git únicamente crea el directorio `.git` con una configuración por omisión. El repositorio está vacío; si listamos los archivos en el directorio no aparecerá nada excepto el directorio `.git`:

```
~/ejemplo $ ls -la
total 56
drwxr-xr-x   3 user user  4096 Jul 17 19:57 .
drwx------ 119 user user 45056 Jul 17 19:57 ..
drwxr-xr-x   6 user user  4096 Jul 17 19:57 .git
```

El estado del repositorio también es vacío, dado que no hemos hecho nada todavía; podemos revisar el estado de un repositorio con el comando `status`:

```
~/ejemplo $ git status .
On branch master
Initial commit
nothing to commit (create/copy files and use "git add" to track)
```

Más adelante veremos qué quiere decir exactamente "branch master". Vamos a agregar un archivo a nuestro repositorio; en Emacs crea el archivo `capitan.txt` dentro del directorio `ejemplo`, escribe lo siguiente en el mismo y guárdalo:

```
Va de cuento: nos regía
Un capitán que venía
Mal herido, en el afán
De su primera agonía.
```

Ahora revisa el estado de tu repositorio; algo del estilo debería aparecerte:

```
~/ejemplo $ git status .
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        capitan.txt
nothing added to commit but untracked files present
(use "git add" to track)
```

Git nos dice que hay archivos fuera del control de versiones ("untracked files") y que podemos agregarlos usando el comando `add`, así que hagamos eso:

```
~/ejemplo $ git add capitan.txt
~/ejemplo $ git status .
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   capitan.txt
```

Si en este paso hubiéramos creado más de un archivo en nuestro repositorio y quisiéramos agregar **todos** al control de versiones, la forma de agregar todos los cambios que hemos hecho es pasar el parámetro `-A` (contracción de `-a`) en vez del nombre de un archivo, el comando quedaría de la siguiente manera:

```
~/ejemplo $ git add -A
```

Existen comandos similares para agregar al control de versiones los cambios en más de un archivo a la vez:

`git add .` Agrega las modificaciones de archivos existentes y los creados después de la última vez que se cargaron cambios. **No carga las remociones de archivos**.

`git add -u` Agrega las modificaciones de archivos existentes y los eliminados después de la última vez que se cargaron cambios. **No carga los archivo que se crearon recientemente**.

Ahora Git nos dice que hay un archivo nuevo, pero que no hemos realizado el cambio. Esto es importante; Git no realizará ningún cambio (no quedará preservado en el historial de versiones) hasta que realicemos (o cometamos, "commit") el mismo. También nos dice el comando para eliminar `capitan.txt` del control de versiones con el comando `rm`.

Vamos a realizar el cambio para que quede en el historial de versiones:

```
~/ejemplo $ git commit capitan.txt -m "Inicio del verso"
[master (root-commit) ae1cf70] Inicio del verso
 1 file changed, 4 insertions(+)
 create mode 100644 capitan.txt
```

El parámetro `-m` nos permite agregar el mensaje que queremos asociar al realizar nuestros cambios; si no la especificamos, Git abrirá un editor para que escribamos el mensaje. No podemos realizar un cambio si no escribimos un mensaje asociado al mismo.

Si revisamos el estado del repositorio, Git nos informará que no existe ninguna modificación:

```
~/ejemplo $ git status .
On branch master
nothing to commit, working tree clean
```

A partir del momento en que realizamos el cambio, esta versión del archivo `capitan.txt` quedará preservada siempre que el repositorio exista; no importa cómo modifiquemos el archivo o incluso que lo eliminemos, siempre podremos recuperar esta versión.

Para ejemplificar esto, vamos a terminar el verso; en Emacs agrega las dos últimas líneas del verso para completarlo y guárdalo:

```
Va de cuento: nos regía
Un capitán que venía
Mal herido, en el afán
De su primera agonía.
¡Señores, qué capitán
el capitán de aquel día!
```

Si ahora verificamos el estado del repositorio, nos dirá que nuestro archivo ha sido modificado.

```
~/ejemplo $ git status .
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in
   working directory)
        modified:   capitan.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

Este ejemplo con un único archivo es muy sencillo, pero si tuviéramos muchos archivos que hemos modificado a lo largo de varios días (lo cual es muy común al programar), es útil ver qué hemos modificado en el repositorio. Para esto podemos usar el comando `diff`, que nos permite ver la diferencia entre la última versión realizada y nuestras nuevas modificaciones:

```
~/ejemplo $ git diff .
commit 8b0d526a13abc49c5b3fcf4a9a1571dad3e3e69c (HEAD -> master)
Author: User <user@unam.mx>
Date:   Thu Jul 17 20:25:58 2017 -0500

    Verso completado

diff --git a/capitan.txt b/capitan.txt
index 8d51b69..3e3e69c 100644
--- a/capitan.txt
+++ b/capitan.txt
@@ -2,3 +2,5 @@ Va de cuento: nos regía
 Un capitán que venía
 Mal herido, en el afán
 De su primera agonía.
+¡Señores, qué capitán
+el capitán de aquel día!
```

En la terminal Git usará un programa paginador (generalmente `less`) para mostrar la diferencia. Podemos ver en la misma que el cambio que hicimos fue agregar las dos últimas líneas.

Para agregar este cambio al repositorio tenemos que usar de nuevo los comandos `add` y `commit`, aunque en este caso podríamos ahorrarnos el primero:

```
~/ejemplo $ git add capitan.txt
~/ejemplo $ git commit capitan.txt -m "Verso completado"
[master 8b0d526] Verso completado
 1 file changed, 2 insertions(+)
```

Git nos informa que un archivo fue modificado y que tuvo dos inserciones. Podemos ver el historial de modificaciones utilizando el comando `log`:

```
~/ejemplo $ git log .
commit 8b0d526a13abc49c5b3fcf4a9a1571dad3e3e69c (HEAD -> master)
Author: User <user@unam.mx>
Date:   Thu Jul 17 20:25:58 2017 -0500

    Verso completado

commit ae1cf70ddbd1bdd7b400267c046c2fe1571d3afb
Author: User <user@unam.mx>
Date:   Thu Jul 17 20:18:00 2017 -0500

    Inicio del verso
```

El número hexadecimal que aparece a la derecha de `commit` es el identificador de cada versión; podemos recuperar cualquier versión pidiéndole a git que revise su identificador con el comando `checkout`:

```
~/ejemplo $ git checkout ae1cf70ddbd1bdd7b400267c046c2fe1571d3afb
Note: checking out ae1cf70ddbd1bdd7b400267c046c2fe1571d3afb

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at ae1cf70... Inicio del verso
~/ejemplo $ cat capitan.txt
Va de cuento: nos regía
```

```
Un capitán que venía
Mal herido, en el afán
De su primera agonía.
```

Cuando revisamos una versión Git nos da bastante información de que ya no estamos a la cabeza de nuestra rama; explicaremos esto más adelante. Para regresar a donde estábamos, con el archivo `capitan.txt` completo, sólo debemos revisar "master":

```
~/ejemplo $ git checkout master
Previous HEAD position was ae1cf70... Inicio del verso
Switched to branch 'master'
```

Por último y antes de explicar las bifurcaciones, vamos a ver cómo eliminar un archivo del repositorio; para esto sólo hay que usar el comando `rm`. Podemos hacer esto sin ningún temor; borramos el archivo del estado *actual* del repositorio, pero el mismo siempre va a estar en el historial de versiones y siempre podremos recuperarlo.

```
~/ejemplo $ git rm capitan.txt
rm 'capitan.txt'
~/ejemplo $ git commit capitan.txt -m 'Eliminado'
[master 2a10a91] Eliminado
 1 file changed, 6 deletions(-)
 delete mode 100644 capitan.txt
```

6.2.1. Bifurcaciones

Como hemos discutido el funcionamiento de Git, podría parecer que las versiones que se van guardando forman una progresión lineal: está el estado inicial de un repositorio, se le hacen cambios y se realizan, luego se le hacen más cambios que también se realizan, etc.

En los hechos es más complejo que esto porque Git está diseñado para trabajar con múltiples personas modificando un mismo repositorio en varias computadoras; muchos usuarios de Git lo que hacen es *bifurcar* un repositorio. Al bifurcar un repositorios, todos los cambios ocurren en una rama distinta a la principal (la que Git llama por omisión "master"), y uno puede realizar modificaciones ahí de forma independiente a la misma.

Para crear una rama uno utiliza el comando `branch`:

```
~/ejemplo $ git branch pruebas
```

Esto únicamente crea la rama, en este momento seguimos trabajando en la rama "master". Para cambiarnos a la rama que creamos, se utiliza el comando de revisión, `checkout`:

```
~/ejemplo $ git checkout pruebas
Switched to branch 'pruebas'
```

La creación de una rama y el cambio a ésta puede resumirse alternativamente a un sólo paso con la opción `-b` agregada al comando `checkout`:

```
~/ejemplo $ git checkout -b pruebas2
Switched to branch 'pruebas2'
```

Después de realizar modificaciones a nuestra rama `pruebas`, podemos ver las diferencias con la rama principal:

```
~/ejemplo $ git diff master
```

Y si estamos contentos con los cambios, podemos regresar a la rama principal con `checkout` y combinarle los cambios de la rama con el comando `merge`:

```
~/ejemplo $ git checkout master
Switched to branch 'master'
~/ejemplo $ git merge pruebas
Updating 2a10a91..b6a5eb4
Fast-forward
 capitan.txt | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 capitan.txt
```

Hemos mezclado el trabajo de la rama `pruebas` con el de `master`, sin embargo la rama `pruebas2` ya no esta al día con el trabajo que hasta el momento nos convence como definitivo en `master` porque fue creada antes del comando `commit` más reciente. Para mezclar los cambios de éstas dos ramas en `pruebas2` y continuar trabajando usamos el comando `rebase`. En el caso de esta comando y `merge`, si hemos editado las mismas líneas con información diferente, tendremos un "conflicto" como mostramos a continuación:

```
~/ejemplo $ git checkout pruebas2
Switched to branch 'pruebas2'

~/ejemplo $ git rebase master
First, rewinding head to replay your work on top of it...
Applying: Cambio al final
Using index info to reconstruct a base tree...
M       capitan.txt
Falling back to patching base and 3-way merge...
Auto-merging capitan.txt
CONFLICT (content): Merge conflict in capitan.txt
Failed to merge in the changes.
Patch failed at 0001 Cambio al final
The copy of the patch that failed is found in:
```

```
   ~/ejemplo/.git/rebase-apply/patch

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run
"git rebase --abort".
```

En este momento, como hemos editado las mismas líneas en el último `commit` de ambas ramas, tenemos un conflicto y nuestro archivo "capitan.txt" se ve así:

```
Un capitán que venía
Mal herido, en el afán
De su primera agonía.
<<<<<<< HEAD
¡Señores, qué capitán
el capitán de aquel día!
=======
No era el hombre más honesto
ni el más piadoso,
pero era un hombre valiente.
>>>>>>> Cambio al final
```

El conflicto se ve señalado con "`<<<<<<< HEAD`" y "`>>>>>>> Cambio al final`". Las líneas hasta antes de la separación "`=======`" corresponde al contenido del archivo "capitan.txt" en la rama master, después de esta vemos el contenido del último `commit` en la rama pruebas2. Para solucionarlo basta con elegir sobre lo que queremos seguir editando y modificar el archivo de acuerdo a esto. En este ejemplo queremos ambos contenidos, así que simplemente eliminaremos las señalizaciones de conflicto, agregaremos los cambios y continuaremos el rebase:

```
~/ejemplo$ git add capitan.txt
~/ejemplo$ git rebase --continue
Applying: Cambio al final
Applying: Cambio al final
```

Y éste será el resultado obtenido en el archivo "capitan.txt" en la rama `pruebas2`:

```
Un capitán que venía
Mal herido, en el afán
De su primera agonía.
¡Señores, qué capitán
el capitán de aquel día!

No era el hombre más honesto
ni el más piadoso,
pero era un hombre valiente.
```

El bifurcar un repositorio nos permite trabajar en un espacio "limpio", sin preocuparnos de lo que otras personas pudieran o no hacer, y sólo al final que ya estemos seguros de los cambios que queremos realizar, combinarlos con la rama principal. Es la manera recomendada para trabajar con Git, ya que está optimizado para poder bifurcar repositorios grandes muy rápidamente.

6.2.2. Deshacer cambios

Hasta el momento hemos hecho énfasis en la importancia de realizar un commit por cada cambio significativo realizado, para poder volver a versiones anteriores de nuestro trabajo en caso de ser necesario. Cuando queremos volver al estado que hay en algún commit previo usamos el comando `reset` con alguna de las opciones que tiene, aquí explicaremos las más usuales que son "`- -soft`", "`- -hard`" y "`- -mixed`".

Primero es importante aclarar algunos términos de nuestro espacio de trabajo con git:

- **HEAD**: Es el último commit cargado en el historial de nuestra rama.
- **Index**: Es el área de ensayo (normalmente llamada *staging area*), el conjunto de archivos que se convertirán en el nuevo commit, es decir, los que vamos agregando con `git add` antes de hacer commit.
- **Copia de trabajo**: Son los archivos sobre los que estás trabajando antes de agregarlos a *Index*.

Para regresar a un commit en específico necesitamos visualizar el regitro de commits que hemos hecho con su información corespondiente usamos el comando `log` como ya lo hemos visto en la sección 6.2.

Aquí una breve explicación de las opciones principales al hacer *reset*:

- **- -hard** Esta opción deshará **todos** los cambios que no hayan sido agregados a un commit, comunmente se utiliza de las siguiente formas:

  ```
  $ git reset –hard HEAD
  ```

  Borrará todos los cambios que hayas hecho desde el último commit y te posicionará en la "cabeza" del historial del repositorio, es decir, el último commit.

  Otra forma de utilizar esta opción es indicando a qué commit quieres regresar:

  ```
  $ git reset –hard f876b67
  ```

  Este comando descartará **todos** los commits y cambios que hayas realizado después del indicado, lo cual dejará a HEAD, *Index* y la *copia de trabajo* en exactamente el mismo estado.

- **- -soft** Pone a HEAD en el commit que indiques sin eliminar cambios posteriores:

  ```
  $ git reset –soft f876b67
  ```

  Este comando colocará a HEAD en el commit indicado y todos los cambios realizados después de esto (incluyendo los siguientes commits) quedarán en Index, así que si después de ejecutar este comando realizas un commit, éste tendrá todo lo que hayas realizado después de `f878b67`. En otras palabras, *Index* y la *copia de trabajo* quedan en el mismo estado.

- **- -mixed** Esta es la opción por omisión de `reset` cuando no hay alguna especificada. Coloca a HEAD en el commit indicado mientras que reinicia a *Index* para quedar en el mismo estado y todos los cambios posteriores pasan a la copia de trabajo. Un ejemplo de uso quedaría análogo a los de las otras opciones:

  ```
  $ git reset –mixed f876b67
  ```

Cuando nos queremos referir a commits específicos que conocemos "hace cuántos commits hicimos", podemos referirnos a ellos con "HEAD~n", donde n representa cuántos commits atrás quieremos ir, si queremos referirnos al commit anterior a HEAD lo llamamos "HEAD~1", al que hicimos dos antes como"HEAD~2", y así sucesivamente. Por ejemplo, si quisiéramos hacer un reset por omisión (lo mismo que si usara `– –mixed`) al 4 commits antes de HEAD, usaríamos: `$ git reset HEAD~4`
Con lo que HEAD e *Index* quedarían en el commit que antes estaba a 4 lugares de HEAD y *la copia de trabajo* contendrá los cambios de los 4 commits que estaban delante.

6.2.3. Usos avanzados

Git es una herramienta muy versátil y poderosa y, consecuentemente, a veces muy compleja. Podría escribirse un libro entero al respecto; y de hecho ya lo hicieron y está disponible en línea en [2]. Les recomendamos que lo lean para ver una explicación más detallada y profunda acerca de Git.

6.3. GitHub

Todos los ejemplos cubiertos en este capítulo utilizan Git localmente, en un repositorio en la misma computadora en la que se trabaja. Este es un uso válido de Git, pero la herramienta es mucho más poderosa y útil cuando se utiliza con múltiples repositorios.

Por suerte existen muchos servicios en línea que proveen alojamiento para repositorios de Git (y en algunos casos otros sistemas de control de versiones). Les recomendamos que revisen GitHub [4] y GitLab [5], que les permiten tener repositorios privados, al menos para estudiantes en el caso de GitHub.

Esto es importante por dos razones: en primer lugar, si quieren tener documentos personales, deben entender que si están en un repositorio público cualquier persona podría verlos. En segundo lugar, porque si quieren mantener prácticas y proyectos escolares en repositorios en línea (que se los recomendamos), más vale que dichos repositorios sean privados: tenerlos en un repositorio público es equivalente a "pasarle la tarea" a sus compañeros.

Del otro lado de la moneda, mencionamos que les conviene tener proyectos de programación de Software Libre en un repositorio *público*; varias compañías de software suelen reclutar programadores únicamente de ver sus perfiles en portales como GitHub y GitLab.

Para mantener respaldos de nuestros repositorios locales de manera remota es necesario contar con una cuenta en GitHub o GitLab. En el caso de Github puedes hacerlo en el sitio github.com y una vez confirmada tu cuenta te recomendamos asociar tu cuenta a GitHub Education para tener acceso a repositorios privados (recuerda que no quieres "pasarle la tarea" a tus compañeros), esto lo puedes hacer en el sitio education.github.com/students

Una vez que tengas una cuenta, te recomendamos configurar a Git desde la consola con el usuario y correo que elegiste para tu cuenta en GitHub, de esta forma todos los cambios que hagas en los repositorios locales te tendrán como autor automáticamente. Para hacerlo existen los siguientes comandos sustituyendo el nombre de usuario y el correo con los tuyos:

```
~/ $ git config --global user.name "usuario"
~/ $ git config --global user.email "usuario@ciencias.unam.mx"
```

Opcionalmente puedes asociar una clave de autenticación remota entre tu computadora personal y tu cuenta de GitHub, esto te servirá para que al subir cambios a un repositorio remoto desde tu computadora no sea necesario introducir tus credenciales (usuario y contraseña).

Primero generamos un par de llaves SSH, pulsamos Enter para generarla con valores predeterminados: (recuerda sustituir tu correo y usuario cuando sea necesario)

```
~/ $ ssh-keygen -t rsa -b 4096 -C "usuario@ciencias.unam.mx"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
b0:8d:d6:4d:87:5e:ad:75:d9:18:27:d6:83:d8:c9:8d
usuario@ciencias.unam.mx
The key's randomart image is:
+--[ RSA 4096]----+
|                 |
```

```
|          + B . . |
|       . . E * o .|
|        * + + o *o|
|       + S . . +.o|
|      .       . . |
|                  |
|                  |
|                  |
+------------------+
```

Inicializamos el agente SSH de nuestra computadora y agregamos la clave que acabamos de generar:

```
~ $ eval $(ssh-agent -s)
Agent pid 10875

~ $ ssh-add ~/.ssh/id_rsa
Identity added: /home/user/.ssh/id_rsa (/home/user/.ssh/id_rsa)
```

Verificamos que haya contenido de nuestra llave pública y lo copiamos en el portapapeles:

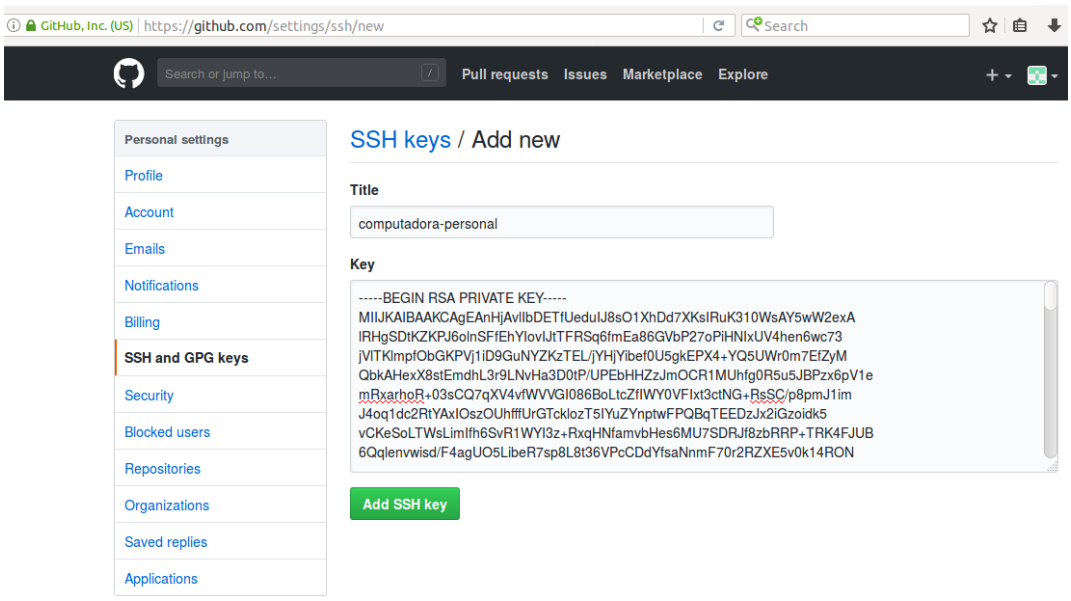
```
~ $ cat < ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDBAnXYiwntPeOpgNRuuYyn8GM+cfWpN
uoU59r1Betq46vp598eimBMlsDb00TpJbW0NX7iuNqbjj5WAYRy+pop4SPhX5+irsJKsp
V6qagDsKSKAjsfansdmuebplA0sOuBbJaMDl+bJo4iKBRaOwjmFjyOzgWzLnKqHrvWIql
J8VyIQ3bhidq8yOV4coa;fnaifhawq9daskwrozsfcjzaSFSEw9sfnlsafDFSwer93fr/
xEaan0efpzEBGKaIwDF9GU3v3tFW+uHUXMLMuNA3t2ZVk7WZNAtXisaI3ZODu2GUJigh6
dPW15ByRe+NVO6KvIrVhl85hrKu8/5aue6IcZMSpUpk0CuPhF5a/WTKeHUY8uZLzFp+iC
uPHsJ6Gr8xeMSaxSTpRKS2KOq15UghcGQ4mLiG0GRpHKHf7aFc2xQfpHmPq67oJ8Wwkxe
6plICRDIrX5/VCK8AfcOuX6nT/U+FQU+I8l76TXf+ibyATEXEEiTkb+kWhcmHthzlKbM=
user@pc
```

Vamos a la configuración de nuestra cuenta en GitHub y elegimos la opción `SSH and GPG keys` (github.com/settings/keys), seleccionamos `New SSH Key` y agregamos la llave pública que generamos en nuestra computadora:

Comprobamos la conexión con el comando `ssh -T git@github.com` y hemos terminado.

A partir de ahora la autenticación con github de nuestro usuario desde nuestra computadora será automática. Lo siguiente es ligar nuestro repositorio local `ejemplo` con uno remoto en GitHub. Tendremos que crear un repositorio desde nuestra cuenta de GitHub; lo primero es ir a GitHub, iniciar sesion y dar clic en el botón “New”, posteriormente asignar

Figura 6.1 Asignamos un título mnemónico a la llave y la agregamos.



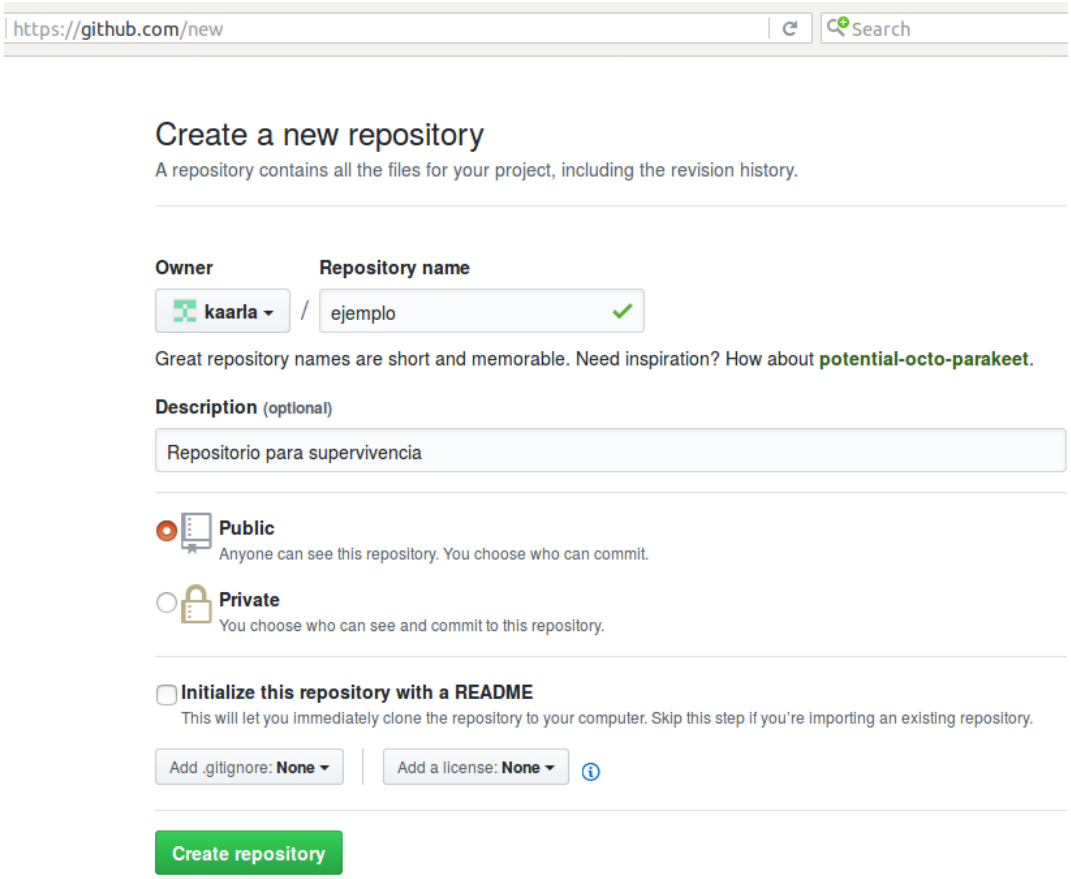
el mismo nombre de nuestro repositorio local al remoto, agregar una descripcion opcional y dar clic en crear:

Nuestro repositorio remoto por ahora esta vacío y nos da diferentes opciones para empezar a trabajar, nosotros eligiremos la segunda enlistada en la pantalla que corresponde a ligarlo a un repositorio local existente por medio de SSH, en caso de no haber configurado tu llave selecciona HTTP y la única variación será que tendrás que introducir tus credenciales (Figura 6.3).

Volvemos al repositorio sobre el que trabajamos en la sección anterior y usamos el comando `remote add` con la convención "origin" que suele usarse para asignar el repositorio remoto. En seguida usaremos el comando `push` que sirve para "empujar" los cambios de la rama actual a su compia remota:

```
~/ejemplo $ git remote add origin git@github.com:usuario/ejemplo.git
~/ejemplo $ git push -u origin master
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 611 bytes | 0 bytes/s, done.
Total 7 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/usuario/ejemplo.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

Figura 6.2 Personalizamos la creación del nuevo repositorio.



Actualizamos la página de configuración de nuestro repositorio y tendremos una copia del local en su estado actual, así se ve en master el documento que hemos trabajado localmente (Figura 6.4).

Si estuviéramos trabajando en equipo y alguien mas actualizara la rama en la que queremos trabajar, master por ejemplo, usaríamos el comando `git pull origin master` desde nuestro repositorio local y automaticamente bajariamos el estado de la rama remota; sólo en caso de haber editado las mismas líneas sería necesario solucionar los conflictos como vimos previamente.

Figura 6.3 Opciones para inicializar un repositorio.

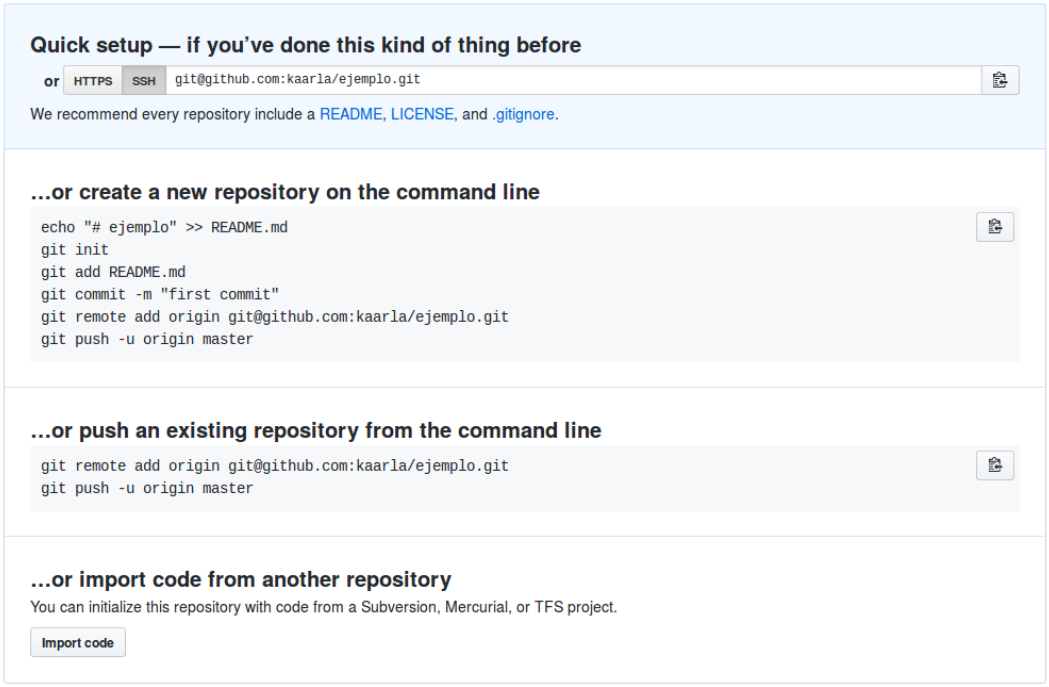
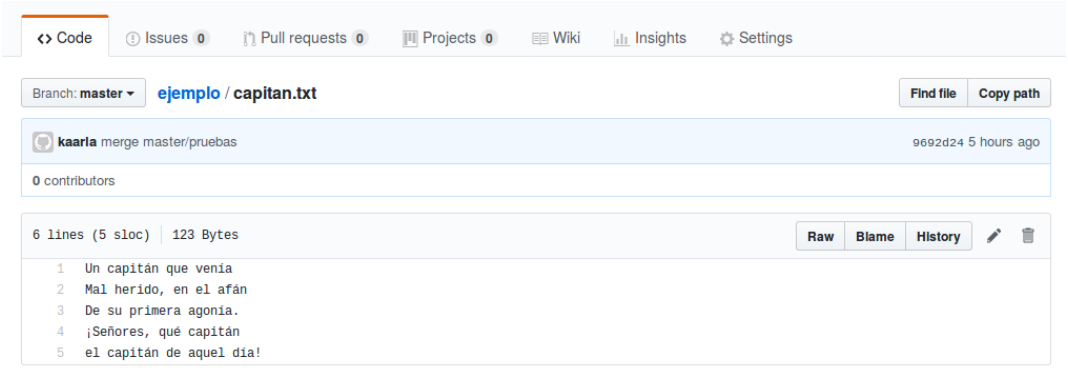


Figura 6.4 Vista final del contenido del archivo en línea.



LaTeX 7

7.1. 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.

7.2. 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

[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.

A partir de las subsubsecciones puedes poner títulos y subtítulos.

7.3. 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 7.1 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 7.1 *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 7.2 *Carga en tu Emacs el archivo template1.tex que se encuentra en el subdirectorio indicado en la página 3.1.*

Actividad 7.3 *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.*

7.3.1. 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:

1. 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, caracteres 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.

7.4. 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 7.2.

Tabla 7.2 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 7.2 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 7.4 *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 7.5 *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 7.6 *Cambia el tipo de documento a* article. *Vuelve a compilar y reporta lo que observas, en cuanto al formato de la hoja.*

Actividad 7.7 *Haz lo mismo que en la actividad anterior, pero con la clase* book.

7.4.1. 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 7.8 *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.

7.5. 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 7.3.

Tabla 7.3 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 7.9 *Vuelve a cargar el archivo* CalcProp.tex *para que vuelvas a tener la versión original.*

Actividad 7.10 *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 7.11 *Quita las llaves que delimitan el texto. Compila y ve el resultado.*

Actividad 7.12 *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 7.4 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 7.13 *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 7.14 *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.

7.6. 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.

7.6.1. 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.

7.6.2. 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.

7.6.3. 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 7.15 *Crea un archivo con clase de documento* article *y que use los mismos paquetes que usa* CalcProp.tex.

Actividad 7.16 *Ponle como título tu nombre y el número de actividad.*

Actividad 7.17 *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 7.18 *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.*

7.6.4. 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.

7.7. 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 7.5 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

p

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 7.19 *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 7.20 *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.

7.7.1. 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 caracteres 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 7.21 *Repite la tabla que construiste y modifícala para agregarle el título y el último renglón.*

7.7.2. 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 caracteres 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 7.22 *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*		

7.7.3. 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.

7.7.4. 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 7.23 *Codifica la siguiente tabla:*

Bonetería Científica, S. A.		
Año	*Precio*	*Comentarios*
	bajo *alto*	
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.*

7.8. 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 caracteres especiales. En general, deseas poder escribir caracteres especiales y, en particular, fórmulas matemáticas.

Para algunos caracteres 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 caracteres 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 7.6 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 7.7 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 7.8 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 7.24 *Crea un archivo nuevo copiando el preámbulo que hemos visto hasta ahora.*

Actividad 7.25 *Construye una tabla de cinco columnas en las que veas el resultado de aplicar seis de los tipos de acentos a las cinco vocales.*

7.8.1. 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^2y

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 7.26 *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 7.27 *Codifica en* LaTeX *las siguientes expresiones:*

$$P_i log_2 P_i$$

$$(x_{i_1} - y_{i_1}) + (x_{i_2}^2 - y_{i_2}^2) + (x_{i_3}^3 - y_{i_3}^3)$$

$$(x_1^2 + y_1^2)^{1/2}$$

$$(X_{max} - X_{min})^2 / (Y_{max} - Y_{min})^2$$

7.9. 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 7.12.1, en la página 178, 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 7.12.1 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 7.9 se muestran las letras griegas disponibles:

Tabla 7.9 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 7.28 *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 7.10.

Tabla 7.10 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 7.11 te mostramos símbolos comúnmente usados en matemáticas como operadores y símbolos relacionales.

Tabla 7.11 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 7.11 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 7.29 *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 7.30 *Codifica en* LaTeX *las siguientes fórmulas:*

$$p \wedge 1 \equiv \neg p \Rightarrow q$$

$$(p \wedge q) \Leftrightarrow \neg(\neg p \vee \neg q)$$

$$\parallel X - Y_j \parallel^2 \leq \parallel X - Y_i \parallel^2, Y_i \in C$$

$$d(x_i, y_i) = x_i \oplus y_j$$

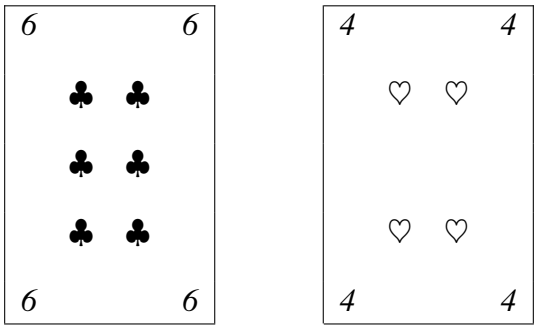
Símbolos varios

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 7.12 ponemos algunos símbolos más que te van a ser de utilidad.

Tabla 7.12 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 7.31 *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 7.13.

Tabla 7.13 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	$\inf$
\ liminf	$\liminf$	\max	$\max$	\sin	$\sin$	\tanh	$\tanh$
a \mod b	$a \bmod b$			\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:

```
$\frac{3}{4}$      \[\ frac {3}{4}\]
```

$\frac{3}{4}$ \[\ frac {3}{4}\] $$\frac{3}{4}$$

```
$\frac{1}{n+1}$    \[\ frac{1}{n+1}\]
```

$\frac{1}{n+1}$ \[\ frac{1}{n+1}\] $$\frac{1}{n+1}$$

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 7.32 *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 7.33 *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 7.14 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 7.14 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 7.34 *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:

$$\left\{\left[\left(\left|prueba\right|\right)\right]\right\}$$

La lista de los símbolos que puedes usar como delimitadores y que se agrandan se encuentra en la tabla 7.15.

Tabla 7.15 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 7.35 *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 7.36 *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 7.16.

Tabla 7.16 Tipos de letras en modo matemático

`$\mathcal{ABCDEF...}$`	$\mathcal{ABCDEF...}$
`$\mathit{ it \acute{a}licas \cdot 2^{ ft }\Psi \log [\psi ]}$`	$\mathit{it\acute{a}licas \cdot 2^{ft}\Psi \log[\psi]}$
`$\mathrm{romanas \cdot 2^{ft}\Psi \log [\psi ]}$`	$\mathrm{romanas \cdot 2^{ft}\Psi \log[\psi]}$

Continúa en la siguiente página

Tabla 7.16 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 7.37 *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 caracteres 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 7.38 *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 7.39 *Codifica cada una de las matrices anteriores usando* array *y delimitadores.*

Actividad 7.40 *Codifica la siguiente fórmula:*

$$\begin{pmatrix} n! \\ r!(n-r)! \end{pmatrix}$$

7.9.1. 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 7.17.

Tabla 7.17 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 7.41 *Codifica en* LaTeX *la siguiente expresión:*

$$A\cap B=\begin{cases}A & \textit{si } A\subseteq B\\ B & \textit{si } B\subseteq A\\ \emptyset & \textit{si } A \textit{ y } B \textit{ son ajenos}\\ \overline{\bar{A}\cup\bar{B}} & \textit{por la ley de De Morgan}\end{cases}$$

7.9.2. 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}
```

$$R=\frac{1}{M}\sum_{k=1}^{M}R_k \qquad (7.1)$$

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.

```
\[\begin{array}[t]{ll}
    x_1=\sum_{k=1}^n a_ky^k
     &\text{primer valor}\\[10pt]
    x_2=\sum_{k=1}^n b_ky^k
     &\text{segundo valor}
   \end{array}
\]
```

$$\begin{array}[t]{ll} x_1=\sum_{k=1}^n a_ky^k & \text{primer valor}\\[10pt] x_2=\sum_{k=1}^n b_ky^k & \text{segundo valor}\end{array}$$

[3]Consulta alguno de los manuales de LaTeX que se mencionan en la bibliografía si así lo deseas.

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}
```

$$x_1 = \sum_{k=1}^{n} a_k y^k \quad \text{Primer valor} \tag{7.2}$$

$$x_2 = \sum_{k=1}^{n} b_k y^k \quad \text{segundo valor} \tag{7.3}$$

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*}
```

$$x_1 = \sum_{k=1}^{n} a_k y^k \quad \text{Primer valor}$$

$$x_2 = \sum_{k=1}^{n} b_k y^k \quad \text{segundo valor}$$

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}
```

$$\begin{bmatrix}\Theta_1\\ \Theta_2\end{bmatrix} + \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\alpha\\ \alpha\end{bmatrix} - \begin{bmatrix}\sqrt{2}\alpha\\ 0\end{bmatrix} \tag{7.4}$$

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}
```

$$\begin{bmatrix}\Theta_1\\ \Theta_2\end{bmatrix} + \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}\alpha\\ \alpha\end{bmatrix} \tag{7.5}$$

$$- \begin{bmatrix}\sqrt{2}\alpha\\ 0\end{bmatrix} \tag{7.6}$$

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 7.18 vemos los distintos tipos de espacios que puedes tener. Están divididos entre positivos (hacia la derecha) y negativos (de retroceso).

Tabla 7.18 Comandos de espacio en modo matemático

Espacio positivo

Abr	Completo	Ejemplo		Normal
mu	\mspace{1mu}	$x\mspace{1mu}x$	xx	
\,	\thinspace	x\,x	$x\,x$	xx
\:	\medspace	x\medspace x	$x\medspace x$	xx
\;	\thickspace	x\;x	$x\;x$	xx
	\enskip	x\enskip x	$x\enskip x$	xx
	\quad	x\quad x	$x\quad x$	xx
	\qquad	x\qquad x	$x\qquad x$	xx

Espacio negativo

Abr	Completo	Ejemplo		Normal
\!	\negthinspace	x\!x	$x\!x$	xx
	\negmedspace	x\negmedspace x	$x\negmedspace x$	xx
	\negthickspace	x\negthickspace x	$x\negthickspace x$	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 7.42 *Codifica en* LaTeX *la siguiente ecuación:*

$$G_{TC} = \frac{\frac{1}{N}\sum_{i=0}^{N-1}\sigma_i^2}{\left(\prod_{i=0}^{N-1}\sigma_i^2\right)^{\frac{1}{N}}} \tag{7.7}$$

Actividad 7.43 *Codifica en* LaTeX *la siguiente ecuación:*

$$s_n \neq s_{n-1} = s_{n-2} \quad M_1 = 0.4 \tag{7.8}$$
$$s_n \neq s_{n-1} \neq s_{n-2} \quad M_2 = 0.9 \tag{7.9}$$
$$s_n = s_{n-1} \neq s_{n-2} \quad M_3 = 1.5 \tag{7.10}$$
$$s_n = s_{n-1} = s_{n-2} \quad M_4 = 2.0 \tag{7.11}$$

Actividad 7.44 *Codifica en* LaTeX *la siguiente ecuación:*

$$\begin{aligned} \begin{bmatrix} x_{01} & x_{01} \\ x_{10} & x_{11} \end{bmatrix} &= \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} \theta_{00} & \theta_{01} \\ \theta_{10} & \theta_{11} \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} \theta_{00}+\theta_{01}+\theta 10+\theta_{11} & \theta_{00}-\theta_{01}+\theta 10-\theta_{11} \\ \theta_{00}+\theta_{01}-\theta 10-\theta_{11} & \theta_{00}-\theta_{01}-\theta 10+\theta_{11} \end{bmatrix} \\ &= \theta_{00}\alpha_{0,0}+\theta_{01}\alpha_{0,1}+\theta_{10}\alpha_{1,0}+\theta_{11}\alpha_{1,1} \end{aligned} \tag{7.12}$$

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.

7.10. Imágenes y figuras

7.10.1. 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 7.19 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 7.19 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 7.20 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 7.21 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 7.21 Conversión de Farenheit a Centígrados

Fahr	Cent
32	0
40	4.5
80	27

La tabla 7.21 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 7.45 *Codifica en LaTeX la tabla 7.22 en la siguiente página. Pon el posicionamiento para que aparezca exactamente donde se tecleó.*

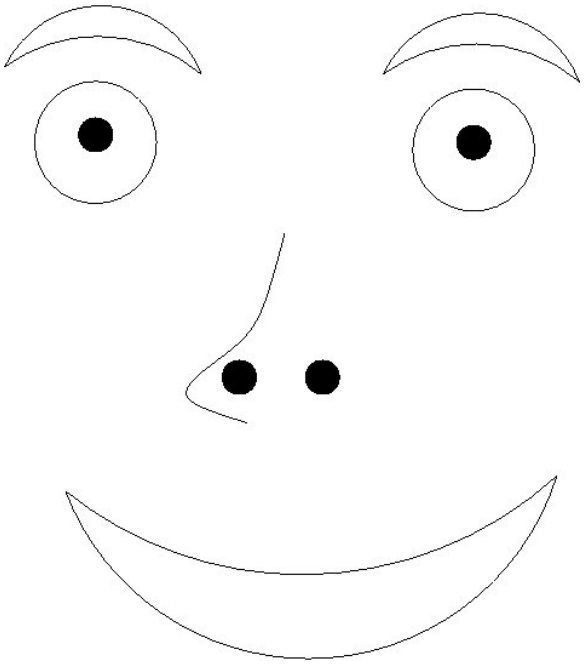
Tabla 7.22 Representaciones numéricas

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 fi-

gura:

Figura 7.1 Gráfico introducido con `includegraphics`

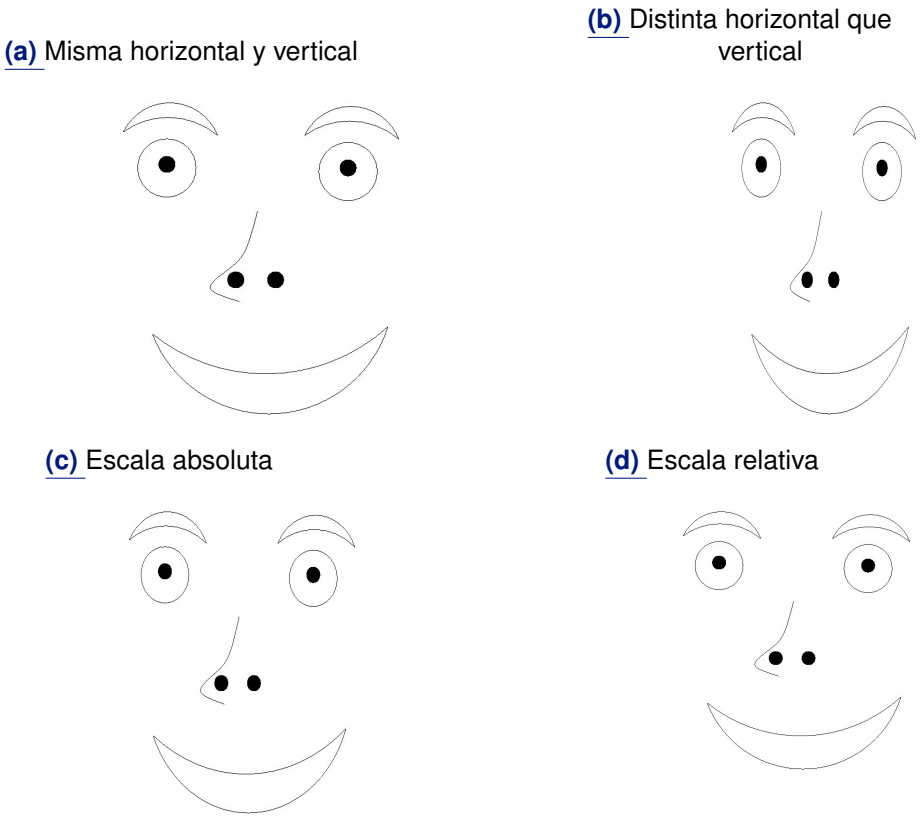


que fue incluida con el siguiente código:

```
\begin{figure}[H]
  \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 7.2a (\scalebox{.25}); o indicando la proporción en la escala horizontal entre llaves y con la escala vertical entre corchetes: `\scalebox{.2}[.3]`, como en 7.2b (\scalebox{.25}[.5]). También puedes dar medidas absolutas, como en la figura 7.2c (\resizebox{8cm}{2cm}) o bien, utilizar una escala relativa, donde omitimos el factor vertical, y se asumirá el mismo que el horizontal, por ejemplo 7.2d (\resizebox{2cm}{!}). En la escala absoluta se deben dar las dos medidas. Cualquiera de ellas puede ser sustituida por `!`.

Figura 7.2 Escalando objetos con graphics.



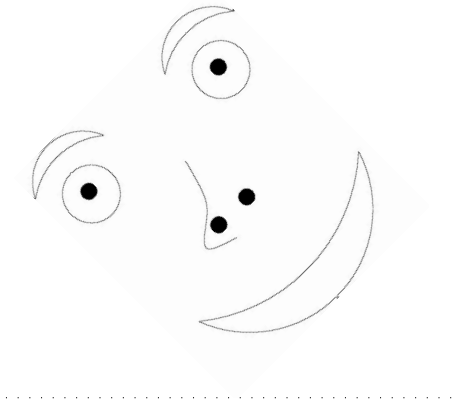
Otras dos posibilidades que tienes con objetos es la de rotarlos o "espejearlos". En la figura 7.3 puedes ver rotaciones de 45 grados 7.3a (\rotatebox{45}), de 180 7.3b (\rotatebox{180}) y una figura y su reflejo 7.3c (\reflectbox). En cada una de ellas se marca la línea base como una línea punteada, 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".

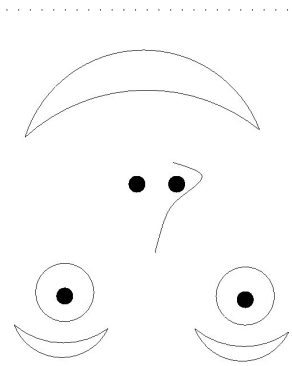
Es importante insistir en que estas operaciones de escalar, rotar y reflejar las puedes aplicar a cualquier objeto: texto, fórmulas matemáticas (siempre y cuando estén en modo matemático .. o \(.. \)), tablas completas, figuras, etc.

Figura 7.3 Rotaciones y reflejos con `graphics`

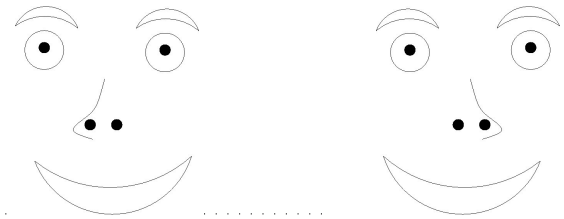
(a) Rotación de 45



(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

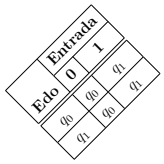
reflejo horizontal

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 7.46 *Codifica en* LaTeX *lo siguiente:*

Tabla 7.23 Comparación de tamaños

Edo	Entrada	
	0	1
q_0	q_0	q_1
q_1	q_0	q_1



Actividad 7.47 *Codifica en* LaTeX *la siguiente expresión:*

$$\alpha = b^2 - 4ac \qquad \mathfrak{A} = \mathfrak{B}_{\mathfrak{Z}} - \sqrt{\mathfrak{A}\mathfrak{C}}$$

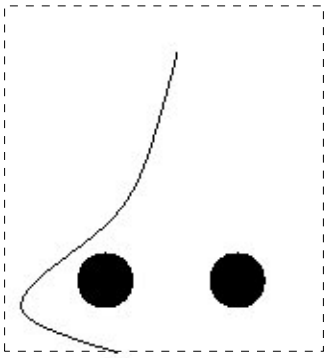
Actividad 7.48 *Importa a un archivo de* LaTeX *que use el paquete* graphics *alguna de las gráficas que elaboraste con* xfig *(la del árbol genealógico, por ejemplo). La debes tener guardada en tu cuenta, por lo que si no la exportaste, abre otra ventana y ejecuta* xfig *para exportarla en postscript.*

7.10.2. 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 7.4.

Figura 7.4 Tomando pedazos de figuras

```
includegraphics*[100pt,130pt][220pt,260pt]\{dummy.ps\}
```



En la figura 7.4 colocamos un marco alrededor del "recorte" para hacerlo más claro. Ese marco no aparece como resultado del comando.

7.10.3. 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 7.24 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 7.24 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 7.25.

[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 7.25 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 7.49 *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).*

7.10.4. 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.

7.10.5. 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:

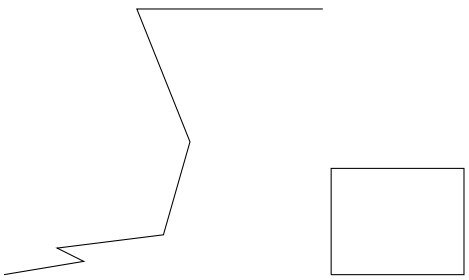
```
\tikz [x=1pt,y=1pt]\draw(0,0) -- (20,30) -- (30,0)
```
. Nota cómo el renglón se acomoda verticalmente para dar lugar al trazo y el texto continúa a partir del final del espacio ocupado. Si deseamos que el texto se acomode encima del dibujo, tenemos que hacerlo a pie recorriéndonos hacia arriba y 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:

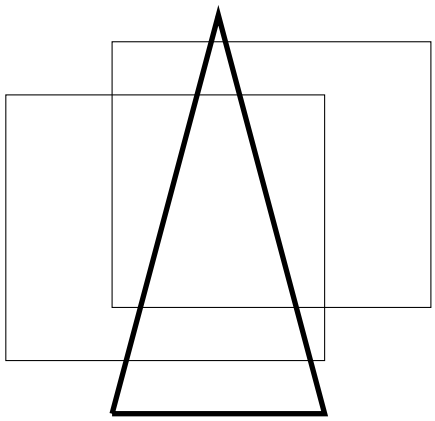
```
\begin{tikzpicture}[x=1pt,y=1pt,%
                    line width=2pt]
  \draw (0,0) -- (30,20) -- (30,20)%
              -- (50,0) -- (50,0)-- (70,20)%
              -- (70,20)-- (90,0) -- (90,0)%
              -- (110,20);
\end{tikzpicture}
```

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}
```

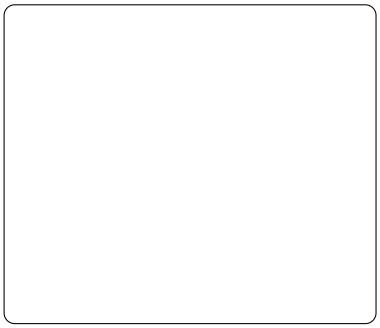
Siempre que introduces un ambiente tikz, debes pensar que reservas un espacio rectangular dado por lo que dibujaste. Ese espacio logra contener todos los objetos que hayas acomodado o dibujado en ese ambiente. El (0,0) es la esquina inferior izquierda (aunque esto lo puedes modificar) y la esquina superior derecha está dada por lo que hayas hecho dentro del ambiente.

Actividad 7.50 *Escribe el código necesario para que se dibuje lo que sigue (el tamaño relativo es lo que importa).*

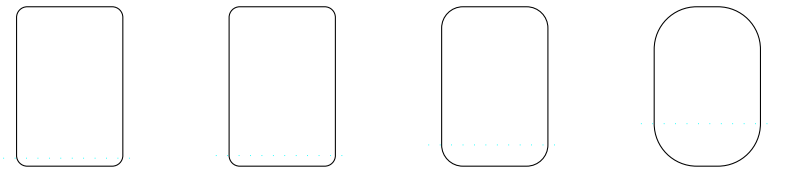


Como ya mencionamos, tikz provee varias figuras geométricas con las que puedes trabajar usando la biblioteca shapes de este paquete. Y como las figuras son cierto tipo de objetos, puedes acomodarlos también en la retícula mediante draw. Además del rectángulo, contamos con círculos (*circle*), diamantes (*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.

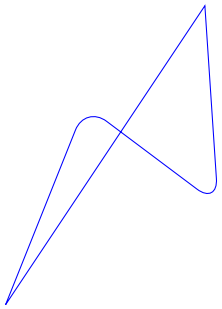


que se consiguió con el siguiente código:

```
\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}
```

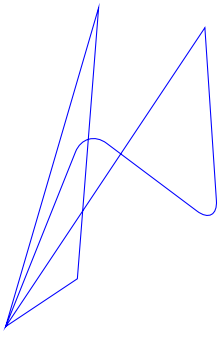
Otros de los modificadores que puedes poner a \draw son el color de las mismas – blue, red, yellow, cyan, green, magenta –; el tipo de línea – [loosely|densely]{solid|dotted|dashed} –. Estos últimos parámetros no llevan signo = o identificador cuando aparecen en el contexto de un comando \draw y puedes 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 7.26, <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 7.26 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 7.27.	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 7.5 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 7.27).	solid

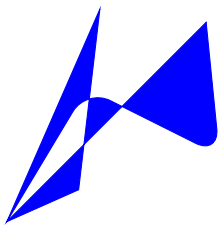
[8]Para ver los tipos de patrones disponibles, consultar el manual de TikZ.

Tabla 7.27 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 7.5 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 7.28.

Tabla 7.28 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 7.29.

Tabla 7.29 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 7.30 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 7.30.

Tabla 7.30 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}
```

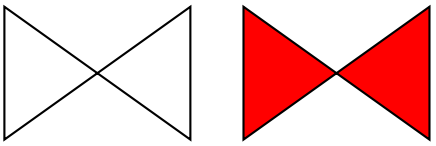
En el manual de TikZ viene una lista de patrones interesantes que puedes desear usar.

En las tablas 7.31 y 7.32 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 7.31 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}
```



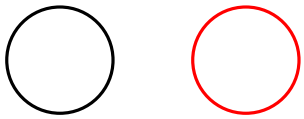
```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0) rectangle (170,70);
  \draw[fill] (10,10) rectangle (80,60);
  \draw[fill=white] (20,20) rectangle (40,40);
\end{tikzpicture}
```

Continúa en la página siguiente

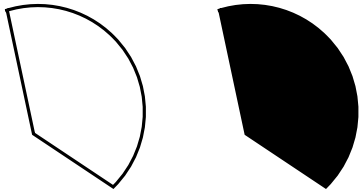
Tabla 7.31 Figuras geométricas que se pueden lograr en TikZ

Continúa de la página anterior

```
\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}
```

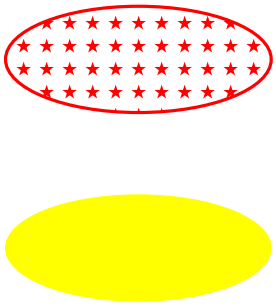


```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0)
                    rectangle (140,70);
  \draw[very thick] (20,38) -- (10,0)
       -- (50,0) arc (0:76:40pt);
  \draw[fill] (100,38) -- (90,0)
       -- (130,0) arc (0:76:40pt);
\end{tikzpicture}
```



```
\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}
```

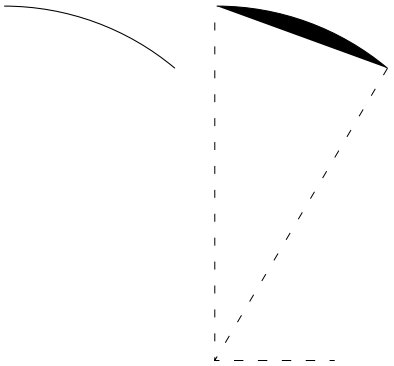


Continúa en la página siguiente

Tabla 7.31 Figuras geométricas que se pueden lograr en TikZ

Continúa de la página anterior

```
\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 7.32.

Tabla 7.32 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}
```

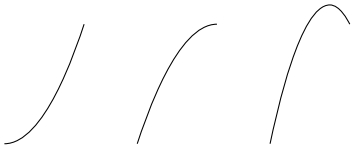
```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (0,0)
                 rectangle (100,50);
  \draw (0,0)
        parabola[parabola height=30pt]
                                 (30,0);
  \draw(50,30)
        parabola[parabola height=-30pt]
                                 (90,30);
\end{tikzpicture}
```

Continúa en la siguiente página

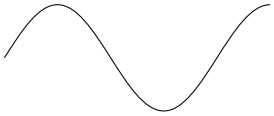
Tabla 7.32 Graficación de funciones comunes en TikZ *Continúa de la página anterior*

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}
```



```
\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}
```



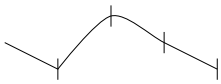
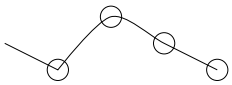
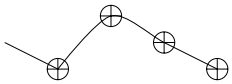
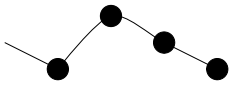
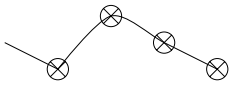
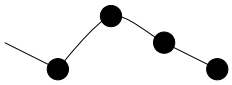
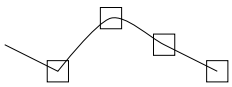
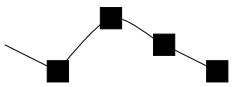
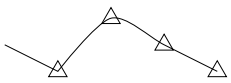
Tanto el seno como el coseno únicamente están definidos para el intervalo $(0, \frac{\pi}{2})$, por lo que es necesario combinar ambas funciones para pintar la trayectoria completa

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}
```

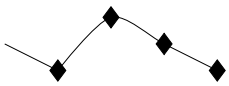
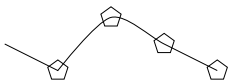
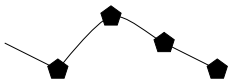
Una vez cargada la biblioteca de marcas, puedes elegir entre las que se encuentran en la tabla 7.33.

Tabla 7.33 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
	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

Continúa en la página siguiente

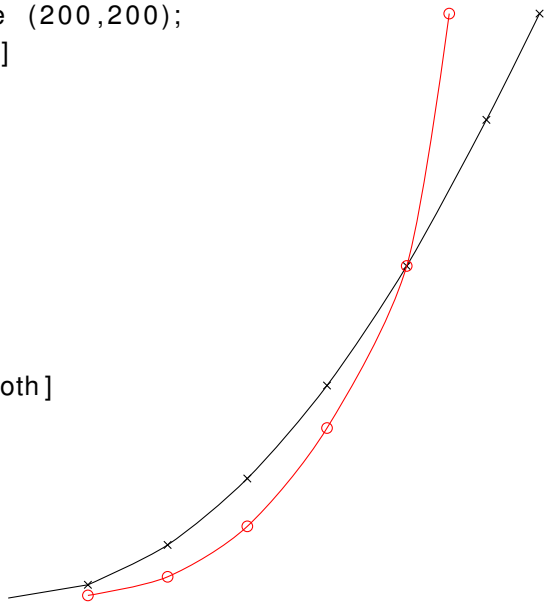
Tabla 7.33 Símbolos para marcar curvas que se grafican *Continúa de la página anterior*

Marca	Identificación	Descripción
	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 7.6 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 7.6 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 7.7.
  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 7.8.
- 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 7.9 y después analizaremos su formato.

Figura 7.7 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}
```

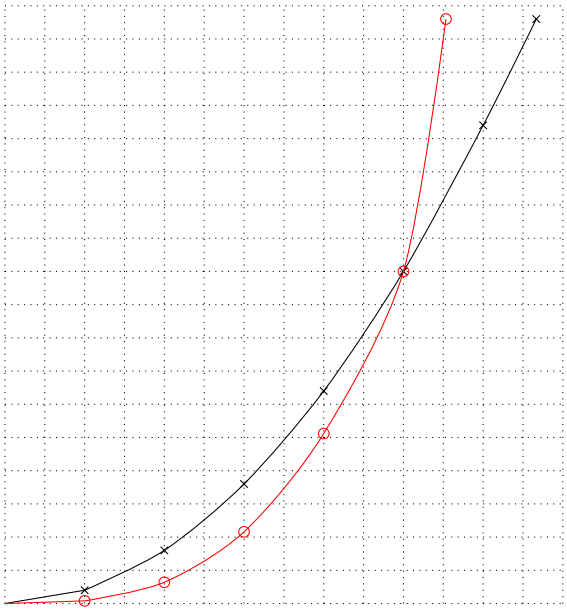
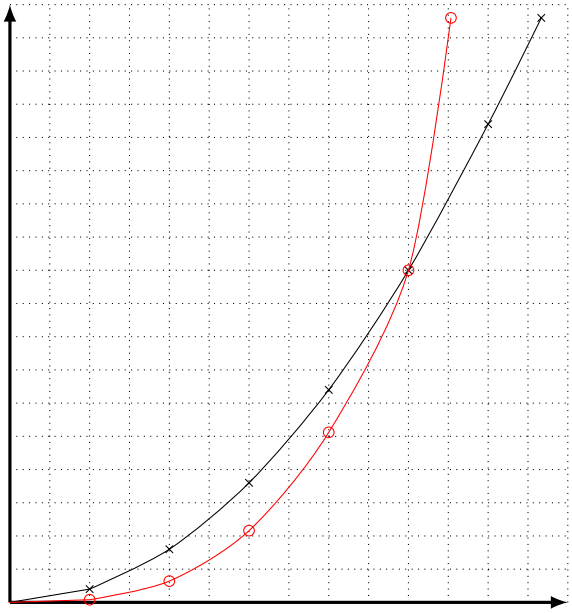


Figura 7.8 Colocación de los ejes

```
\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}
```

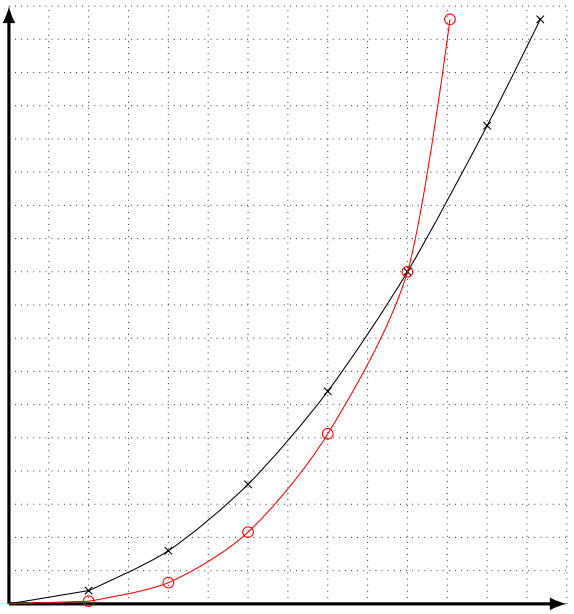


El enunciado \foreach tiene las siguientes partes:

- 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 7.9 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:

$$\langle \textit{primer valor} \rangle \ , \ \langle \textit{segundo valor} \rangle \ , \ \ldots \ , \ \langle \textit{último valor} \rangle$$

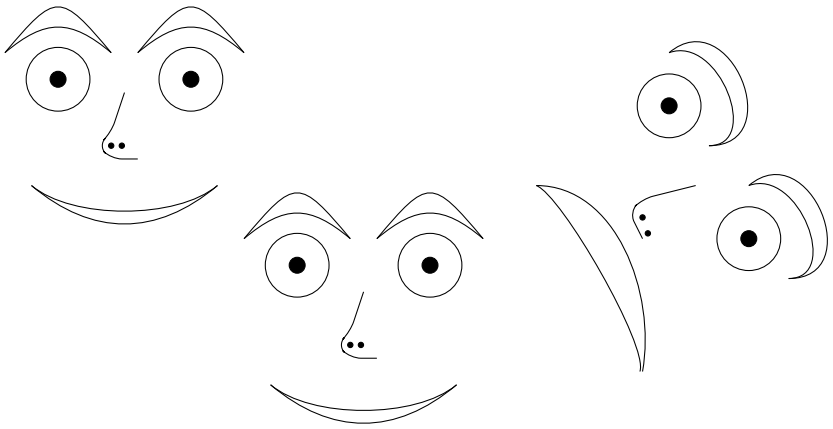
Los dos \foreach que tenemos los podríamos haber escrito de las formas que se muestran en el listado 7.1.

Listado 7.1 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 7.10 que usa únicamente los conceptos que hemos dado hasta ahora.

Figura 7.10 Divertimento con sólo trazo de líneas



Los comandos de TikZ que logran el dibujo anterior se encuentran en el listado 7.2.

Listado 7.2 Dibujos arbitrarios (1/2)

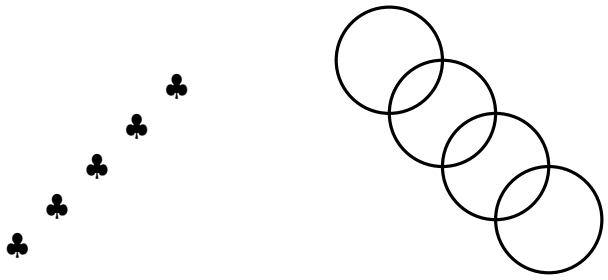
```
\begin{tikzpicture}[x=1pt,y=1pt]
  \useasboundingbox (30,30) rectangle (350,180);

  %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
```

Listado 7.2 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 7.51 *Escribe los comandos necesarios en TikZ para dibujar lo siguiente:*



7.10.6. 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:

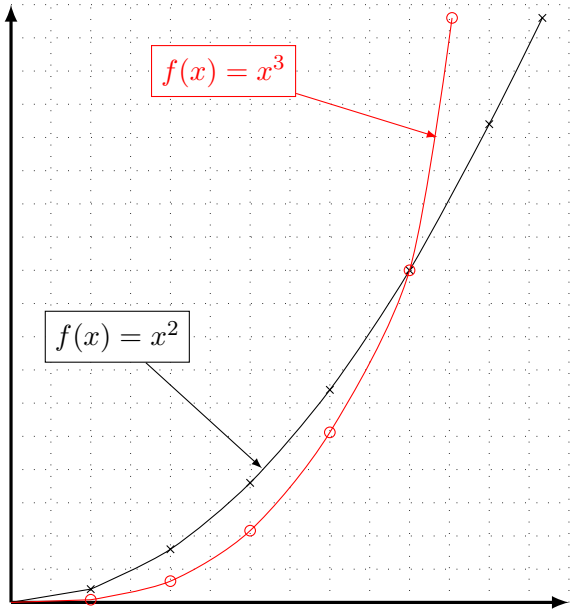
- Ocupa una cierta posición, dada por una pareja (x,y).
- Tiene alguna forma que puede ser circle, rectangle, diamond, ellipse.
- Puede tener un identificador asociado.
- Puede dibujarse o no su contorno con cualquier tipo de línea.
- Puede rellenarse o no con distintos patrones y en colores diversos.

El formato general para colocar un nodo es el siguiente:

\node (⟨*etiqueta*⟩) [⟨*modificadores*⟩] at (⟨*posición*⟩) {⟨*contenido*⟩};

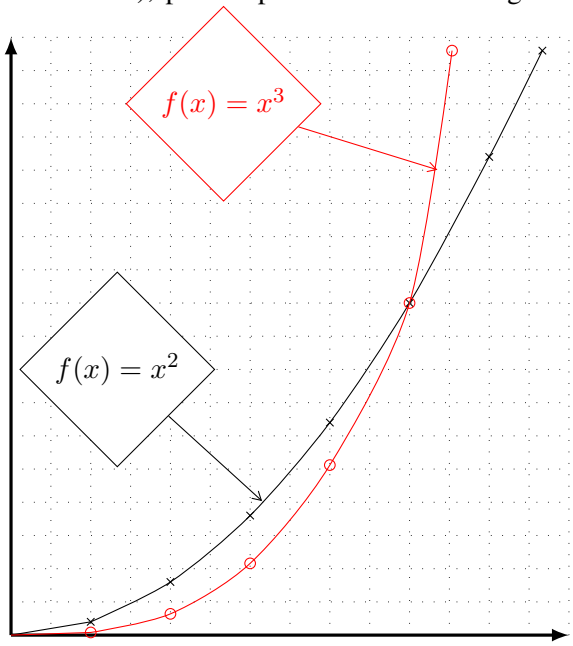
Por ejemplo, puedes marcar algunas posiciones en el espacio en el que graficas las funciones para colocar allí una descripción de las mismas.

```
\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.

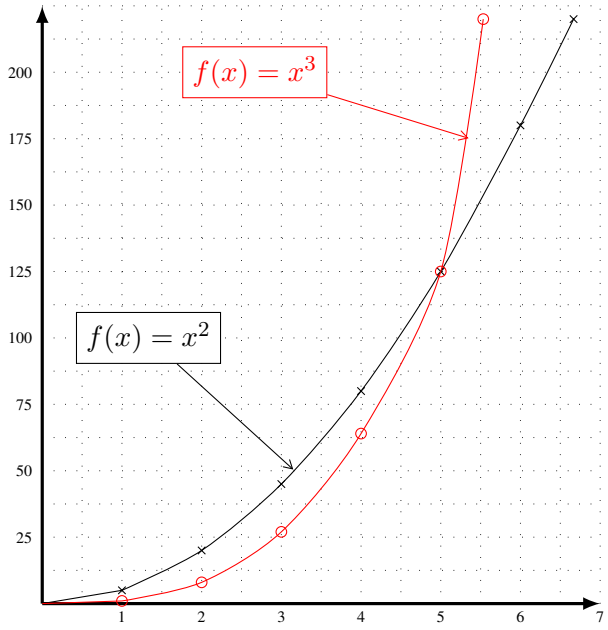
- La forma (*shape*) puede ser cualquiera de las que se encuentran en la biblioteca shapes de TikZ, que se carga mediante el comando

  ```
  \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.

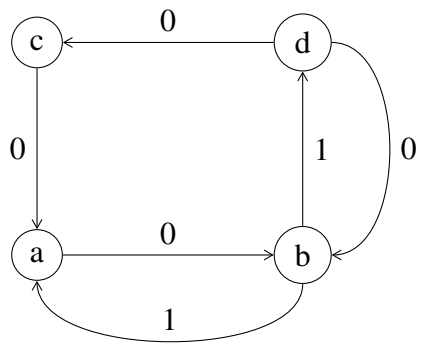
Al colocar las etiquetas en el eje de las y utilizamos un nuevo modificador en el nodo – nota que no utilizamos nombre para el nodo pues no es necesario – que acomoda al contenido del nodo respecto a la posición del mismo. En este caso deseamos que los números se alineen a la derecha, por lo que indicamos 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 7.11 que se hizo con el código del listado 7.3.

Figura 7.11 Dibujo de una gráfica en LaTeX



Listado 7.3 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.

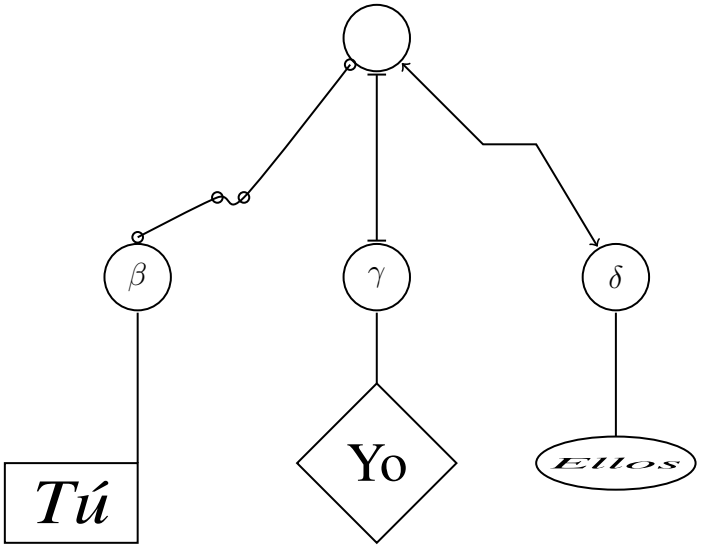
Este tipo de dibujos pueden alcanzar muchísima complejidad en TikZ, por lo que no seguimos con el tema. Recomendamos nuevamente al lector interesado que consulte el manual de TikZ.

Actividad 7.52 *Codifica los dibujos de la figura 7.12 en LaTeX.*

Actividad 7.53 *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 7.12 Actividad 7.52



7.11. 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.

7.11.1. 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 7.13 y 7.15 puedes ver estos documentos.

Figura 7.13 Ejemplo de LATEX usando BIBTEX (1/2)

```
\documentclass{article}
\usepackage[latin1]{inputenc}
\pagestyle{empty}
\begin{document}
\section{Ejemplo de citas}
```

Figura 7.13 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 7.14); 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 7.16, donde lo hemos puesto usando tanto el estilo `plain` como el estilo `acm`.

Figura 7.14 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 7.15 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 7.16 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 Amsterdan, 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 Amsterdan, 1996.

[6] Wadler, P. How to add laziness to a strict language without even being odd.

7.11.2. Índices

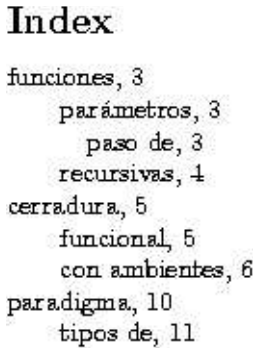
Para encontrar un tema de interés en un documento grande, el lector tiene dos alternativas, usar la tabla de contenidos o el índice. En general, el índice es la forma de acceso más común a un documento técnico y, por tanto, una parte esencial del documento.

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

7.12. 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 5.2, 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.

7.12.1. 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

$ yum 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 5.2.18). 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 7.34 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 subpar-

tes 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 caracteres 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.

7.12.2. 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.

7.12.3. 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:

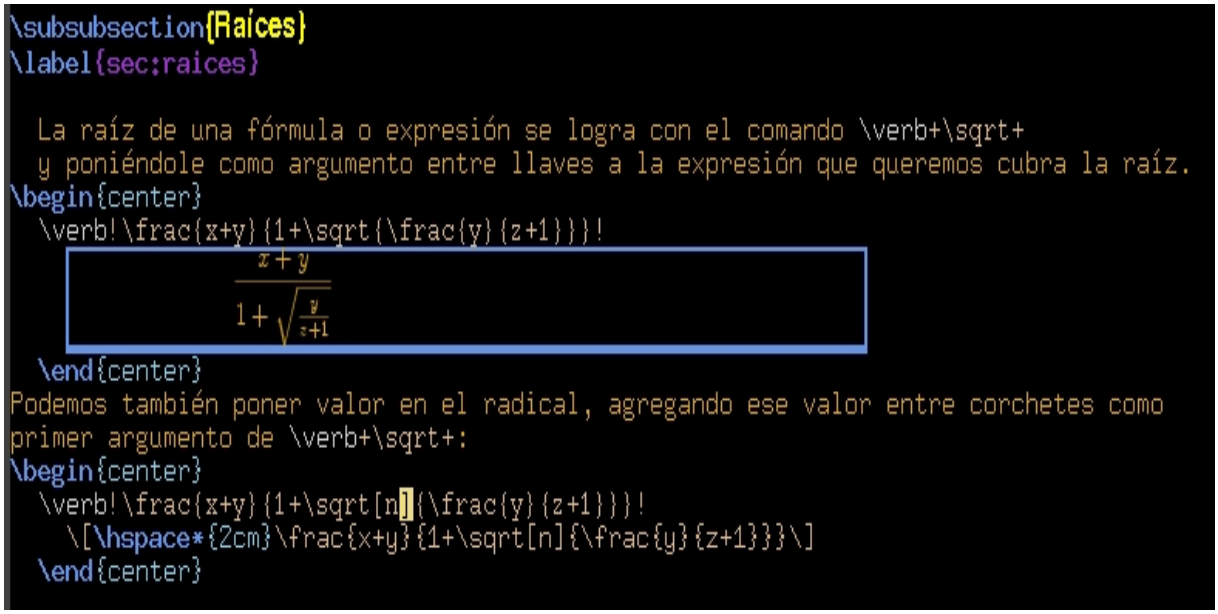
```
(load "preview-latex.el" nil t t)
```

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 7.17 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

8

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.

8.1. 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 8.1

Listado 8.1 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>
```

Listado 8.1 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 caracteres 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 caracteres 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 caracteres 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
&	&
>	>
<	<
'	"
"	'

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 seríá como se ve en el ejemplo del listado 8.2.

Listado 8.2 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.

8.2. 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.

8.2.1. 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 8.3.

Una vez más, podríamos utilizar directamente "é" en lugar de é, porque el documento utiliza el conjunto de caracteres 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.

Listado 8.3 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.

8.3. 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 8.4.

Listado 8.4 Hoja de estilo en cascada (1/2)

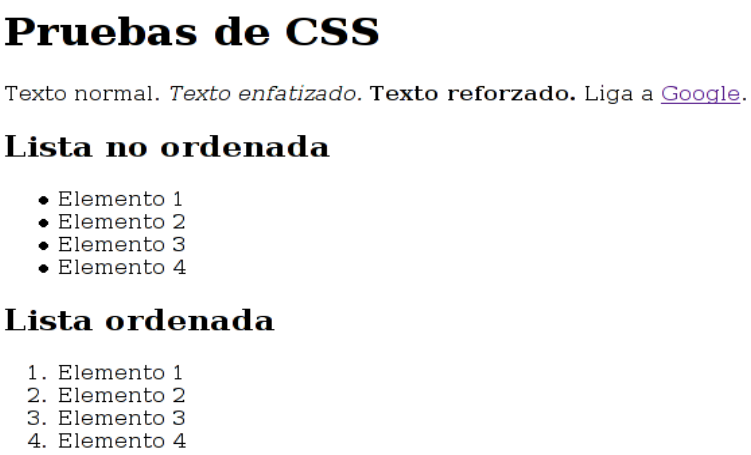
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
 <head>
  <title>Pruebas de CSS</title>
 </head>
```

Listado 8.4 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 8.1.

Figura 8.1 Página sin hoja de estilo



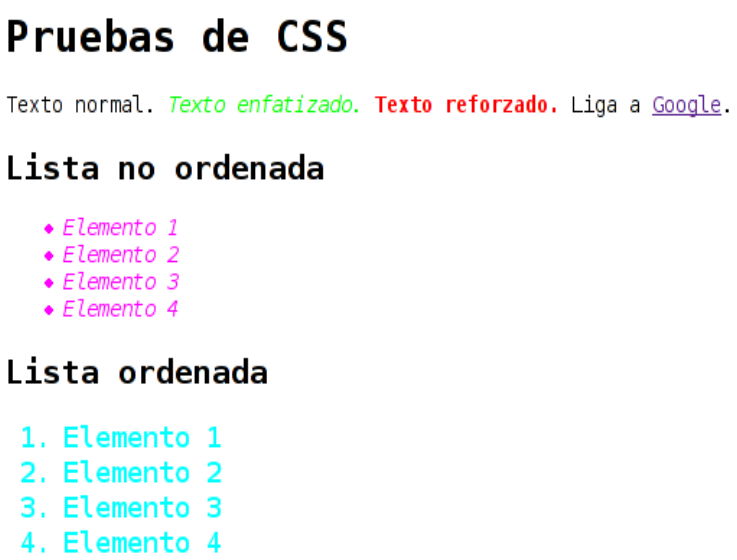
Sin modificar en *nada* el contenido del documento, podemos cambiar por completo cómo se ve, usando la hoja de estilo del listado 8.5.

Listado 8.5 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 8.2.

Figura 8.2 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:

Listado 8.6 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).

8.3.1. 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 8.7,

Listado 8.7 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 8.8.

Listado 8.8 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).

8.4. 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.

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. Fedora

A.1.1. 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 direcciones: 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 pro-

yecto 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.1.2. 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[1]. Para resolver las dependencias que puede tener un paquete, e instalarlas automáticamente, se hace uso de la herramienta yum[2]. 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
```

Con lo cual yum te mostrara cuales son los paquetes que va a instalar, sus dependencias y cuanto espacio van a ocupar.

[1] Redhat Package Manager
[2] Yellow dog Update Manager

A.2. Ubuntu

A.2.1. 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.3), 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.2.2. 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[3] (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
```

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.
```

[3]Entre las distintas secciones están: main universe multiverse restricted

```
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.3. Debian

A.3.1. 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[4], 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.3.2. 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.2.

[4] dando esto como resultado varias "guerras santas"

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 Nautilus. 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 "LC_CTYPE" "en_US.utf-8")
(require 'psvn)
```

Bibliografía

[1] AUTORES, D. The tex catalogue on line. http://www.tex.ac.uk/tex-archive/help/Catalogue/catalogue.html. Contiene muchísimos paquetes con su documentación, listos para ser bajados.

[2] CHACON, S., AND STRAUB, B. Pro git. https://git-scm.com/book/en/v2. Hay una traducción parcial al español, pero recomandamos leerlo en inglés.

[3] ETTRICH, M. Kde. http://www.kde.org.

[4] Github. https://github.com.

[5] Gitlab. https://gitlab.com.

[6] HILBRICH, T. Emacs: Dictionary. http://www.myrkr.in-berlin.de/dictionary/.

[7] ITO, A. Emacs: w3m. http://w3m.sourceforge.net.

[8] KINNUCAN, P. Emacs: Jdee - java development environment for emacs. http://jdee.sunsite.dk/.

[9] LUDLAM, E. Emacs: Cedet - collection of emacs development environment tools. http://cedet.sourceforge.net/.

[10] MITTELBACH, F., AND GOOSENS, M. *The LaTeX Companion, Second Edition*. Addison-Wesley, 2004.

[11] REICHOER, S. Emacs: psvn. http://www.xsteve.at/prg/vc_svn/.

[12] STALLMAN, R. Emacs, gcc y gdb. http://www.gnu.org. Presidente de la Free Software Foundation. Fundador GNU.

[13] TORVALDS, L. El sistema operativo linux. http://www.linux.org.

[14] TSUCHIYA, M. Emacs: emacs-w3m. http://emacs-w3m.namazu.org.

[15] WALLIN, I., AND CEDEQVIST, P. Emacs: Elib - emacs lisp library. http://elib.sourceforge.net.

[16] WANG, P. S. *An Introduction to Unix with X and the Internet*. PWS Publishing Company, 1997.