

G.L.U.P. – Guía de Linux Para el Usuario

Copyright © 1993, 1994, 1996 Larry Greenfield

Todo lo que Ud. necesita saber para comenzar a utilizar Linux, el clon gratuito de Unix. Este manual cubre los comandos básicos de Unix, y también los específicos de Linux. El destinatario de este manual es el usuario principiante de Unix, aunque puede ser de utilidad como referencia para los usuarios más experimentados.

Esta página se ha dejado en blanco intencionalmente.

UNIX es marca registrada de X/Open.

MS-DOS y Microsoft Windows son marcas registradas de Microsoft Corporation.

OS/2 y Operating System/2 son marcas registradas de IBM

X Window System es una marca registrada del X Consortium, Inc.

Motif es una marca registrada de la Open Software Foundation.

Linux no es marca registrada, ni tiene ninguna conexión con UNIX o con los UNIX System Laboratories, o con X/Open.

Por favor, ponga en conocimiento del autor cualquier marca registrada que no se haya mencionado.

Copyright © Larry Greenfield

427 Harrison Avenue

Highland Park, NJ

08904

`leg+@andrew.cmu.edu`

Traducción al castellano 1997 Grupo LuCAS.

Ver apéndice LuCAS para una relación completa.

Se garantiza el permiso para realizar y distribuir copias literales de este manual, siempre que se preserven la nota de derechos de autor y este permiso en todas las copias.

Se garantiza el permiso para copiar y distribuir versiones modificadas de este manual bajo las condiciones de las copias literales, siempre que las secciones en las cuales se reimprime “La Licencia Pública General GNU”, “La Licencia Pública General de Biblioteca GNU”, y otras en las cuales haya partes claramente marcadas bajo un derecho de autor separado, se reproduzcan bajo las mismas condiciones que ellas estipulan, y se logre que el trabajo derivado resultante en su totalidad se distribuya bajo los términos de una notificación de permiso idéntica a esta misma.

Se garantiza el permiso de copiar y distribuir traducciones de este manual a otros idiomas bajo las condiciones dadas para versiones modificadas. “La Licencia Pública General GNU” y “La Licencia Pública General de Biblioteca GNU” pueden incluirse a través de una traducción aprobada por la Free Software Foundation, en lugar de los originales en inglés.

A su opción, Ud. puede distribuir copias literales o modificadas de este documento bajo los términos de la “La Licencia Pública General GNU”, excepto las secciones marcadas claramente bajo otros derechos de autor.

Con distintos objetivos, pueden garantizarse ciertas excepciones a esas reglas. Escriba a Larry Greenfield a la dirección dada más arriba, o utilice correo electrónico a `leg+@andrew.cmu.edu` y consulte. Se solicita (pero no se requiere) que Ud. notifique al autor cuando se imprima a gran escala o se comercialice este documento. Las regalías y donaciones serán aceptadas y sustentarán las siguientes ediciones.

Estas son algunas de las convenciones tipográficas que se utilizan en este libro.


negrita Se utiliza para iniciar **nuevos conceptos**, **AVISOS**, y **palabras reservadas** de lenguajes.


itálica Se utiliza para *enfatizar* el texto.

inclinada Se usa para marcar **meta-variables** en el texto, especialmente en representaciones de la línea de comandos. Por ejemplo, “`ls -l foo`” donde *foo* se coloca en la posición donde “iría” un nombre de archivo, como por ejemplo **/bin/cp**.

monoespaciada Se usa para representar la interacción en la pantalla.

También se usa para los ejemplos de código fuente, sea que se trate de código “C”, guiones de shell, o lo que sea, y para mostrar archivos en general, como por ejemplo archivos de configuración. Cuando razones de claridad adicional lo exijan, estos ejemplos se enmarcarán en cuadros finos.

 Representa la tecla que debe oprimir. En general lo verá utilizado de la siguiente manera:

“Oprima  para continuar.”



El diamante en el margen, como los diamantes negros en las montañas de ski, marcan los lugares de “peligro” o “precaución”. Debe leer cuidadosamente los párrafos marcados de este modo.



Esta X en el margen indica la existencia de instrucciones especialmente destinadas a los usuarios del X Window System.



Esto indica un párrafo que contiene información especial que debe leerse con todo cuidado.

Agradecimientos

El autor quiere agradecer a las siguientes personas por su invaluable ayuda, tanto con Linux en sí como con la escritura de *The Linux User's Guide*.

Linus Torvalds por proveer el material acerca del cual se escribe este libro.

Karl Fogel me ha proporcionado una enorme ayuda con la escritura de mi documentación de Linux y además escribió la mayor parte del Capítulo 8 y el Capítulo 9. Cualquier reconocimiento hacia su labor será insuficiente.

Maurizio Codogno escribió gran parte del Capítulo 11.

David Channon escribió el apéndice sobre el vi. (Apéndice A)

Yggdrasil Computing, Inc. por su generoso (y voluntario) sostén para este manual.

Red Hat Software por su (más reciente y también voluntario) sostén.

El programa fortune por proveerme algunas de las preciosas citas con las cuales comenzar cada capítulo. Aún cuando a nadie más le gusten, a mí me encantan.

Índice General

1	Introducción	1
1.1	¿Quién debe leer este libro?	1
1.1.1	¿Qué debe hacer antes de leer este libro?	1
1.2	¿Cómo evitar leer este libro?	2
1.3	¿Cómo leer este libro?	2
1.4	Documentación de Linux	3
1.4.1	Otros libros de Linux	4
1.4.2	COMOs	4
1.4.3	¿Qué es el Proyecto de Documentación de Linux?	4
1.5	Sistemas operativos	4
2	A propósito, ¿qué es Unix?	7
2.1	La historia de Unix	7
2.2	La historia de Linux	8
2.2.1	Linux aquí y ahora	9
2.2.2	Unas pocas preguntas y respuestas	9
2.2.3	Software comercial en Linux	10
3	En el principio	11
3.1	La computadora despierta	11
3.2	Linux despierta	12
3.3	La actuación del usuario	14
3.3.1	El ingreso	14
3.3.2	Al abandonar la computadora	15
3.3.3	Como apagar la computadora	16

3.4	Mensajes del núcleo	17
4	El shell de Unix	21
4.1	Comandos Unix	21
4.1.1	Un comando Unix típico	22
4.2	Autoayuda	23
4.3	Almacenaje de la información	24
4.3.1	Miremos los directorios con ls	25
4.3.2	El directorio actual y cd	27
4.3.3	Creación y borrado de directorios	29
4.4	Información en movimiento	30
4.4.1	cp como un monje	31
4.4.2	La poda con rm	32
4.4.3	Sería interesante tener un rastrillo	33
5	El sistema de ventanas X	35
5.1	Ejecución y salida del sistema de ventanas X	35
5.1.1	Ejecución de X	35
5.1.2	Saliendo de X	35
5.2	¿Qué es el sistema de ventanas X?	36
5.3	¿Qué es esto que hay en mi pantalla?	36
5.3.1	XClock	37
5.3.2	XTerm	38
5.4	Gestores de ventanas	38
5.4.1	Cuando se crean nuevas ventanas	38
5.4.2	Foco	39
5.4.3	Moviendo ventanas	39
5.4.4	Profundidad	40
5.4.5	Iconizar	40
5.4.6	Variando el tamaño	40
5.4.7	Maximización	41
5.4.8	Menús	41
5.5	Atributos X	41

5.5.1	Geometría	41
5.5.2	Presentación	42
5.6	Características comunes	43
5.6.1	Botones	43
5.6.2	Barras de menú	44
5.6.3	Barras de desplazamiento	44
6	Trabajando con Unix	47
6.1	Comodines	47
6.1.1	¿Qué ocurre <i>realmente</i> ?	48
6.1.2	El signo de interrogación	48
6.2	Ganar tiempo con bash	49
6.2.1	Editando la línea de comandos	49
6.2.2	Completamiento de comandos y nombres de fichero	49
6.3	La entrada estándar y La salida estándar	50
6.3.1	Algunos conceptos de Unix	50
6.3.2	Redireccionar la salida	50
6.3.3	Redireccionar la entrada	51
6.3.4	Las tuberías	51
6.4	Multitarea	52
6.4.1	Usando el control de trabajos	52
6.4.2	Teoría del control de trabajos	56
6.5	Consolas virtuales: como estar en varios lugares a la vez	57
7	Pequeños programas potentes	61
7.1	El poder de Unix	61
7.2	Trabajando con ficheros	62
7.3	Estadísticas del sistema	64
7.4	¿Qué hay en un fichero?	65
7.5	Comandos de edición	67
8	Editando archivos con Emacs	71
8.1	¿Qué es Emacs?	71
8.2	Comenzar rápidamente en X	73

8.3	Editando varios archivos al mismo tiempo	74
8.4	Terminando una sesión de edición	75
8.5	La tecla Meta	76
8.6	Cortar, pegar, destruir y tirar	76
8.7	Buscar y reemplazar	77
8.8	¿Qué es lo que ocurre realmente?	78
8.9	Pidiendo ayuda a Emacs	79
8.10	Especializando buffers: Modos	80
8.11	Modos de programación	81
8.11.1	Modo C	81
8.11.2	Modo Scheme	81
8.11.3	Modo de correo	82
8.12	Como ser más eficiente aún	83
8.13	Personalizando Emacs	84
8.14	Averiguando más	88
9	¡Tengo que ser yo mismo!	91
9.1	Personalización del bash	91
9.1.1	Arranque del shell	91
9.1.2	Ficheros de arranque	92
9.1.3	Creando alias	92
9.1.4	Variables de entorno	93
9.2	Los ficheros de inicio de X Window	99
9.2.1	Configuración de twm	101
9.2.2	Configuración de fvwm	107
9.3	Otros ficheros de inicio	107
9.3.1	El fichero de configuración de Emacs	107
9.3.2	Configuración por defecto del FTP	107
9.3.3	Permitiendo un acceso remoto sencillo a su cuenta	108
9.3.4	Redirección de correo	109
9.4	Veamos algunos ejemplos	110

10 Hablando con los demás	111
10.1 Correo electrónico	111
10.1.1 Enviar correo	111
10.1.2 Leer el correo	112
10.2 Noticias más que de sobra	113
10.3 Localizar a la gente	113
10.3.1 Planes y proyectos	114
10.4 Uso remoto de sistemas	114
10.5 Intercambio de ficheros	115
10.6 Viajando por la telaraña	116
 11 Comandos divertidos	 117
11.1 <code>find</code> , el buscador de ficheros	117
11.1.1 Generalidades	117
11.1.2 Expresiones	118
11.1.3 Opciones	119
11.1.4 Test	120
11.1.5 Acciones	121
11.1.6 Operadores	122
11.1.7 Ejemplos	123
11.1.8 Una última palabra	124
11.2 <code>tar</code> , el archivador en cinta	125
11.2.1 Introducción	125
11.2.2 Opciones principales	125
11.2.3 Modificadores	125
11.2.4 Ejemplos	125
11.3 <code>dd</code> , el duplicador de datos	125
11.3.1 Opciones	125
11.3.2 Ejemplos	127
11.4 <code>sort</code> , el clasificador de datos	127
11.4.1 Introducción	127
11.4.2 Opciones	127
11.4.3 Ejemplos	127

12 Errores, equivocaciones, bugs, y otras molestias	129
12.1 Evitando errores	129
12.2 Qué hacer cuando algo va mal	130
12.3 No es fallo tuyo	130
12.3.1 Cuando hay un error	131
12.3.2 Notificando un error	131
A Introducción a vi	133
A.1 Una rápida historia de Vi	133
A.2 Rápido tutorial de Ed	134
A.2.1 Crear un fichero	134
A.2.2 Editar un fichero existente	135
A.2.3 Números de línea en detalle	136
A.3 Rápido tutorial de Vi	137
A.3.1 Ejecutar vi	137
A.3.2 Comandos de movimiento del cursor	138
A.3.3 Borrar texto	138
A.3.4 Salvar un fichero	138
A.3.5 ¿Qué viene a continuación?	138
A.4 Tutorial avanzado de Vi	139
A.4.1 Movimiento	139
A.4.2 Modificación del texto	140
A.4.3 Copiar y mover bloques de texto	142
A.4.4 Búsqueda y cambio de texto	144
B The GNU General Public License	147
C The GNU Library General Public License	155
D ¿Qué es LuCAS?	165
E ¿Qué es INSFLUG?	167

Capítulo 1

Introducción

How much does it cost to entice a dope-smoking Unix system guru to Dayton?
Brian Boyle, *Unix World's First Annual Salary Survey*

1.1 ¿Quién debe leer este libro?

¿Es Ud. la persona que debe leer este libro? Permítame responder a esta pregunta haciéndole estas otras: ¿Acaba de obtener Linux de alguna fuente, lo instaló y ahora quiere saber que puede hacer? O, ¿es un usuario de computadoras que nunca ha utilizado Unix y está considerando usar Linux pero quiere saber lo que puede hacer con él?

Si tiene este libro, la respuesta a esas preguntas probablemente es “sí”. Cualquiera que tiene Linux (el clon gratuito de Unix escrito por Linus Torvalds) en su PC, pero no sabe que hacer después de la instalación, debe leer este libro. Aquí encontrará explicaciones sobre la mayoría de los comandos básicos de Unix, e incluso de algunos avanzados. También trataremos acerca del potente editor GNU Emacs y de muchas otras aplicaciones importantes en Unix.

1.1.1 ¿Qué debe hacer antes de leer este libro?

Este libro supone que Ud. tiene acceso a un sistema Unix (¡Para aprender a nadar hay que mojarse!). Más importante aún, el sistema Unix antes mencionado debe ser una PC INTEL corriendo Linux. Este requerimiento no es estrictamente necesario, pero cuando las versiones de Unix difieran en su comportamiento, nos ocuparemos de las respuestas que exhiba Linux y no de las otras variantes de Unix.

Linux está disponible de distintas maneras, llamadas distribuciones. Espero que haya encontrado una distribución completa, como por ejemplo la MCC-Interim, y que la tenga instalada. Existen diferencias entre las distintas distribuciones de Linux, pero la mayoría de ellas son insignificantes. Ocasionalmente verá que en este libro las cosas se ven un poquito distintas a lo que ve en su

computadora. Si le sucede esto, es probable que esté utilizando una versión distinta de la mía: estoy interesado en recibir informes de tales casos.

Si Ud. es el superusuario (el ocupado del mantenimiento, el instalador) del sistema, deberá crearse una cuenta de usuario normal para Ud. mismo. Consulte por favor los manuales de instalación para obtener la información necesaria. Si no es el superusuario, deberá solicitarle a quien lo sea que le cree una cuenta.

Ahora deberá tener tiempo y paciencia. El aprendizaje de Linux no es fácil —la mayoría de las personas encuentran que es más fácil aprender el sistema operativo del Macintosh. Una vez que aprende Linux las cosas se vuelven mucho más fáciles. Unix es un sistema muy poderoso y con él es muy fácil realizar tareas complejas.

Además, este libro supone que Ud. está moderadamente familiarizado con algunos términos computacionales. Aunque este requerimiento no es necesario, hace más fácil la lectura. Debe conocer términos como ‘programa’ o ‘ejecución’. Si no los maneja, necesitará de la ayuda de alguien más avezado durante su aprendizaje de Unix.

1.2 ¿Cómo evitar leer este libro?

La mejor manera de aprender el uso de casi cualquier programa de computadora es poniéndose a trabajar sobre la computadora. La mayoría de las personas encuentran de poco beneficio la lectura de un libro si no utilizan a la vez el programa. Por lo tanto, la mejor manera de aprender Unix y Linux es utilizándolos. Utilice Linux para todo lo que pueda. Experimente. No tenga miedo —es *posible* que haga algunos líos, pero siempre puede volver a instalar todo. Mantenga actualizadas las copias de seguridad¹ y ¡diviértase!

Unix no es un sistema operativo tan intuitivamente obvio como algunos otros. Por lo tanto, es probable que deba leer al menos los capítulos 4, 5, y el 6, antes de empezar a experimentar.

La manera número uno para evitar la lectura de este libro es que utilice la documentación disponible en línea en Linux. Para ello debe aprender a utilizar el comando `man` —su descripción está en la Sección 4.2.

1.3 ¿Cómo leer este libro?

La manera sugerida para aprender Unix es leer un poquito, luego jugar un poquito. Juegue hasta que se sienta cómodo con los conceptos, y luego puede saltar entre los temas que se cubren en el libro. Encontrará una variedad de temas, algunos de los cuales serán de su interés. Luego de un tiempo, se sentirá lo suficientemente confiado como para utilizar comandos sin saber exactamente que es lo que deberían hacer. Esta es una buena señal.

Lo que mucha gente asocia con Unix es el shell de Unix: se trata de un programa especial que interpreta comandos. En la práctica, ésta es una manera conveniente de ver las cosas, pero debe

¹N. del T.: backups

saber que Unix es realmente mucho más que el shell, o mucho menos —dependiendo de como lo mire. Este libro le dirá como utilizar el shell, los programas que usualmente vienen con Unix, y algunos programas que no son muy comunes (pero que Linux casi siempre trae).

El capítulo actual es un meta-capítulo —se discute acerca del libro y de como aplicarlo para poder hacer nuestro trabajo. Los otros capítulos contienen:

Capítulo 2 trata del origen y destino de Unix y Linux. También se habla de la Free Software Foundation y el proyecto GNU.

Capítulo 3 habla de como arrancar y detener su computadora, y de lo que sucede durante esas operaciones. Gran parte de esto tiene que ver con temas que no es necesario conocer para utilizar Linux, pero aún así son útiles e interesantes.

Capítulo 4 introduce el shell de Unix. Es con este programa con el cual las personas realizan su trabajo, y corren los programas que necesitan. Trata de los programas y comandos básicos que necesitará conocer para utilizar Unix.

Capítulo 5 cubre el sistema de ventana X². X es la cara gráfica principal de Unix, e incluso algunas distribuciones lo configuran como opción predeterminada.

Capítulo 6 analiza algunas de las características más avanzadas del shell de Unix. El aprendizaje de las técnicas tratadas en este capítulo le ayudará a ser más eficiente.

Capítulo 7 tiene breves descripciones de muchos de los comandos Unix. Cuanto más herramientas sabe utilizar un usuario, más rápidamente puede concluir su trabajo.

Capítulo 8 describe el editor de textos Emacs. Se trata de un programa muy extenso que integra varias de las herramientas de Unix bajo una única interfaz.

Capítulo 9 trata acerca de las distintas maneras que hay de personalizar un sistema Unix para satisfacer sus gustos personales.

Capítulo 10 investiga los medios por los cuales un usuario Unix puede comunicarse con otras máquinas alrededor del mundo, incluyendo el correo electrónico y la Red de Alcance Mundial³.

Capítulo 11 describe algunos de los comandos con más opciones y más difíciles de utilizar.

Capítulo 12 muestra formas sencillas de evitar errores en Unix y Linux.

1.4 Documentación de Linux

Este libro, *Guía de Linux Para el Usuario*, está pensado para el principiante en Unix. Afortunadamente, el Linux Documentation Project (Proyecto de Documentación de Linux) también está escribiendo libros para los usuarios más experimentados⁴.

²N. del T.: X Window System.

³N. del T.: también conocida como WWW o World Wide Web.

⁴N. del T.: El proyecto LuCAS está ocupándose de traducir parte de la documentación de Linux al castellano. Para obtener información actualizada de las traducciones, vea el apéndice D.

1.4.1 Otros libros de Linux

Los otros libros incluyen *Installation and Getting Started*⁵, que es una guía para obtener e instalar Linux, *The Linux System Administrator's Guide*, trata como organizar y mantener un sistema Linux, y *The Linux Kernel Hacker's Guide*, que explica como modificar Linux. *The Linux Network Administrator Guide*⁶ muestra como instalar, configurar y utilizar una conexión de red.

1.4.2 COMOs

Además de los libros, el Proyecto de Documentación de Linux⁷ ha realizado una serie de pequeños documentos que describen como configurar algún aspecto particular de Linux. Por ejemplo, el SCSI-COMO describe algunas de las complicaciones que aparecen al utilizar SCSI —que es una manera estándar de comunicarse con ciertos dispositivos— y Linux.

Estos COMOs están disponibles en varios formatos: en un libro como *The Linux Bible* o *Dr. Linux*; en el foro de discusión `comp.os.linux.answers`; o en varios lugares de la WWW. El sitio central de la información acerca de Linux es <http://www.linux.org>.

1.4.3 ¿Qué es el Proyecto de Documentación de Linux?

Como casi todo lo asociado con Linux, el Proyecto de Documentación de Linux consiste en un conjunto de gente que trabaja alrededor del mundo. Originalmente organizado por Lars Wirzenius, el Proyecto está coordinado en la actualidad por Matt Welsh con la ayuda de Michael K. Johnson.

Se espera que el Proyecto de Documentación de Linux provea los libros que satisfagan todas las necesidades de documentación de Linux en algún momento. Por favor, díganos si hemos tenido éxito o qué es lo que debemos mejorar. Puede contactar al autor en `leg+@andrew.cmu.edu`, a Matt Welsh en `mdw@cs.cornell.edu`⁸.

1.5 Sistemas operativos

El propósito principal de un sistema operativo es dar el soporte que necesitan los programas que hacen un trabajo de interés para el usuario. Por ejemplo, Ud. puede estar utilizando un editor si quiere crear un documento. Ese editor no puede realizar su trabajo sin la ayuda del sistema operativo —necesitará esta ayuda para interactuar con la terminal, los archivos, y el resto de la computadora.

Si todo lo que hace el sistema operativo es dar soporte a las aplicaciones que Ud. desea utilizar, ¿por qué se necesita todo un libro sólo para hablar del sistema operativo? Existe un montón de actividades de mantenimiento de rutina (además de los programas que utilizará principalmente)

⁵N. del T.: Disponible en castellano como *Linux: Instalación y Primeros Pasos*. Vea el apéndice D

⁶N. del T.: Disponible en castellano como *Guía de Administración de Redes con Linux*. Vea el apéndice D

⁷N. del T.: El grupo INSFLUG ha traducido varios de los documentos HOWTO, acuñando el término COMO. Vea el apéndice E.

⁸N. del T.: y al coordinador de esta traducción César Ballardini en `cballard@santafe.com.ar`

que Ud. también necesitará realizar. En el caso de Linux, el sistema operativo contiene además un montón de “mini-aplicaciones” para ayudarle a realizar de manera más eficiente su trabajo. El conocimiento del sistema operativo le será de ayuda si su trabajo no se centra en alguna aplicación monolítica.

Los sistemas operativos (SO, para abreviar) pueden ser simples y mínimos, como DOS, o grandes y complejos, como OS/2 o VMS. Unix trata de estar en un punto medio. A la vez que proporciona más recursos y hace más que los primitivos SO, no trata de hacer *todo*. Unix se diseñó originalmente como una simplificación de un sistema operativo que se denominó Multics.

La filosofía original de diseño de Unix fue la de distribuir la funcionalidad en pequeñas partes: los programas⁹. De esta forma, el usuario puede obtener nueva funcionalidad y nuevas características de una manera relativamente sencilla mediante las diferentes combinaciones de pequeñas partes (programas). Además, en el caso de que aparezcan nuevas utilidades (y de hecho aparecen), puede Ud. integrarlas en su espacio de trabajo. Cuando escribo este documento, por ejemplo, utilizo activamente los siguientes programas: `fvwm` para manejar mis “ventanas”, `emacs` para editar el texto, `LATEX` para darle forma, `xdvi` para ver una presentación preliminar, `dvips` para prepararlo para la impresión, y luego `lpr` para imprimirlo. Si mañana llegara a conseguir otro programa de presentación preliminar de archivos dvi, podría utilizarlo en lugar del `xdvi` sin cambiar el resto de los programas que uso. En el momento actual mi sistema está corriendo treinta y ocho programas simultáneamente. (Gran parte de ellos son programas del sistema que “duermen” hasta que tienen algún trabajo específico para realizar.)

Cuando utilice un sistema operativo, deseará minimizar la cantidad de trabajo que tiene que llevar a cabo para concluir su tarea. Unix proporciona varias herramientas que pueden ayudarle, pero para ello debe conocer que es lo que hacen. No es muy productivo perderse una hora intentando poner a andar alguna cosa y al final tener que abandonar todo. Este libro le enseñará qué herramientas debe Ud. utilizar en qué situación, y como combinar esas herramientas entre sí.

La parte clave de un sistema operativo se denomina “núcleo”¹⁰. En muchos sistema operativos, como Unix, OS/2, o VMS, el núcleo proporciona las funciones necesarias para los programas que están corriendo, y los planifica para su ejecución. Esto significa básicamente que el programa A puede tomar una cierta cantidad de tiempo, el programa B la misma cantidad, etc.. El núcleo siempre está funcionando: es el primer programa que arranca cuando se enciende el sistema, y el último programa que hace algo cuando se apaga el equipo.

⁹Este diseño estuvo determinado por el hardware sobre el cual funcionó originalmente Unix. Por alguna extraña razón, el sistema operativo resultante fue muy útil sobre otras plataformas de hardware. El diseño básico fue tan bueno que aún veinticinco años después sigue usándose.

¹⁰N. del T.: *núcleo* del inglés “*kernel*”.

Capítulo 2

A propósito, ¿qué es Unix?

Ken Thompson tiene un automóvil en cuyo proyecto de diseño trabajó. A diferencia de otros automóviles, éste no tiene velocímetro, nivel de combustible, ni ninguna de las estúpidas luces que importunan al conductor moderno. En su lugar, si el conductor comete algún error, se enciende un enorme “?” en el centro del tablero. “El conductor experimentado”, dice, “sabrá usualmente que es lo que anda mal.”

2.1 La historia de Unix

En 1965, los Bell Telephone Laboratories (Bell Labs, una división de AT&T) estaba trabajando con General Electric y el Proyecto MAC del MIT para escribir un sistema operativo llamado Multics. Para hacer más corta una historia de por sí larga, Bell Labs decidió que el proyecto no estaba yendo a ninguna parte y disolvió el grupo. Esto, sin embargo, dejó a la Bell sin un buen sistema operativo.

Ken Thompson y Dennis Ritchie decidieron esbozar un sistema operativo que supiera las necesidades de la Bell Labs. Cuando Thompson llegó a necesitar un entorno de desarrollo (1970) que corriera en una PDP-7, aprovechó para poner en práctica sus ideas. Brian Kernighan le dio el nombre de Unix como un juego de palabras contra Multics.

Un tiempo después, Dennis Ritchie inventó el lenguaje de programación “C”. En 1973, se reescribió el sistema operativo Unix en C, en lugar del ensamblador original¹. En 1977, se movió Unix a una nueva máquina distinta de las PDP en las que había corrido previamente, mediante un proceso que se denomina **migración**. El hecho de que Unix estaba escrito en C facilitó la migración, pues gran parte del código ya escrito necesitó sólo una simple recompilación, y no una rescritura.

Al final de los '70, AT&T tenía prohibido competir en la industria de la computación, por lo que otorgó licencias económicas de Unix a varios colegios y universidades. Unix se difundió lentamente fuera de las instituciones académicas, pero pronto también se hizo popular en el ámbito comercial. El Unix de la actualidad es distinto del Unix de 1970. Existen dos corrientes principales: System V,

¹ “Lenguaje ensamblador” es un lenguaje de computadoras muy elemental, y que está indisolublemente ligado a un tipo particular de ordenador.

de los Unix System Laboratories (USL), que es una subsidiaria de Novell², y BSD, Berkeley Software Distribution. La versión USL está ahora por su cuarta versión, o SVR4³, mientras que la última BSD es la 4.4. Sin embargo, existen muchas diferentes versiones de Unix entre estas dos. La mayoría de las versiones de Unix son desarrolladas por compañías de software y derivan de uno de los dos agrupamientos. Las versiones que se utilizan actualmente incorporan características de ambas variaciones.

Las versiones actuales de Unix para PC Intel cuestan entre \$500 y \$2000.

2.2 La historia de Linux

El autor primario de Linux es Linus Torvalds. A partir de la versión original, ha sido mejorado por incontables personas de todo el mundo. Se trata de un clon del sistema operativo Unix, escrito desde cero. Ni USL, ni la Universidad de California, Berkeley, tuvieron participación en la escritura de Linux. Una de las cosas más interesantes de Linux es que el desarrollo ocurre simultáneamente alrededor del mundo. La gente ha contribuido desde Australia a Finlandia y esperamos que lo siga haciendo.

Linux comenzó como un proyecto para explorar el chip 386. Uno de los primeros proyectos de Linus fue un programa que pueda alternar entre la impresión de AAAA y BBBB. Esto luego evolucionó hacia Linux.

Linux ha sido registrado bajo los términos de la Licencia Pública General GNU (GNU General Public License) o GPL. Esta licencia, escrita por la Free Software Foundation (FSF), está diseñada para evitar que alguna persona restrinja la distribución de software. En pocas palabras, dice que aunque Ud. cobre a alguien por entregarle una copia, no podrá impedir que ese alguien la regale. También significa que debe estar disponible el código fuente⁴. Esto es útil para los programadores. Cualquiera puede modificar Linux y aún distribuir sus modificaciones, siempre que mantenga el código bajo la misma licencia.

En Linux puede correr la mayoría del software popular en Unix, incluyendo el Sistema de Ventanas X. X se desarrolló en el Instituto Tecnológico Massachusetts, para permitir a los sistemas Unix la creación de ventanas gráficas, y la cómoda interacción con las mismas. En la actualidad, el sistema X se usa en todas las versiones disponibles de Unix.

Además de las dos variaciones de Unix, System V y BSD, existe un conjunto de documentos de estandarización publicados por la IEEE denominados **POSIX**. Linux antes que nada satisface los documentos POSIX-1 y POSIX-2. Su apariencia se asemeja mucho a la de BSD en ciertas partes, mientras que es parecido a System V en otras. Es una combinación (y para la mayoría de las personas, una buena) de los tres estándares.

Algunas de las utilidades incluidas con las distribuciones de Linux proceden de la Free Software Foundation y son parte del proyecto GNU. El proyecto GNU es un esfuerzo para escribir un avanzado

²Recientemente vendida a Novell, antes era propiedad de AT&T.

³Una manera críptica de decir "System five, Revision four".

⁴El **código fuente** de un programa es lo que el programador lee y escribe. Luego se traduce a un código de máquina ilegible para las personas, que es el que la computadora interpreta.

sistema operativo portable con el estilo de Unix. “Portable” significa que pueda correr en una variedad de máquinas distintas, no sólo en PCs Intel, Macintoshes, o lo que sea. El sistema operativo del Proyecto GNU se llama Hurd. La principal diferencia entre el Linux y el GNU Hurd no se encuentra en la interfaz del usuario, sino en la del programador —el Hurd es un sistema operativo moderno, mientras que Linux es más parecido al diseño original de Unix.

La historia precedente de Linux olvida mencionar otra persona *más allá* de Linus Torvalds. Por ejemplo, H. J. Lu se ha ocupado de mantener `gcc` y la biblioteca C de Linux (dos elementos necesarios para todos los programas en Linux) casi desde el principio de la vida del sistema operativo. Ud. puede encontrar una lista de personas que merecen reconocimiento por su trabajo en cada sistema Linux, en el archivo `/usr/src/linux/CREDITS`.

2.2.1 Linux aquí y ahora

La primer cifra en el número de versión de Linux indica revisiones realmente importantes en alcance. Este número cambia muy lentamente y para la época en que esto se escribe (febrero de 1996), sólo está disponible la versión “1”⁵. La segunda cifra indica revisiones de menor envergadura. Los segundos números cuando son pares representan versiones más estables, en las que uno puede confiar plenamente; cuando son impares indican versiones en desarrollo, que son más propensas a contener errores. La última cifra es el número menor de lanzamiento —cada vez que se libera una nueva versión, que tal vez sólo arregla unos pocos problemas o agrega unas pocas características, se incrementa en uno este número. En febrero de 1996, la última versión estable es la 1.2.11, y la última en desarrollo es la 1.3.61.

Linux es un sistema grande y desafortunadamente contiene errores que los desarrolladores reparan al encontrarlos. Aún cuando algunas personas puedan experimentar malfuncionamientos regularmente, esto se debe normalmente a causa de piezas de hardware que no son estándares o que están defectuosas; los errores⁶ que afectan a todos los usuarios son escasos y no se dan seguido.

Por supuesto, éstos son los errores del núcleo. Los errores pueden presentarse en casi cada aspecto del sistema, y los usuarios de poca experiencia tienen problemas para diferenciar los distintos programas entre sí. Por ejemplo, cierto problema puede hacer que todos los caracteres aparezcan como un galimatías —¿es un error o una “característica”? Sorpresivamente es una característica —existen ciertas secuencias de control que al aparecer provocan el galimatías. Espero que este libro le sea de ayuda para poder identificar esas situaciones.

2.2.2 Unas pocas preguntas y respuestas

Antes de embarcarnos para nuestra larga travesía, repasemos las cuestiones ultra-importantes.

Pregunta: ¿Cómo se pronuncia Linux?

Respuesta: De acuerdo con Linus, debe pronunciarse con un sonido corto *ih*, como `prInt`, `mInImal`, etc. Linux debe rimar con Minix, que es otro clon de Unix. *No* debe pronunciarse como

⁵N. del T.: La traducción se hace en febrero de 1997, y ya la versión “2” es cosa de todos los días.

⁶N. del T.: *error* del inglés *bug*.

la pronunciación (americana) de “*Peanuts*” en Linus, sino más bien como “*LIH-nucks*”. Y la *u* es cortante, como en “*rule*”, no suave como en “*ducks*”. Linux debe al menos rimar con “*cynics*”.

Pregunta: ¿Por qué trabajar en Linux?

Respuesta: ¿Por qué no? Linux es generalmente más barato que los otros sistemas operativos (o al menos no más caro) y con frecuencia es menos problemático que algunos sistemas comerciales. Puede que no sea el mejor sistema para sus aplicaciones en particular, pero para alguien que está interesado en utilizar las aplicaciones Unix disponibles para Linux, se trata de un sistema de alta “performance”.

2.2.3 Software comercial en Linux

Existe un montón de software comercial disponible para Linux. Comenzando con Motif —que es una interfaz para el sistema X Window que vagamente nos recuerda a Microsoft Windows, Linux tiene cada vez más software comercial. En estos tiempos, Ud. puede adquirir cualquier cosa, desde Word Perfect (el popular procesador de palabras) hasta Maple, que es un paquete que realiza complejas manipulaciones simbólicas, para Linux.

Para aquellos lectores interesados en las cuestiones legales, esto está permitido por la licencia de Linux. A la vez que la licencia GNU General Public License (que se reproduce en el Apéndice B) cubre el núcleo de Linux, la licencia GNU Library General Public License (que se puede ver en el Apéndice C) cubre la mayor parte del código del cual dependen las aplicaciones. Esto permite que los proveedores de software comercial puedan vender sus aplicaciones y reservarse el código fuente de las mismas.

Debe Ud. percatarse de que los dos documentos antes nombrados son notificaciones de derechos de autor. Ellos *no* regulan como utiliza Ud. el software, solamente indican bajo cuales circunstancias Ud. puede copiar dicho software, o sus productos derivados. Para la Free Software Foundation, ésta es una distinción muy importante: Linux no involucra ninguna licencia “estilo pitón”, que lo envolverá y asfixiará, sino que está protegido por la misma ley que le impide a Ud. fotocopiar un libro.

Capítulo 3

En el principio

This login session: \$13.99, but for you \$11.88.

Tal vez ha tenido experiencia con MS-DOS u otro sistema operativo monousuario, como OS/2 o el de Macintosh. En esos sistemas operativos, Ud. no tiene que identificarse frente a la computadora antes de poder utilizarla; siempre se supone que es el único usuario en el sistema y que puede acceder a todo lo que quiera. Por el contrario, Unix es un sistema operativo multiusuario —no sólo puede utilizarlo más de una persona a la vez, sino que las distintas personas recibirán distinto trato.

Para poder identificar las personas, Unix necesita un *usuario* para que el proceso denominado **ingreso**¹ pueda identificarlo o identificarla². Apenas se enciende una computadora, tiene lugar un complejo proceso antes que la misma esté lista para ser usada. Como esta guía está orientada a Linux, les diré que es lo que sucede durante la secuencia de arranque³.

Debe advertir Ud. que si está utilizando Linux en alguna clase de ordenador distinto a una PC Intel, algunos temas de este capítulo no se aplicarán a su caso. En su mayoría, dichos temas se encontrarán en la Sección 3.1.

Si Ud. está interesado únicamente en utilizar su computadora, puede saltarse toda la información de este capítulo, excepto la de la Sección 3.3.

3.1 La computadora despierta

Lo primero que sucede cuando se conecta la alimentación a su computadora es la ejecución de un programa denominado BIOS. Su nombre proviene de las iniciales de **B**asic **I**nput/**O**utput **S**ystem, que significa Sistema de Entrada/Salida Básico. Este programa está almacenado de manera permanente dentro de la computadora en chips que normalmente son del tipo de lectura solamente. Para

¹N. del T.: *ingreso* en inglés **logging in**

²Desde ahora en adelante, utilizaré los pronombres masculinos para identificar a todas las personas. Esta es la convención estándar en inglés (N. del T.: y en castellano), y con esto no quiero decir que sólo los hombres pueden utilizar computadoras.

³N. del T.: *arranque*, del inglés *boot-up*.

nuestros propósitos, el BIOS es un programa que nunca se puede cambiar. Al funcionar, realiza ciertas comprobaciones mínimas y luego se fija si hay un disquete en la primer disquetera. Si encuentra alguno, mira en su “sector de arranque” y comienza a ejecutar su código, si lo hay. Si hay un disquete pero no contiene un sector de arranque, el BIOS mostrará el siguiente mensaje:

Non-system disk or disk error

Si quitamos el disquete y apretamos una tecla, el proceso de arranque continuará.

Si no existe un disquete en la disquetera, el BIOS se fija si existe un registro maestro de arranque (MBR: master boot record) en el disco rígido. Si es así comienza a ejecutar el código que allí encuentra, que se ocupa de cargar el sistema operativo. En los sistemas Linux, el cargador se llama LILO —por **L**inux **L**Oader—, puede ocupar el MBR y en ese caso cargará el Linux. (La distribución en particular que esté utilizando puede manejar el arranque desde el disco rígido de otra manera, así que debe controlar esto en la documentación que se incluye con la distribución. Otra buena referencia es la documentación de LILO, [1].)

3.2 Linux despierta

Luego que el BIOS le pasa el control a LILO, LILO a su vez le pasa el control al **núcleo** de Linux. El núcleo es el programa central del sistema operativo, que controla a todos los demás. Lo primero que hace Linux una vez que comienza su ejecución es cambiar la CPU a modo protegido. El procesador 80386⁴ que controla su computadora tiene dos modos que se denominan “modo real” y “modo protegido”. El sistema operativo DOS corre en modo real, al igual que el BIOS. Sin embargo, los sistemas operativos más avanzados necesitan correr en modo protegido. Por ello, cuando Linux arranca, descarta el BIOS.

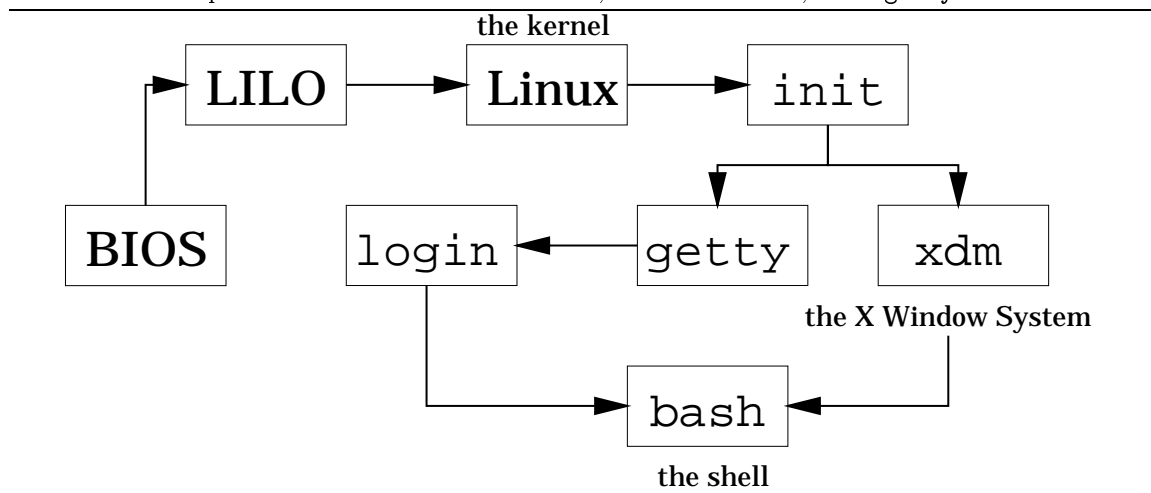
Los procesadores distintos al 386 llegan a este estado de manera diferente. Ningún otro procesador necesita cambiar a modo protegido, y sólo unos pocos tienen un contexto tan pesado de carga, como LILO y el BIOS. Una vez que arranca el núcleo, Linux trabaja casi de la misma manera.

A continuación, Linux mira que clase de hardware tiene debajo. Necesita saber que clase de discos rígidos tiene, si hay o no un ratón de bus, si está conectado a una red, y otras trivialidades como esas. Linux no puede recordar ciertas cosas luego de apagado, de manera que las pregunta cada vez que arranca. Afortunadamente, no se las pregunta a *Ud.*: ¡se las pregunta al *hardware*! Durante el arranque, el núcleo de Linux muestra unos cuantos mensajes, aunque con ciertas variaciones. Puede leer acerca de esos mensajes en la Sección 3.4. Este proceso de consulta puede causar algunos problemas con el sistema, pero si lo hace, lo hará probablemente cuando instale Linux por primera vez. Si tiene problemas, consulte la documentación de la distribución.

El núcleo solamente se ocupa de administrar los otros programas, entonces cuando está satisfecho con que todo anda bien debe arrancar otro programa para que haga los trabajos útiles. El programa que el núcleo arranca se llama `init`. (Note la diferencia en el tipo de letra. Las cosas en **este tipo de letra** serán usualmente los nombres de programas, de archivo, de directorio u otros items

⁴Cuando hable del 80386, también hago referencia a computadoras basadas en 80486, Pentium, y Pentium Pro a menos que específicamente indique lo contrario. Además, abreviaré 80386 mediante el término 386.

Figura 3.1 El camino que sigue un PC Intel hasta que nos muestra el indicativo del intérprete de comandos. `init` puede o no arrancar X. Si lo hace, corre `xdm`. Si no, corre `getty`.



relacionados con las computadoras.) Una vez que el núcleo arranca `init`, no lanza ningún otro programa. El núcleo se transforma así en un administrador y proveedor, no en un programa activo.

Por lo tanto, para saber que es lo que hace la computadora luego que el núcleo arranca, deberemos examinar `init`. La complicada secuencia de arranque por la que atraviesa `init` no es idéntica en todas las computadoras. Para Linux existen varias versiones de `init`, y cada una hace las cosas a su manera. Además también influye si su máquina está en red, e incluso cual distribución utilizó para instalar Linux. Algunas de las cosas que pueden suceder cuando `init` arranca son:

- El control de integridad del sistema de archivos. ¿Pero qué es un sistema de archivos?, se estará preguntando Ud. . Un sistema de archivos es la disposición de los archivos en el disco rígido. Además permite que Unix sepa cuales partes del disco rígido están ocupadas y cuales no. Desafortunadamente, ciertos factores como los cortes en el suministro de energía hacen que la información que el sistema de archivos tiene sobre la disposición en el disco de los archivos no coincida con la disposición real. En estos casos se ve la utilidad de un programa llamado `fsck`, que es capaz de encontrar estas situaciones y (con suerte) corregirlas.
- Se lanzan programas especiales de encaminamiento para las redes. Estos programas informan a su computadora cómo se supone que puede comunicarse con las otras.
- Se borran los archivos temporales que crean ciertos programas.
- Se actualiza correctamente el reloj del sistema. Esto es más complicado de lo que puede parecer, pues Unix de manera predeterminada, necesita la hora en UCT (Universal Coordinated Time), también conocido como hora de Greenwich, y el reloj de la CMOS —que es alimentado por una batería dentro de la computadora— muy probablemente estará configurado con la hora local. Esto significa que debe tener algún programa que lea la hora del reloj de la CMOS y la corrija transformándola en hora UCT.

Después que `init` termina con sus actividades de arranque, comienza con sus tareas planificadas. `init` se convierte así en el padre de todos los procesos del sistema Unix. Un proceso es simplemente un programa que está corriendo; como cualquier programa puede correr más de una vez, entonces puede haber más de un proceso para un programa dado en particular.

En Unix, los procesos —instancias de un programa— se crean mediante una *llamada al sistema*⁵ —que es un servicio provisto por el núcleo— denominada `fork`. (Se lo llama “fork” (bifurcación) pues un proceso se bifurca en dos independientes.) `init` *forkea* (bifurca) unos cuantos procesos, los que a su vez *forkean*⁶ otros. En su sistema Linux con toda seguridad `init` corre varias instancias de un programa llamado `getty`. `getty` es el programa que le permitirá iniciar el ingreso al usuario, y que a continuación lanzará el programa `login`.

3.3 La actuación del usuario

3.3.1 El ingreso

Lo primero que hay que hacer para poder utilizar una máquina con Unix es identificarse frente a la misma. Este proceso, conocido en inglés como **logging in (registro de ingreso)**, es la manera que tiene Unix de saber cuales son los usuarios autorizados para utilizar el sistema. Durante el ingreso se le preguntará un nombre de cuenta y una contraseña⁷. Los nombres de las cuentas son por lo general parecidos a los nombres de las personas, y se lo asignará el administrador del sistema. Si Ud. es el administrador del sistema deberá crearse una cuenta para poder trabajar sin los cuidados especiales que requiere la cuenta `root`. (La información necesaria para llevar a cabo dicha tarea podrá encontrarla en *Installation and Getting Started*⁸ o en *The Linux System Administrator's Guide*.)

Luego de que tenga lugar la secuencia de procedimientos durante el arranque, Ud. verá en la pantalla algo como lo que se muestra a continuación: (la primer línea es meramente un saludo —podría ser alguna advertencia legal, o cualquier otra cosa)

```
Welcome to the mousehouse. Please, have some cheese.
```

```
mousehouse login:
```



Sin embargo, puede ser que lo que el sistema le presente al arrancar *no* se parezca a esto. Por ejemplo, puede que en lugar de una aburrida pantalla de texto nos muestre una pantalla gráfica. Aún en este caso, la computadora le pedirá sus datos al ingreso y más o menos de la misma manera. Si éste es el caso en su sistema, entonces el entorno de trabajo que utilizará es el sistema X. El X Window System es un sistema de ventanas y en el capítulo 5 se discutirán algunas de las diferencias con las que se enfrentará. De cualquier manera, el proceso de ingreso será similar en ambos casos. Si va a utilizar X, busque las apariciones de una X grandota en el margen de esta guía.

⁵N. del T.: *llamada al sistema* del inglés *system call*.

⁶N. del T.: Mis disculpas por el término, prometo no usarlo más :-).

⁷N. del T.: *contraseña, palabra clave* del inglés *password*

⁸N. del T.: Puede consultar la versión en castellano ya existente

Esta es, por supuesto, la invitación que Ud. buscaba para **ingresar** (login). A lo largo de este manual utilizaremos el usuario ficticio (o no tan ficticio, dependiendo de su máquina) **larry**⁹. Donde Ud. vea **larry** debe poner el nombre de su propia cuenta. Los nombres de las cuentas se basan en general en los nombres reales; los sistemas Unix más grandes y serios tienen cuentas que usan el apellido del usuario, alguna combinación de nombre y apellido, y aún se da el caso que deban agregarle algunos dígitos. Por ejemplo, nombres de cuentas posibles para Larry Greenfield podrían ser: **larry**, **greenfie**, **lgreenfi**, **lg19**.

mousehouse será, de la misma manera, el “nombre” de la máquina sobre la que trabajaré. Es posible que cuando Ud. instale Linux se le pregunte el nombre que desea asignarle. Esto no es demasiado importante, pero cada vez que haga falta utilizaré **mousehouse** o, más raramente, **lionsden** cuando necesite de un segundo sistema por razones de claridad o de contraste.

Luego de teclear **larry** y apretar Intro, nos encontramos con lo siguiente:

```
mousehouse login: larry
Password:
```

Linux está solicitando su **contraseña**¹⁰. Cuando escriba su contraseña no podrá ver lo que está tecleando, en razón de la privacidad necesaria a dicha palabra. Escriba cuidadosamente: puede borrar caracteres, pero no podrá ver lo que está editando. No escriba muy despacio si hay gente mirándole —podrían llegar a descubrirla. En cualquier caso, si se equivoca no se haga problemas, pues el sistema le dará otra oportunidad de ingresar.

Luego que ingrese correctamente su nombre de cuenta o nombre de usuario y su contraseña, aparecerá un corto mensaje denominado “mensaje del día”¹¹ que se obtiene del archivo **/etc/motd**. Este archivo se utiliza para dar a conocer cualquier clase de información a los usuarios con respecto al estado del sistema y es responsabilidad del administrador del sistema fijar su contenido. Después de todo esto aparece un **prompt**. Un prompt es un símbolo que nos indica que la computadora está lista para recibir un comando. Debe parecerse a la siguiente figura:

```
/home/larry$
```



Si Ud. está utilizando X Window, el prompt aparecerá en alguna de las “ventanas” que hay en pantalla. (Las “ventanas” son simples cajas rectangulares.) Para escribir en el prompt, mediante el ratón, mueva el cursor del mismo (probablemente se ve como una gran “X” o una flecha) dentro de la ventana.

3.3.2 Al abandonar la computadora

¡No apague la computadora directamente! ¡Se arriesga a perder valiosos datos!

A diferencia de la mayoría de las versiones de DOS, no es una buena idea apagar la llave de alimentación de la computadora así como así, cuando termine de utilizarla. Si reinicia la máquina

⁹N. del T.: He dejado aquí sin traducir ni cambiar el nombre del autor de la versión en inglés, y lo mismo vale para otros nombres como el de la máquina, etc. .

¹⁰N. del T.: *contraseña* del inglés *password*.

¹¹N. del T.: *mensaje del día* del inglés Message Of The Day(motd).



(con el botón de reset) sin antes tomar las debidas precauciones será igual de pernicioso. Linux tiene una **antememoria** o **caché** que mejora el rendimiento del disco. Esto significa que temporalmente guarda en RAM información perteneciente al sistema de almacenamiento permanente¹². Las diferencias entre lo que Linux cree que hay en el disco y lo que efectivamente está almacenado en el disco se sincronizan cada 30 segundos. Si desea apagar o reiniciar la computadora, necesitará ejecutar algún procedimiento que indique a Linux que debe detener el sistema de caché y actualizar la información en el disco.

Si ya ha terminado sus tareas con la computadora, pero aún está dentro del sistema (ya tecleó su nombre de usuario y su palabra clave), lo primero que debe hacer es registrar la salida (logout) del sistema. Para ello, teclee el comando `logout`. Todos los comandos se envían oprimiendo la tecla marcada como `Enter`, `Return` ó `Intro`. Hasta que no apreta `Intro` no pasa nada, y por lo tanto, puede borrar lo que escribió y recomenzar.

```
/home/larry$ logout
```

```
Welcome to the mousehouse. Please, have some cheese.
```

```
mousehouse login:
```

Ahora puede ingresar otro usuario.

3.3.3 Como apagar la computadora

Si el suyo se trata de un sistema en el cual trabaja un único usuario —*Ud.*, :-)— puede desear apagar la computadora cuando haya finalizado su trabajo con ella¹³. Para “*bajar el sistema*”¹⁴ o sea, apagarlo ordenadamente— deberá ingresar a una cuenta especial denominada **root**. La cuenta del **root** es la cuenta del administrador del sistema y puede acceder a todos los archivos que existen en el sistema. Si desea apagar la computadora, primero debe bajar el sistema, y para ello deberá obtener la palabra clave que utiliza el administrador del sistema. Por favor, ingrese Ud. entonces como **root**:

```
mousehouse login: root
```

```
Password:
```

```
Linux version 1.3.55 (root@mousehouse) #1 Sun Jan 7 14:56:26 EST 1996
```

```
/# shutdown now
```

```
Why? fin del dia de trabajo
```

```
URGENT: message from the sysadmin:
```

¹²La diferencia entre “RAM” y un disco rígido es como la diferencia entre la memoria de corto plazo y la de largo plazo. Al quitar la alimentación de la computadora es como si le diéramos un fortísimo golpe en la cabeza—olvidará todo lo que tenía en la memoria de corto plazo. Pero las cosas guardadas en la memoria de largo plazo, el disco rígido, estarán bien. El disco es miles de veces más lento que la RAM.

¹³Para evitar el debilitamiento prematuro de ciertos componentes de hardware, es mejor que apague la computadora sólo cuando ya no la vaya a utilizar por el resto del día. Encender y apagar una vez al día la computadora probablemente es el mejor compromiso entre ahorrar energía y machacar el sistema.

¹⁴N. del T.: *bajar el sistema* del inglés *shutdown*.

```
System going down NOW
```

```
... fin del dia de trabajo ...
```

```
Now you can turn off the power...
```

El comando “shutdown now” prepara al sistema para que pueda apagarse o reiniciarse. Debe esperar por el mensaje que dice que es seguro apagar o reiniciar el sistema. (Cuando el sistema le pregunta “Why?”¹⁵, sólo le está preguntando una razón para decirle a los otros usuarios. Como no hay otra persona utilizando el sistema cuando Ud. lo baja, puede responderle con lo que se le ocurra, o con nada.)

Un pequeño dato para el perezoso: como alternativa al enfoque de logout/login se puede utilizar el comando `su`. Mientras está trabajando con su nombre de cuenta habitual (`larry` a nuestros efectos) escriba “su” y luego `Intro`. Se le solicitará la palabra clave del administrador del sistema, y si la ingresa correctamente tendrá a partir de dicho momento los privilegios que le corresponden al mismo. Ahora, con estos privilegios agregados, podrá bajar el sistema mediante el comando “shutdown now”.

3.4 Mensajes del núcleo

Cuando se conecta la alimentación a su computadora, aparece una serie de mensajes en la pantalla, que describe el hardware conectado al ordenador. Estos mensajes son producto de la actividad del núcleo de Linux. En esta sección, intentaremos describir y explicar esos mensajes.

Naturalmente, estos mensajes varían de máquina a máquina. Describiré entonces los mensajes que aparecen en la mía. El siguiente ejemplo contiene todos los mensajes estándar y algunos específicos. (En general, la máquina sobre la que voy a realizar la descripción está mínimamente configurada: no verá por lo tanto mucha información relacionada con la configuración de dispositivos.) Se utilizó Linux versión 1.3.55 —uno de los más recientes al momento de escribir.

1. Lo primero que hace Linux es decidir que clase de tarjeta de video y pantalla tiene conectado, de manera de seleccionar un tamaño de letra apropiado. (Cuanto más pequeño es el tamaño, más caben en una pantalla a la vez.) Linux puede consultarle si desea algún tipo de letra en particular, o puede que tenga una selección ya compilada en su interior¹⁶.

```
Console: 16 point font, 400 scans Console: colour VGA+ 80x25, 1 virtual console (max
```

En este ejemplo, el dueño de la máquina decidió, al momento de compilar, que utilizaría el tipo de letra estándar tamaño grande. Además, notará que la palabra “color” está mal escrita. Evidentemente, Linus aprendió la versión incorrecta de idioma inglés¹⁷.

¹⁵N. del T.: *why?* significa *¿por qué?*.

¹⁶“Compilada” se refiere al proceso mediante el cual un programa de computadora escrito por una persona se traduce a algo que entiende la computadora. Si una característica se ha “compilado en su interior” significa que está incluida en el programa.

¹⁷N. del T.: “colour” es la variante ortográfica del inglés insular y “color” la norteamericana.

2. La próxima tarea del núcleo es informar acerca de la velocidad de su sistema, medida en “BogoMIPS”. Un “MIP” equivale a un millón de instrucciones por cada segundo, mientras que un “BogoMIP” es un “bogus MIP”¹⁸: cuantas veces la computadora puede hacer absolutamente nada por cada segundo. (Como este bucle no hace nada en realidad, el número calculado no es tampoco una medida de la velocidad del sistema.) Linux utiliza este número cuando necesita esperar por algún dispositivo de hardware.

```
Calibrating delay loop.. ok - 33.28 BogoMIPS
```

3. El núcleo de Linux también nos comenta acerca de la utilización de la memoria:

```
Memory: 23180k/24576k available (544k kernel code, 384k reserved, 468k data)
```

Esto significa que la máquina tiene 24 megabytes de memoria. Parte de esta memoria está reservada por el núcleo. El resto puede ser utilizada por los programas. Éste es el lugar temporal conocido como **RAM** que se utiliza sólo como almacenamiento de corto plazo. Su computadora también dispone de una memoria permanente denominada **disco rígido**. El contenido del disco rígido se mantiene intacto aún luego de quitar la alimentación del ordenador.

4. A lo largo del proceso de arranque, Linux controla distintas partes del hardware y nos muestra mensajes indicativos de las mismas.

```
This processor honours the WP bit even when in supervisor mode. Good.
```

5. Ahora Linux pasa a la configuración de red. Lo que veremos a continuación está mejor explicado en *The Linux Networking Guide*¹⁹, y está fuera del alcance de este documento.

```
Swansea University Computer Society NET3.033 for Linux 1.3.50
IP Protocols: ICMP, UDP, TCP
```

6. Linux soporta la FPU, la unidad de punto flotante²⁰. Ésta es un chip especial (o parte de un chip en el caso del procesador 80486DX) que se ocupa de la aritmética con números no enteros. Algunos de esos chips no están en buenas condiciones, y cuando Linux trata de identificarlos la máquina “se cuelga”²¹. La máquina deja de funcionar. Si esto sucede, Ud. verá en su pantalla lo siguiente:

```
Checking 386/387 coupling...
```

Sino, el mensaje será:

¹⁸N. del T.: *bogus* significa *espurio* (para los argentinos: *trucho*).

¹⁹N. del T.: Disponible en castellano. Vea el apéndice D.

²⁰N. del T.: *unidad de punto flotante* del inglés *Floating Point Unit*.

²¹N. del T.: *se cuelga* del inglés *crashes*.

```
Checking 386/387 coupling... Ok, fpu using exception 16 error reporting.
```

si está utilizando un 486DX. Si tiene un 386 con un 387, le aparecerá:

```
Checking 386/387 coupling... Ok, fpu using irq13 error reporting.
```

7. Y ahora se controla la instrucción “halt”.

```
Checking 'hlt' instruction... Ok.
```

8. Luego de esta configuración inicial, Linux escribe una línea identificándose a sí mismo. En ella explica cual es la versión del núcleo, la del compilador de C de GNU que se utilizó para compilarlo, y la fecha de dicha compilación.

```
Linux version 1.3.55 (root@mousehouse) (gcc version 2.7.0) #1 Sun Jan 7 14:56:26 EST 1996
```

9. El controlador serie comienza a consultar acerca del hardware instalado. Un **controlador** es una parte del núcleo que controla un dispositivo, normalmente un periférico. Se hace responsable por los detalles de la comunicación entre la CPU y el dispositivo. Esto permite que la gente que escribe aplicaciones de usuario se pueda concentrar en la aplicación, pues no tiene que preocuparse por como trabaja exactamente la computadora.

```
Serial driver version 4.11 with no serial options enabled
tty00 at 0x03f8 (irq = 4) is a 16450
tty01 at 0x02f8 (irq = 3) is a 16450
tty02 at 0x03e8 (irq = 4) is a 16450
```

Aquí vemos que ha encontrado tres puertos serie. Un puerto serie es el equivalente del puerto COM en DOS; es el dispositivo que se utiliza normalmente para comunicarse con modems y ratones.

Lo que significan las líneas anteriores es que el puerto serie 0 (COM1) tiene la dirección 0x03f8. Cuando dicho puerto interrumpe al núcleo, usualmente para decir que tiene datos listos, utiliza la IRQ 4²². Las IRQ son un medio que tienen los periféricos de hablarse con el software. Cada puerto serie también tiene un chip controlador, que usualmente se denomina 16450; aunque también puede ser el 8250 o el 16550.

10. A continuación viene el puerto paralelo, que normalmente se conecta a la impresora. Los puertos paralelos en Linux tienen nombres que comienzan con **lp**. **lp** son las iniciales de **L**ine **P**rinter²³, aunque en los tiempos actuales parece más apropiado usar las iniciales de **L**aser **P**rinter. (Sin embargo, Linux se comunica sin problemas con cualquier clase de impresora conectada al puerto paralelo: de matriz de puntos, chorro de tinta, o laser.)

²²N. del T.: IRQ, Interrupt ReQuest (petición de interrupción).

²³N. del T.: Impresora de líneas.

```
lp0 at 0x03bc, (polling)
```

Este mensaje dice que ha encontrado un puerto paralelo, y que usará el controlador de dispositivo estándar.

11. Linux identifica luego las unidades de disco rígido. En el sistema que le estoy mostrando, **mousehouse**, tengo instalados dos unidades de disco rígido IDE.

```
hda: WDC AC2340, 325MB w/127KB Cache, CHS=1010/12/55
hdb: WDC AC2850F, 814MB w/64KB Cache, LBA, CHS=827/32/63
```

12. El núcleo ahora busca las unidades de disco flexible. En este ejemplo, la máquina tiene dos unidades: la “A” es una unidad de 5 1/4", y la “B” es una de 3 1/2". Linux denomina **fd0** a la unidad “A”, y **fd1** a la unidad “B”.

```
Floppy drive(s): fd0 is 1.44M, fd1 is 1.2M
floppy: FDC 0 is a National Semiconductor PC87306
```

13. El próximo controlador que arranca en mi sistema ejemplo es el controlador SLIP. Escribe un mensaje acerca de su configuración.

```
SLIP: version 0.8.3-NET3.019-NEWTTY (dynamic channels, max=256) (6 bit encapsulation enabled)
CSLIP: code copyright 1989 Regents of the University of California
```

14. El núcleo explora dentro de los discos rígidos que ha encontrado. Busca las distintas particiones que tiene cada uno. Una partición es una sección lógica de una unidad que se utiliza para evitar que interfieran entre sí distintos sistemas operativos coexistentes. En este ejemplo, la computadora tiene dos discos (**hda** y **hdb**) con cuatro y una particiones, respectivamente.

```
Partition check:
hda: hda1 hda2 hda3 hda4
hdb: hdb1
```

15. Finalmente, Linux **monta** la partición raíz. Ésta es la partición del disco en la cual reside el sistema operativo Linux. Cuando Linux “monta” esta partición, hace que esté disponible para los usuarios.

```
VFS: Mounted root (ext2 filesystem) readonly.
```


Capítulo 4

El shell de Unix

Es muy fácil crear archivos en el sistema operativo UNIX. Por lo tanto, los usuarios tienden a crear muchos archivos que utilizan una gran cantidad de espacio. Se ha dicho que la única cosa estándar y común a todos los sistemas UNIX es el mensaje-del-día que les pide a los usuarios que borren los archivos que no necesitan.

Guía del administrador de System V.2

4.1 Comandos Unix

Cuando ingresa al sistema Unix, se enfrenta con algo más o menos como lo que se muestra a continuación:

```
/home/larry$
```

Ese “algo” se denomina **prompt**¹. Como su nombre sugiere, le solicita a Ud. que ingrese un comando. Todos los comandos Unix consisten de una secuencia de letras, números y caracteres. No son válidos los espacios dentro del nombre del comando. Algunos comandos válidos son **mail**, **cat**, y **CMU_is_Number-5**. Algunos caracteres no están permitidos—volveremos a este tema más adelante. Unix además hace **diferencia entre mayúsculas y minúsculas**², lo que significa que **Cat** y **cat** son comandos distintos.

El prompt se muestra como resultado del accionar de un programa especial denominado **intérprete de comandos**³. El intérprete de comandos o shell acepta los comandos que escribe el usuario y los ejecuta. Los comandos pueden formar programas en el lenguaje del intérprete de comandos, y a dichos programas se los denomina “guiones de shell”.

¹N. del T.: *prompt* significa *solicitud*.

²La diferenciación entre mayúsculas y minúsculas es una cosa muy personal. Algunos sistemas operativos como OS/2 y Windows NT preservan las diferencias, pero no las cuentan como distintas. En la práctica habitual con Unix, se utiliza raramente la diferenciación. La situación de tener un par de comandos **Cat** y **cat** diferentes, no es común.

³N. del T.: *intérprete de comandos* del inglés *shell*.

Los shell en Unix se clasifican en dos grandes grupos: los tipo Bourne y los tipo C. Los shell tipo Bourne toman su nombre a partir de su inventor, Steven Bourne. Steven Bourne escribió el shell original de Unix, denominado **sh**; a partir de entonces, la mayoría de los shells tienen un nombre con el sufijo **sh** para indicar que son extensiones de la idea original. Existen varias implementaciones de este shell, que colectivamente llevan el nombre de shells Bourne. También son comunes los shells tipo C cuyo original fue implementado por Bill Joy. Tradicionalmente, los shell Bourne se han utilizado para los scripts de shell y por razones de compatibilidad con el **sh** original, mientras que los shells C han sido más comunes en su aplicación interactiva. (Los C tienen ventajas en cuanto a sus mejores características interactivas, aunque son más difíciles de programar.)

Linux viene con un shell Bourne denominado **bash**, escrito por la organización “Free Software Foundation”⁴. El nombre **bash** proviene de **B**ourne **A**gain **S**hell, uno de los tantos juegos de palabras en Unix. Se trata de un shell Bourne “avanzado”: tiene las capacidades estándar de programación que se encuentran en todos los shells Bourne y además varias de las características interactivas que se encuentran en los shells C. **bash** es el shell predeterminado cuando uno usa Linux.

Apenas ingresa por primera vez, el prompt que Ud. ve es producto de la acción de **bash**, en otras palabras: está Ud. corriendo su primer programa Unix, el shell **bash**. Mientras esté conectado, el shell **bash** estará permanentemente funcionando.

4.1.1 Un comando Unix típico

El primer comando que debe conocer es **cat**. Para utilizarlo, escriba “**cat**” y luego oprima Intro:

```
/home/larry$ cat
```

Si tiene ahora el cursor posicionado en una línea nueva, entonces lo que ha hecho está bien. Existen unas cuantas variantes que podría haber tecleado—algunas funcionarán, otras no.

- Si tuvo algún error de tecleo al escribir “**cat**”, debería haber visto algo más o menos así:

```
/home/larry$ ct
ct: command not found
/home/larry$
```

Por lo tanto, el shell le informa que no pudo encontrar un programa denominado “**ct**”, y le ofrece otro prompt para seguir trabajando. Recuerde que Unix hace diferencia entre mayúsculas y minúsculas: **CAT** está mal escrito.

- Puede que Ud. haya escrito algunos espacios en blanco antes del comando, como:⁵

```
/home/larry$      cat
```

⁴N. del T.: La Fundación para el Software Libre —tal es la traducción de su nombre— se ocupa de la producción y distribución de software que no tiene limitaciones en su uso y/o copia, enfrentando de esta manera las prácticas de las compañías tradicionales de desarrollo. Que el software sea libre no significa que sea barato: la libertad incluye la gratuidad, pero es mucho más que ello.

⁵El ‘**␣**’ indica que el usuario escribió un espacio.

Sin embargo, el resultado es correcto, y el programa `cat` corre sin problemas.

- También puede que haya oprimido un `Intro` en una línea en blanco; no se preocupe y continúe, pues no tiene ningún efecto.

Doy por sentado que Ud. ha corrido con éxito `cat` y está esperando que haga algo que lo maraville. Pues bien, no, no es un juego. `cat` es una muy útil utilidad que no parece muy útil a primera vista. Escriba cualquier cosa y luego oprima `Intro`. Lo que verá es:

```
/home/larry$ cat
Help! I'm stuck in a Linux program!
Help! I'm stuck in a Linux program!
```

(El texto *inclinado* marca que es lo que he tecleado.) Lo que parece haber hecho `cat` es devolver un eco de lo escrito. Esto es útil a veces, tal vez no ahora. Así que salgamos de este programa y veamos otros con beneficios más obvios.

Para finalizar ciertos comandos Unix, teclee `Ctrl-d`⁶. `Ctrl-d` es el carácter end-of-file⁷, o EOF, para abreviar. Puede que en ciertos libros de texto aparezca como end-of-text⁸. Nos referiremos a este carácter como EOF. Es un carácter de control que informa a los programas Unix que ha cesado el ingreso de datos. Cuando `cat` ve que no teclea más nada, termina.

Para ver otro ejemplo parecido, pruebe el programa `sort`. Como su nombre lo indica, `sort` es un programa de clasificación. Si Ud. teclea unas cuantas líneas y luego oprime `Ctrl-d`, `sort` las mostrará a la salida de manera ordenada. Esta clase de programas se denominan **filtros**, porque toman texto desde su entrada, lo filtran, y lo vierten a su salida modificado de alguna manera. Tanto `cat` como `sort` son filtros inusuales. `cat` es inusual pues lee el texto de entrada y *no* lo cambia. `sort` es inusual porque lee *todas* las líneas de entrada hasta el EOF, antes de emitir su salida. La mayoría de los filtros trabajan sobre la base de línea por línea: leen una línea de la entrada, realizan cierto cómputo, y escriben una línea diferente de salida.

4.2 Autoayuda

El comando `man` muestra las páginas de la guía de referencia para un comando dado⁹. Por ejemplo:

```
/home/larry$ man cat

cat(1) cat(1)

NAME
```

⁶Mantenga oprimida la tecla etiquetada `Ctrl` y aprete `d`, luego suelte ambas.

⁷N. del T.: fin de archivo.

⁸N. del T.: fin de texto.

⁹`man` también puede mostrar información acerca de las llamadas al sistema, subrutinas, formatos de archivos, etc.. En la versión original de Unix el comando mostraba la información exactamente igual a la que aparecía en la documentación impresa. Por ahora, es probable que Ud. sólo esté interesado en obtener ayuda acerca de los comandos.

cat - Concatenates or displays files

SYNOPSIS

```
cat [-benstuvAET] [--number] [--number-nonblank] [--squeeze-blank]
[--show-nonprinting] [--show-ends] [--show-tabs] [--show-all]
[--help] [--version] [file...]
```

DESCRIPTION

This manual page documents the GNU version of cat ...

Hay aproximadamente una página completa de información sobre `cat`¹⁰. Pruebe correr “`man`” en este momento. No espere entender la página de manual que se le muestra. Las páginas de manual suponen un cierto conocimiento de Unix —conocimiento que tal vez Ud. no tenga en este momento. Cuando termine de leer la página, es probable que vea un bloque en video inverso al final de la página, parecido a “`--more--`” o a “`Line 1`”. Se trata del **pedido de más** (información), que pronto le será muy familiar.

En lugar de dejar escapar el texto fuera de los límites de la pantalla, `man` se detiene al final de cada página, y espera para ver que ha decidido hacer Ud.. Si desea seguir leyendo, oprima Barra espaciadora y avanzará una página. Si desea terminar con la lectura de la página del manual, oprima q. Regresará entonces al prompt del shell, que esperará hasta que Ud. escriba otro comando.

`man` provee además una función de búsqueda de palabras clave. Por ejemplo, digamos que Ud. está interesado en el tema PostScript, el lenguaje de control de impresoras desarrollado por Adobe. Si escribe “`man -k ps`” o “`man -k Postscript`”, recibirá como resultado una lista de todos los comandos, llamadas al sistema, y otras partes documentadas de Unix que contengan la palabra “`ps`” (o “`Postscript`”) en su nombre o descripción breve. Esto puede llegar a ser muy útil cuando quiere buscar una herramienta para hacer algo, pero no conoce su nombre —o si existe.

4.3 Almacenaje de la información

Los filtros son muy útiles cuando Ud. se ha transformado en un usuario experimentado, pero tienen un pequeño problema. ¿Cómo almacena Ud. la información. Seguramente, ¡no se espera que Ud. teclee todas las cosas cada vez que desea utilizar el programa! Por supuesto que no. Unix provee **archivos y directorios**.

Un directorio es como una carpeta, contiene hojas de papel, o archivos. Una carpeta suficientemente grande puede contener otras carpetas—puede haber directorios dentro de directorios. En Unix, se denomina sistema de archivo a la colección de archivos y directorios. Inicialmente en el sistema de archivo sólo existe un directorio, denominado “`root`”¹¹. Dentro de este directorio hay

¹⁰N. del T.: El ejemplo se ha mantenido en inglés, pues es muy probable que Ud. tenga instaladas las páginas de manual en dicho idioma, de manera predeterminada. Pruebe más tarde de instalar el conjunto de páginas de manual que están disponibles en castellano.

¹¹N. del T.: directorio *raíz*.

más directorios, y dentro de ellos hay archivos, y aún más directorios¹².

Cada archivo y cada directorio tiene un nombre. Nos referiremos a su **nombre corto**¹³ —que puede coincidir con el de otro archivo y/o directorio en alguna otra parte del sistema de archivo, y al **nombre largo**¹⁴ —que es único. Un nombre corto para un archivo puede ser **joe**, mientras que su **nombre completo** podría ser **/home/larry/joe**. El nombre completo se denomina usualmente **trayectoria**¹⁵. La trayectoria puede decodificarse como una secuencia de directorios. Por ejemplo, veremos como se lee **/home/larry/joe**:

```
/home/larry/joe
```

La barra inicial indica el directorio raíz.

Aquí esta el directorio denominado **home**. Está dentro del directorio raíz.

Este es el directorio **larry**, el cual está dentro de **home**.

joe está dentro de **larry**. Una trayectoria puede referirse tanto a un directorio como a un nombre de archivo, así que **joe** podría ser cualquiera de ellos. Todos los items *antes* del nombre corto deben ser directorios.

Una manera sencilla de visualizar todo esto es mediante un diagrama en árbol. Para ver el diagrama de un sistema típico Linux, mire la Figura 4.1. Debe Ud. notar que dicho diagrama no está completo —¡un sistema Linux completo tiene más de 8000 archivos!— y sólo muestra algunos de los directorios estándar. Por lo tanto, puede haber algunos directorios del diagrama que no estén en su sistema, y de hecho su sistema tendrá directorios que no aparecen en la figura.

4.3.1 Miremos los directorios con **ls**

Ahora que ya sabe de la existencia de archivos y directorios, debe haber también una manera de manipularlos. Por supuesto. El comando **ls** es uno de los más importantes, y lo que hace es listar los archivos. Si prueba a correr el comando “**ls**”, entonces verá:

```
/home/larry$ ls  
/home/larry$
```

Está bien, no se preocupe por no ver nada. Unix es intencionalmente callado: si no hay archivos para mostrar, no muestra nada (ni siquiera un “no hay archivos”). Por lo tanto, la ausencia de salida es la manera de **ls** de decir que no encontró ningún archivo.

Pero hace un momento dije que hay más de 8000 archivos por allí: ¿dónde están? Acaba de encontrar el concepto de directorio “actual”. Como puede ver en el prompt, su directorio actual es **/home/larry** y allí no hay archivos. Si desea la lista de archivos de un directorio más poblado, pruebe con el directorio raíz:

```
/home/larry$ ls /
```

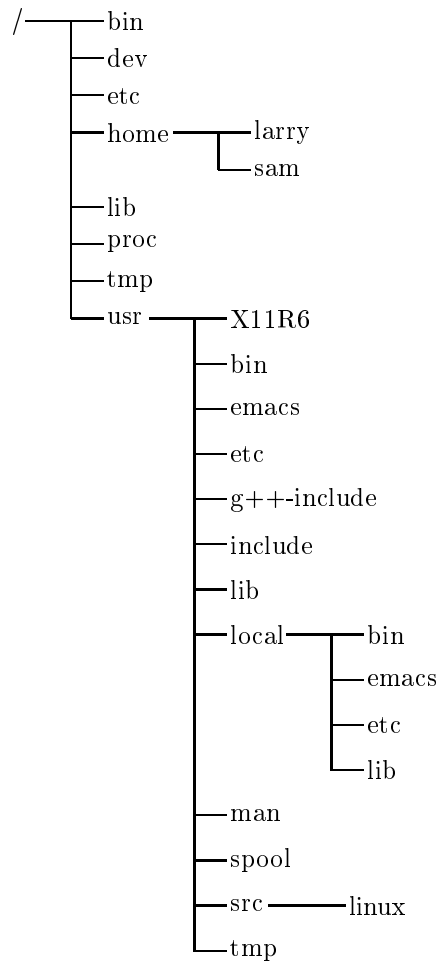
¹²Puede haber o no un límite a la “profundidad” a la cual puede llegar el sistema de archivo. (En mi caso nunca la alcancé—uno fácilmente puede tener directorios de 10 niveles de profundidad.)

¹³N. del T.: Nombre relativo del archivo o directorio.

¹⁴N. del T.: Nombre completo o absoluto.

¹⁵N. del T.: *trayectoria* del inglés *path*.

Figura 4.1 Típico árbol (podado) de directorios Unix.



```
bin      etc      install  mnt      root     user      var
dev      home     lib      proc     tmp      usr       vmlinux
/home/larry$
```

En el comando anterior, “ls /”, el directorio (“/”) es lo que se denomina un **parámetro**. La primera palabra de un comando es el nombre del comando, y cualquier cosa que le siga son sus parámetros. Los parámetros generalmente modifican la forma en la que actúa un programa—para `ls`, el parámetro le dice de cual directorio desea Ud. la lista de archivos. Algunos comandos tienen parámetros especiales denominados opciones o interruptores¹⁶ Para ver esto, pruebe:

```
/home/larry$ ls -F /
bin/      etc/      install/  mnt/      root/     user/     var@
dev/      home/    lib/      proc/     tmp/      usr/      vmlinux
/home/larry$
```

¹⁶N. del T.: *interruptores* del inglés *switches*.

El “-F” es una **opción**. Las opciones son clases especiales de parámetros que comienzan con un guión y modifican el funcionamiento del programa. Para “ls”, “-F” es una opción que le permite a Ud. ver cuales de las entradas son directorios, cuales son archivos especiales, cuales programas, y cuales son archivos normales. Cualquiera que termina con una barra es un directorio. Trataremos con más detalle las características de ls más adelante. ¡Es un programa sorprendentemente complejo!

Por ahora, hay dos lecciones que se deben aprender. Primero, debe aprender que es lo que hace ls. Pruebe unos cuantos otros directorios de los que se muestran en el árbol de la Figura 4.1, y fíjese en su contenido. Naturalmente, algunos estarán vacíos, y otros tendrán muchos archivos dentro. Le sugiero que utilice “ls” con y sin la opción “-F”. Por ejemplo, “ls /usr/local” debe aparecer más o menos así:

```
/home/larry$ ls /usr/local
archives  bin      emacs    etc      ka9q     lib      tcl
/home/larry$
```

La segunda lección es más general. Muchos comandos Unix son como ls. Tienen opciones, que en general consisten de un carácter precedido por un guión, y tienen parámetros. A diferencia de ls, algunos comandos *requieren* ciertos parámetros y/u opciones. Para mostrar como se usan los comandos, utilizaremos el siguiente formato:

ls [-aRF] [*directorio*]

En general, y desde ahora en adelante, utilizaré plantillas de comandos como la mostrada, antes de introducir comandos nuevos para Ud.. La primer palabra es el comando (en este caso ls). A continuación del comando siguen todos los parámetros. Los parámetros opcionales se encerrarán entre corchetes (“[” y “]”). Las meta-variables están *inclinadas*—son palabras que toman el lugar de los parámetros reales. (Por ejemplo, más arriba puede Ud. ver que dice *directorio*, lo cual debe ser reemplazado por el nombre de un directorio real.)

Las opciones son un caso especial. Están encerradas entre corchetes, pero Ud. puede utilizar una sola de ellas sin necesidad de emplear todas juntas. Por ejemplo, con las tres opciones enumeradas más arriba para el caso de ls, Ud. tiene ocho maneras posibles de correr el comando: con o sin cada una de las opciones. (Contraste “ls -R” con “ls -F”.)

4.3.2 El directorio actual y cd

pwd

El uso de los directorios puede ser insoportable si se tiene que teclear la trayectoria completa cada vez que se quiere acceso a un directorio. En lugar de ello, los shell de Unix tienen una característica denominada directorio “actual”, “presente”, o “de trabajo”. Es muy probable que su configuración de shell muestre dicho directorio como parte del prompt: **/home/larry**. Si no lo hace, pruebe el

comando **pwd**, que proviene de la sigla de las palabras **p**resent **w**orking **d**irectory¹⁷. (Algunas veces el prompt muestra el nombre de la máquina. Esto sólo es útil realmente en un entorno de red en el cual hay una gran cantidad de máquinas diferentes.)

```
mousehouse>pwd
/home/larry
mousehouse>
```

`cd` [*directorio*]

Como puede Ud. ver, **pwd** le dice cuál es su directorio actual¹⁸— se trata de un comando muy simple. La mayoría de los comandos actúa, de forma predeterminada, sobre el directorio actual. Podemos cambiar nuestro directorio actual mediante el comando **cd**. Por ejemplo, pruebe:

```
/home/larry$ cd /home
/home$ ls -F
larry/      sam/        shutdown/  steve/     user1/
/home$
```

Si Ud. omite el parámetro opcional *directorio*, se lo retornará a su directorio raíz personal, o directorio original. De otro modo, **cd** lo cambiará al directorio especificado. Por ejemplo:

```
/home$ cd
/home/larry$ cd /
/$ cd home
/home$ cd /usr
/usr$ cd local/bin
/usr/local/bin$
```

Como puede ver, **cd** le permite expresar el directorio de destino con trayectorias absolutas o relativas. Una **trayectoria absoluta** comienza con una “/” y especifica todos los directorios que existen antes del archivo que Ud. desea. Una **trayectoria relativa** está referida a su directorio actual. En el ejemplo anterior, cuando estaba en “/usr”, hice un movimiento relativo a “**local/bin**”—“local” es un directorio que existe bajo “/usr”, y “bin” es un directorio bajo “local”. (“**cd home**” también fue un cambio de directorio relativo.)

Existen dos directorios que se utilizan solamente para trayectorias relativas: “.” y “..”. El directorio “.” se refiere al directorio actual, y “..” es el directorio anterior: son “abreviaturas” de directorio, y existen en *todos* los directorios, aunque no cuadran bien en la metáfora de “carpeta dentro de carpeta”. Aún el directorio raíz tiene un directorio anterior: él es su propio directorio anterior.

¹⁷N. del T.: Directorio de trabajo actual.

¹⁸En este libro verá todos los términos en uso: directorio de trabajo presente, directorio actual, o directorio de trabajo. Prefiero “directorio actual”, aunque a veces recurra a las otras formas a causa de razones de estilo.

El archivo `./chapter-1` viene a ser el archivo **chapter-1** que está en el directorio actual. Ocasionalmente, será necesario que Ud. coloque explícitamente el “`./`” para que ciertos comandos funcionen, aunque esto es raro. En la mayoría de los casos, dará igual poner `./chapter-1` o **chapter-1**.

El directorio “`..`” es más útil cuando queremos “retroceder”:

```
/usr/local/bin$ cd ..
/usr/local$ ls -F
archives/  bin/      emacs@    etc/      ka9q/     lib/      tcl@
/usr/local$ ls -F ../src
cweb/      linux/    xmris/
/usr/local$
```

En este ejemplo, me cambié al directorio anterior mediante “`cd ..`” y luego listé el directorio `/usr/src` desde `/usr/local` utilizando el nombre `../src`. Debe advertir que si hubiese estado en `/home/larry`, ejecutar el comando “`ls -F ../src`” no me habría servido al propósito de ver lo que hay en `/usr/src`.

El directorio “`~/`” es un alias para su directorio personal:

```
/usr/local$ ls -F ~/
/usr/local$
```

Puede ver que aún no hay nada en su directorio personal. “`~/`” será más útil a medida que aprendamos más acerca de como manipular los archivos.

4.3.3 Creación y borrado de directorios

mkdir *directorio1* [*directorio2* ... *directorioN*]

En Unix, crear directorios es extremadamente simple, y puede ser una muy útil herramienta de organización. Para crear un nuevo directorio, utilice el comando **mkdir**. Por supuesto, **mkdir** viene de **make directory**¹⁹.

Con un pequeño ejemplo, veamos como funciona esto:

```
/home/larry$ ls -F
/home/larry$ mkdir report-1993
/home/larry$ ls -F
report-1993/
/home/larry$ cd report-1993
/home/larry/report-1993$
```

mkdir puede tomar más de un parámetro, a los que interpreta como nombres de directorios que debe crear. Puede especificarlos mediante su nombre relativo o absoluto; **report-1993** es un caso de ejemplo con trayectoria relativa.

¹⁹N. del T.: Crear directorio.

```
/home/larry/report-1993$ mkdir /home/larry/report-1993/chap1 ~/report-1993/chap2
/home/larry/report-1993$ ls -F
chap1/  chap2/
/home/larry/report-1993$
```

`rmdir` *directorío1* [*directorío2* ... *directoríoN*]

El opuesto de `mkdir` es `rmdir` (**re**move **di**rectory²⁰). `rmdir` trabaja exactamente igual que `mkdir`.

Un ejemplo de `rmdir` es:

```
/home/larry/report-1993$ rmdir chap1 chap3
rmdir: chap3: No such file or directory
/home/larry/report-1993$ ls -F
chap2/
/home/larry/report-1993$ cd ..
/home/larry$ rmdir report-1993
rmdir: report-1993: Directory not empty
/home/larry$
```

Como puede apreciar, `rmdir` se niega a borrar un directorio que no existe, y tampoco permite eliminar directorios que tengan algo dentro. (Recuerde que **report-1993** tiene un subdirectorio, **chap2**, dentro de él) Existe un tema interesante para pensar, entonces: ¿qué pasa si intenta eliminar su directorio actual? Averigüémoslo:

```
/home/larry$ cd report-1993
/home/larry/report-1993$ ls -F
chap2/
/home/larry/report-1993$ rmdir chap2
/home/larry/report-1993$ rmdir .
rmdir: .: Operation not permitted
/home/larry/report-1993$
```

Otra situación del mismo estilo, es que sucede si intenta eliminar el directorio anterior al directorio actual. Bien, esto no es un problema nuevo: el directorio anterior al actual no está vacío (existe el actual en él), así que no puede ser borrado.

4.4 Información en movimiento

Todos este asunto de los directorios está muy lindo, pero no son de ninguna ayuda, a menos que Ud. tenga algún sitio donde almacenar sus datos. Los dioses del Unix vieron este problema y lo solucionaron haciendo que los usuarios tengan **archivos**²¹

²⁰N. del T.: Eliminar directorio.

²¹N. del T.: También denominados *ficheros*.

Aprenderemos más acerca de la creación y edición de archivos en los próximos capítulos.

Los comandos básicos para manipular archivos en Unix son `cp`, `mv`, y `rm`. Sus nombres provienen de `copy`²², `move`²³, y `remove`²⁴, respectivamente.

4.4.1 `cp` como un monje

`cp [-i] fuelle destino`

`cp [-i] archivo1 archivo2 ... archivoN directorio-de-destino`²⁵

`cp` es una utilidad muy práctica en Unix, y además muy poderosa. Permite que una persona pueda copiar más información en un segundo que lo que podía copiar un monje del siglo XIV en todo un año.

Debe ser muy cuidadoso con `cp` si no dispone de una cantidad importante de espacio en disco. A nadie le hace gracia ver el mensaje “Disk full”²⁶ cuando está trabajando con archivos importantes. `cp` también puede sobrescribir archivos existentes sin previo aviso —trataremos este tema más adelante.

Bien, hablaremos primero acerca de la primer línea en la plantilla del comando. El primer parámetro de `cp` es el nombre del archivo que hay que copiar, el segundo es el lugar donde se desea depositar la copia. Puede obtener una copia con un nombre distinto, o una en un directorio diferente. Veamos algunos ejemplos:

```
/home/larry$ ls -F /etc/passwd
/etc/passwd
/home/larry$ cp /etc/passwd .
/home/larry$ ls -F
passwd
/home/larry$ cp passwd frog
/home/larry$ ls -F
frog  passwd
/home/larry$
```

El primer comando `cp` que corrí tomó el archivo `/etc/passwd`, que contiene los nombres de todos los usuarios en un sistema Unix, y lo copió en mi directorio raíz. `cp` no borra el archivo fuente, así que no hice nada que pudiera dañar el sistema. O sea que ahora existen dos copias del contenido del archivo `/etc/passwd`, ambas se denominan `passwd`, pero una está en el directorio `/etc` y la otra en `/home/larry`.

²²N. del T.: Copiar.

²³N. del T.: Mover.

²⁴N. del T.: Quitar, borrar.

²⁵`cp` necesita de dos renglones en su plantilla porque el significado del segundo parámetro depende de la cantidad de parámetros.

²⁶N. del T.: Disco lleno.



Luego, creamos una *tercera* copia de `/etc/passwd` cuando tecleamos “`cp passwd frog`” — ahora hay tres copias: `/etc/passwd`, `/home/larry/passwd` y `/home/larry/frog`. El contenido de esos tres archivos es el mismo, sus nombres de archivo difieren.

`cp` puede copiar archivos entre directorios si el primer parámetro es un archivo y el segundo es un directorio. En este caso, el nombre simple (sin el path) de destino coincidirá con el original, para cada archivo.

También se puede copiar un archivo y cambiar su nombre de destino en una sola operación, caso que se da cuando ambos parámetros son nombres de archivo. Aquí reside uno de los peligros de `cp`. Si tecleara “`cp /etc/passwd /etc/group`”, el comando `cp` crearía normalmente un archivo nuevo con el contenido idéntico al de `/etc/passwd` y con el nombre `/etc/group`. Por lo tanto, si `/etc/group` ya existiera, `cp` destruiría el contenido del antiguo sin darle la oportunidad de guardarlo. (Ni siquiera escribirá un mensaje en el cual le advierta que está por destruir un archivo al copiarle otro encima.)

Echemos una mirada a otro ejemplo de `cp`:

```
/home/larry$ ls -F
frog  passwd
/home/larry$ mkdir passwd_version
/home/larry$ cp frog passwd passwd_version
/home/larry$ ls -F
frog      passwd      passwd_version/
/home/larry$ ls -F passwd_version
frog  passwd
/home/larry$
```

¿Cómo usé `cp` en este caso? Evidentemente, `cp` puede tomar *más* de dos parámetros, como se puede apreciar en la segunda línea de plantillas del comando. Lo que el comando anterior realizó es copiar los archivos que aparecen listados (**frog**, y **passwd**) al directorio **passwd_version**. De hecho, `cp` puede tomar cualquier número de parámetros: interpreta los primeros $n - 1$ como los nombres de archivo que debe copiar, y el $n^{\text{ésimo}}$ parámetro como el nombre del directorio al cual se deben copiar los anteriores.

No puede renombrar archivos cuando copia más de uno a la vez—siempre mantienen su nombre simple. Esto nos lleva a una pregunta interesante. ¿Qué es lo que pasaría si tecleamos “`cp frog passwd toad`”, donde **frog** y **passwd** existen, y **toad** no es un directorio? Inténtelo y lo sabrá.

4.4.2 La poda con `rm`

`rm [-i] archivo1 archivo2 ... archivoN`

Ahora que hemos aprendido como crear millones de archivos con `cp` (y créame, muy pronto encontrará nuevas maneras de crear más archivos), sería útil aprender la manera de borrarlos. De



SLOW

hecho, es muy simple: el comando que anda necesitando es `rm`, y funciona como Ud. se lo esperaría: a cualquier archivo cuyo nombre se pasa como parámetro a `rm` se lo borra.

Por ejemplo:

```
/home/larry$ ls -F
frog          passwd          passwd_version/
/home/larry$ rm frog toad passwd
rm: toad: No such file or directory
/home/larry$ ls -F
passwd_version/
/home/larry$
```

Como puede Ud. apreciar, `rm` es extremadamente espartano. No sólo no pide su confirmación, sino que borra cosas aún en el caso en que haya errores en la línea de comandos. Esto bien puede ser peligroso. Considere la diferencia entre estos dos comandos:

```
/home/larry$ ls -F
toad frog/
/home/larry$ ls -F frog
toad
/home/larry$ rm frog/toad
/home/larry$
```

y este otro:

```
/home/larry$ rm frog toad
rm: frog is a directory
/home/larry$ ls -F
frog/
/home/larry$
```



Como puede ver, la diferencia en *un* caracter hace una diferencia importante en la salida del comando. ¡Es vital que Ud. controle lo que ha escrito en sus líneas de comando antes de presionar **Intro**!

4.4.3 Sería interesante tener un rastrillo

```
mv [-i] nombre-viejo nombre-nuevo
mv [-i] archivo1 archivo2 ... archivoN nuevo-directorio
```

Por último, el otro comando del que debe cuidarse es `mv`. `mv` se parece un montón a `cp`, excepto que borra el archivo original después de copiarlo. Se puede entender como la utilización conjunta de `cp` y `rm`. Veamos que podemos hacer:

```
/home/larry$ cp /etc/passwd .
/home/larry$ ls -F
```

```
passwd
/home/larry$ mv passwd frog
/home/larry$ ls -F
frog
/home/larry$ mkdir report
/home/larry$ mv frog report
/home/larry$ ls -F
report/
/home/larry$ ls -F report
frog
/home/larry$
```

Como puede ver, `mv` renombra un archivo (que es el primer parámetro) si el segundo parámetro es un archivo. Si el segundo parámetro es un directorio, `mv` moverá el archivo al nuevo directorio, manteniendo el mismo nombre simple.

Debe Ud. ser muy cuidadoso con `mv` —el comando no controla si el archivo ya existe, y borrará cualquier archivo que hubiera con el nombre de destino. Por ejemplo, si ya tengo un archivo de nombre **frog** existente en mi directorio **report**, el comando “`mv frog report`” borrará el archivo **/report/frog** y lo reemplazará con **/frog**.

De hecho, existe una manera de lograr que tanto `rm`, `cp`, y `mv` le consulten a Ud. antes de eliminar archivos. Los tres comandos nombrados aceptan la opción “`-i`” a tal efecto. Si Ud. utiliza un **alias** puede hacer que el shell ejecute: “`rm -i`” automáticamente cuando teclee: “`rm`”. Aprenderá más de estos temas luego en la Sección 9.1.3 en la página 92.



SLOW

Capítulo 5

El sistema de ventanas X

Lo mejor de los estándares es que haya tantos para elegir.

Andrew S. Tanenbaum



Este capítulo sólo es aplicable para aquellos que usen el sistema de ventanas X. Si se encuentra con una pantalla con múltiples ventanas, colores o un cursor que sólo se mueve con el ratón, está usted usando X. (Si su pantalla contiene caracteres blancos sobre fondo negro no está usando X. Para arrancarlo, refiérase a la Sección 5.1.)

5.1 Ejecución y salida del sistema de ventanas X

5.1.1 Ejecución de X

Incluso cuando X no arranque de forma automática durante el ingreso en el sistema, es posible arrancarlo desde la línea de comandos del intérprete en modo texto. Hay dos comandos que arrancan X, bien `startx` o bien `xinit`. Debe intentar `startx` primero. Si el intérprete de comandos se queja de que ese comando no existe, entonces pruebe con `xinit` a ver si arranca X de ese modo. Si no funciona ninguno de los comandos anteriores, puede que no tenga X instalado en su sistema—refiérase a la documentación local de su distribución.

Si el comando se ejecuta pero regresa al intérprete en modo texto tras un rato, X está instalado pero no ha sido configurado todavía. En ese caso debe consultar la documentación que acompañaba su distribución en lo referente a como configurar X.

5.1.2 Saliendo de X

Dependiendo de cómo se haya configurado X, puede haber dos modos diferentes de salir de X. El primero es que su gestor de ventanas controle la ejecución de X. En ese caso, deberá salir de X usando un menú (ver Sección 5.4.8 en la página 41). Para ver el menú, basta apretar un botón del ratón sobre el fondo.

La entrada del menú que busca debe ser algo así como “Exit Window Manager”¹ o “Exit X”². Trate de encontrar esa entrada (puede que haya más de un menú—intente usar diferentes botones) y elíjala.

El otro método utiliza una ventana `xterm` especial para controlar X. En este caso, seguramente existirá una ventana cuyo título sea “login”³ o “system xterm”⁴. Para salir de X, mueva el cursor a esa ventana y escriba “exit”⁵.

Si X arrancó automáticamente tras su ingreso en el sistema, uno de estos métodos probablemente le sacará de él. Basta conectarse de nuevo para regresar a él. Si ejecutó X de forma manual, al salir volverá al interprete de comandos en modo texto. (Si quiere salir del sistema, bastará con escribir “logout”.)

5.2 ¿Qué es el sistema de ventanas X?

El Sistema de Ventanas X es un método de trabajo gráfico y distribuido, desarrollado principalmente en el Instituto Tecnológico de Massachusetts. Actualmente está a cargo de un consorcio de fabricantes (debidamente llamado “El Consorcio X”) y es mantenido por ellos.

El Sistema de Ventanas X (que a partir de ahora abreviaremos como “X”⁶ tiene revisiones cada pocos años, conocidas como lanzamientos. La última revisión ha sido X11R6, o sexto lanzamiento⁷. El número 11 indica la versión oficial pero no ha habido cambios en los últimos años y tampoco hay planes para cambiarla en un futuro próximo.

Al ser cliente y servidores programas diferentes, es posible ejecutar cada uno en *en máquinas completamente diferentes*. Además de constituir un método estándar para aplicaciones gráficas, es posible ejecutar un programa en una máquina remota (¡incluso al otro lado del país, si quiere!) y que los resultados aparezcan en la estación de trabajo que tiene enfrente suyo.

Un tercer concepto con el que debe familiarizarse es el de **gestor de ventanas**. El gestor de ventanas es un cliente especial que le dice al servidor en que posición deben colocarse las diferentes ventanas y permite al usuario moverlas. El servidor, por sí mismo, no interacciona con el usuario. Se trata de un medio que conecta el usuario y el cliente.

5.3 ¿Qué es esto que hay en mi pantalla?

Al arrancar X, varios programas son ejecutados. Primero, arranca el servidor. Luego, generalmente arrancan varios clientes. Desgraciadamente no hay un estándar común entre las diferentes distribu-

¹N.T.: En inglés, salir del gestor de ventanas

²N.T.: En inglés, salir de X.

³N.T.: En inglés, ingreso en el sistema o autenticación.

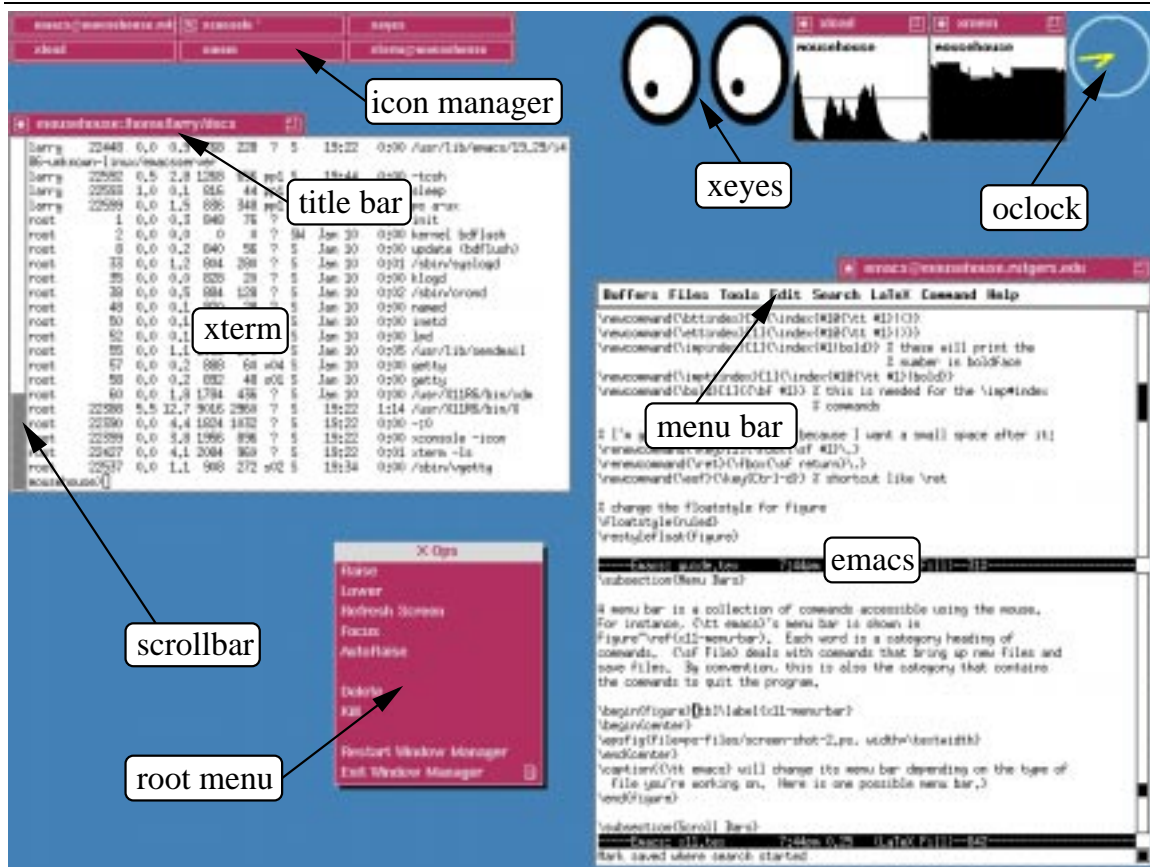
⁴N.T.: En inglés, xterm del sistema o terminal X del sistema.

⁵N.T.: En inglés, salir.

⁶Existen varias formas de referirse al Sistema de Ventanas X. Una forma que aunque común es incorrecta es “X Window” o “Ventanas X”.

⁷N.T.: La palabra inglesa es “release”. Por ello los lanzamientos se diferencian añadiendo una R y el número correspondiente.

Figura 5.1 Ejemplo comentado de una pantalla X estándar. En este ejemplo el usuario está ejecutando `twm`. El reloj estándar ha sido sustituido por uno transparente llamado `oclock`.



ciones. Es bastante probable que entre los clientes se encuentre un gestor de ventanas, bien `fvwm` o `twm`, un intérprete de comandos, `xterm`, y un reloj, `xclock`.

5.3.1 XClock

`xclock` [-digital] [-analog] [-update *segundos*] [-hands *color*]

Primero explicaremos el más simple: `xclock` funciona exactamente como se puede esperar. Marca los segundos, minutos y horas en una ventana pequeña.

Ni usar los botones del ratón, ni escribir en la ventana `xclock` produce efecto alguno— eso es **todo** lo que hace. ¿O quizá no? En realidad hay varias opciones diferentes que permiten hacer que el programa se comporte de modos diferentes. Así, por ejemplo, “`xclock -digital`” crea un reloj digital. “`xclock -update 1`” hace que el segundero se mueva cada segundo y “`xclock -update 5`”

hace que se mueva cada cinco segundos.

Para obtener más información sobre las opciones de `xclock` puede consultar la página del manual correspondiente—“`man xclock`”. Si piensa ejecutar varias copias de `xclock`, probablemente debería consultar la Sección 6.4 (Multitarea) para saber como se ejecutan a la vez que sus otros programas. (Si ejecuta una copia de `xclock` en primer plano—que es la forma usual en la que se ejecuta un programa—y quiere salir de él, basta teclear `Ctrl-c`.)

5.3.2 XTerm

La ventana que muestra el prompt del intérprete de comandos (podrá ver algo parecido a `/home/larry$`) es controlada por un programa llamado `xterm`. `xterm` es un programa engañosamente complicado. A primera vista, no parece que haga mucho pero realmente hace muchísimo trabajo. `xterm` emula una terminal de forma que las aplicaciones de modo texto de Unix funcionen correctamente. También mantiene en memoria información de forma que se pueden consultar comandos usados previamente. (Para ver como hace uso de esto, refiérase a la Sección 5.6.3.)

Gran parte de este libro está dedicada a aprender como usar el interprete de comandos de Unix que se encuentra dentro de la ventana `xterm`. Para poder escribir en la `xterm`, *generalmente* es necesario mover el cursor (que posiblemente aparece en forma de “X” o de flecha) al interior de la ventana `xterm`. De todos modos este comportamiento es dependiente del gestor de ventanas.

Uno de los modos en los que se puede arrancar un programa bajo X es a través de una `xterm`. Al tratarse los programas X de programas Unix estándar, pueden ser arrancados a través de un intérprete de comandos como el de las `xterms`. Para evitar bloquear la `xterm` ejecutando un programa largo desde ella, generalmente se prefiere arrancar los programas X en segundo plano. Para consultar este tema puede referirse a la Sección 6.4.

5.4 Gestores de ventanas

Bajo Linux, los gestores de ventanas más comunes son dos. El primero, llamado `twm` que viene de “Tab Window Manager”. Es un programa mayor que el segundo, `fvwm`. (`fvwm` viene de “F(?) Virtual Window Manager”—el autor nunca explicó que significaba la “f”⁸). Ambos gestores de ventanas son altamente configurables lo cual impide dar instrucciones específicas sobre que teclas usar para hacer algo en particular.

Si le interesa saber algo sobre la configuración de `twm`, refiérase a la Sección 9.2.1. La configuración de `fvwm` se describe en la Sección 9.2.2.

5.4.1 Cuando se crean nuevas ventanas

Hay tres tipos de cosas que un gestor de ventanas puede hacer cuando se crea una nueva ventana. Es posible configurar el gestor de ventanas de manera que aparezca la forma de la ventana, permitiéndole

⁸N. del T.: En el Linux Journal N.º 43, Noviembre 1997, hay una entrevista a Robert Nation, págs. 46-47, y éste dice que la “F” está por “Feeble” (Gracias José Luis Gurpegi por la aclaración.)

colocarla en la posición que prefiera en la pantalla. Este modo se llama **colocación manual**. Si ve aparecer la forma de la ventana, puede moverla con el ratón y colocarla presionando el botón izquierdo.

A veces es posible que el gestor de ventanas las coloque por sí mismo. Se trata de la **colocación aleatoria**.

Finalmente, a veces, la aplicación pregunta por el lugar preciso en el que el quiere que se sitúe. También el gestor de ventanas puede configurarse de forma que siempre sitúe las ventanas de ciertas aplicaciones en lugares fijos. (Así por ejemplo, se puede especificar que `xclock` siempre aparezca en la esquina superior derecha de la pantalla.)

5.4.2 Foco

El gestor de ventanas controla otras cosas importantes. Lo que probablemente le interesa más es el **foco**. El foco del servidor determina que ventana recibe los caracteres introducidos mediante el teclado. Generalmente, en X, el foco es determinado por la posición del cursor del ratón. Si mueve el cursor del ratón dentro de una ventana `xterm`⁹, esa `xterm` recibe los caracteres tecleados. Se trata de un comportamiento diferente al de otros sistemas de ventanas como, Windows de Microsoft, OS/2, o Macintosh, donde debe pinchar con el ratón en una ventana para que adquiera el foco. Generalmente bajo X, si el cursor del ratón se sale de una ventana, se pierde el foco y le resultará imposible escribir en ella.

Debe notar, sin embargo, que es posible configurar tanto `twm` como `fvwm` de forma que para cambiar el foco necesite pinchar la ventana correspondiente, y pinchar fuera para desenfocarla, de forma que se comporten igual que Windows de Microsoft. Una de dos, trate de averiguar como funciona su gestor de ventanas probando o consulte la documentación local.

5.4.3 Moviendo ventanas

Otro aspecto de X que es altamente configurable es el método para mover ventanas. Yo tengo configurados tres métodos para mover ventanas en `twm`. El más obvio es mover el cursor del ratón sobre la **barra del título** y arrastrar la ventana. Desgraciadamente, la configuración permite definir el movimiento usando cualquiera de los tres botones¹⁰ (Para arrastrar, sitúe el cursor sobre la barra del título, y mantenga presionado el botón del ratón correspondiente mientras se mueve). Lo más probable es que su configuración use el botón *izquierdo* para mover ventanas.

Otra forma de mover ventanas puede ser manteniendo una tecla pulsada mientras mueve el ratón. Así, en *mi* caso, si mantengo pulsada la tecla **Alt** y muevo el cursor sobre una ventana, puedo arrastrarla usando el boton izquierdo de mi ratón.

Por supuesto, para ver como funciona en su caso puede probar igual que antes o referirse a la documentación local. Alternativamente, si quiere tratar de interpretar los ficheros de configuración de los gestores de ventanas puede referirse a la Sección 9.2.1 para `twm` o 9.2.2 para `fvwm`.

⁹¡Puede ejecutar varias copias de `xterm` al mismo tiempo!

¹⁰Muchos ratones de PCs tienen dos botones únicamente. En este caso, generalmente es posible emular el botón central presionando ambos botones simultáneamente.

5.4.4 Profundidad

Dado que las ventanas pueden solaparse en X, necesitamos el concepto de **profundidad**. A pesar de que tanto las ventanas como la pantalla son bidimensionales, unas ventanas pueden estar delante de otras, de forma que cubren total o parcialmente aquella situada detrás.

Existen varias operaciones que manejan la profundidad:

- **Subir** o poner una ventana delante. Generalmente se hace pinchando con alguno de los botones en la barra de título de la ventana en cuestión. El botón usado depende de la configuración del gestor de ventanas. (Es posible que sean varios los botones que produzcan este efecto.)
- **Bajar** o empujar una ventana hacia atrás. Generalmente se hace pinchando igualmente en la barra de título pero usando otro botón. También es posible configurar el gestor de ventanas de forma que pinchar traiga la ventana hacia delante siempre que haya algo encima de ella y la lleve hacia atrás si no hay nada delante.
- **Alzar en ciclo**, de forma que el gestor de ventanas va alzando las ventanas una a una, en orden.

5.4.5 Iconizar

Existen otras operaciones que pueden tapar ventanas o hacerlas desaparecer completamente. La primera es “iconización”. Dependiendo del gestor de ventanas que use, este proceso puede realizarse de varias formas. Con `twm`, muchas personas configuran un **gestor de iconos**. Se trata de una ventana especial que contiene una lista con las demás ventanas en uso en la pantalla. Pinchando con el ratón en uno de los nombres (o, dependiendo de la configuración en uno de los botones) la ventana desaparece—es iconizada. La ventana está aún activa, pero no puede verse. Pinchando otra vez en el gestor de iconos, la ventana vuelve a aparecer en la pantalla.

Se trata de algo muy útil. Se podría, por ejemplo tener `xterm` remotas en diferentes máquinas que use ocasionalmente al mismo tiempo. Dado, sin embargo, que sólo las usa raramente, puede mantenerlas iconizadas mientras trabaja con un pequeño grupo. El único problema es que resulta fácil “perder” alguna ventana. Generalmente eso lleva a crear nuevas ventanas que dupliquen la funcionalidad de las que están iconizadas.

Otros gestores de ventanas crean verdaderos iconos en la parte inferior de la pantalla o simplemente en la ventana raíz.

5.4.6 Variando el tamaño

Existen diversos métodos para variar el tamaño de las ventanas bajo X. De nuevo, esto depende de su gestor de ventanas y de cómo haya sido configurado. El método al que están acostumbrados los usuarios de Windows de Microsoft consiste en pinchar en el borde de la ventana y arrastrarlo. Si las ventanas que produce su gestor tienen un borde gordo que hace que el cursor del ratón cambie al pasar por encima de él, seguramente también puede usar ese método.

Otra forma es crear un botón de “variación de tamaño” en la barra del título. Es el botón visible en la parte derecha de las barras de los títulos de la Figura 5.3. Para usarlo, pinche con el botón izquierdo del ratón en él y manteniéndolo presionado, muévalo fuera de la ventana para variar el tamaño. Para fijar el nuevo tamaño basta soltar el botón del ratón.

5.4.7 Maximización

La mayoría de los gestores de ventanas soportan maximización. En `twm`, por ejemplo, puede maximizar la altura de una ventana, su ancho o ambos a la vez. Aunque `twm` llama a este proceso “zooming” yo prefiero usar maximización. Las aplicaciones pueden responder de modo diferente a un cambio de tamaño. (Por ejemplo, `xterm` no cambia el tamaño de letra pero aumenta el espacio de trabajo.)

Desgraciadamente no hay un modo estándar de maximizar ventanas.

5.4.8 Menús

Los gestores de ventanas también proporcionan al usuario un sistema de menús para realizar tareas de forma rápida una y otra vez. Por ejemplo, podría tener una opción que ejecute Emacs de forma automática o una `xterm` adicional. De ese modo no tendría que ejecutar el comando dentro de una `xterm`—esto es especialmente bueno si no hay ninguna `xterm` ejecutándose en la que pueda escribir para ejecutar un programa nuevo!

De forma general, se puede acceder a los menús pinchando en la ventana raíz, que es una ventana inmóvil que siempre está debajo de las demás. Por defecto esta ventana es de color gris, pero puede tener cualquier aspecto.¹¹ Si quiere probar a ver un menú, pinche y mantenga cualquier botón del ratón sobre la ventana raíz. Debería aparecer un menú. Para seleccionar una opción, mantenga presionado el botón y mueva el ratón hasta marcar la opción deseada. Entonces suelte el botón.

5.5 Atributos X

Hay muchos programas que usan X. Algunos, como `emacs`, puede ejecutarse **tanto** en modo texto **como** en una ventana X que él mismo crea automáticamente. Pero en general, la mayoría de los programas para X sólo funcionan bajo X.

5.5.1 Geometría

Existe una serie de cosas que son comunes a todos los programas que se ejecutan bajo X. En X el concepto de **geometría** engloba el lugar y el tamaño de la ventana. La geometría de una ventana tiene cuatro componentes, a saber:

¹¹Un programa divertido que puede probar se llama `xfishtank`. Este programa crea un pequeño acuario en el fondo de su pantalla.

- La dimensión horizontal, generalmente medida en pixels. (Un pixel es la unidad más pequeña que puede colorearse y representarse en la pantalla. Muchas configuraciones X en PCs basados en un procesador Intel tienen una resolución de 1024 pixels horizontalmente por 768 pixels verticalmente.) Algunas aplicaciones como `xterm` o `emacs`, miden su tamaño en caracteres. (Por ejemplo, ochenta caracteres por línea.)
- La dimensión vertical, también medida en pixels generalmente. Al igual que antes, en algunos casos se puede medir en líneas.
- La distancia horizontal a uno de los bordes de la pantalla. Por ejemplo, `+35` significa a treinta y cinco pixels desde la izquierda. Sin embargo, `-50` significa que el extremo derecho de la ventana está a cincuenta pixels del borde derecho de la pantalla. Generalmente no es posible lanzar una ventana fuera de la pantalla, aunque si se puede mover fuera. (La principal excepción es cuando la ventana es demasiado grande.)
- La distancia vertical desde la parte superior o inferior de la pantalla; la distancia se mide desde la parte superior si es positiva o desde la inferior si es negativa.

Los cuatro componentes se unen formando la cadena de geometría: `503x73-78+0`. (Esto marca la geometría de una ventana de 503 pixels de largo, 73 pixels de alto, situada cerca de la esquina superior derecha de la pantalla.) O de otro modo, tendremos *tamaño_h × tamaño_v ± situación_h ± situación_v*.

5.5.2 Presentación

Toda aplicación X tiene asociada una presentación. Ésta determina cuál es la pantalla controlada por el servidor X. La presentación está formada por tres elementos:

- El nombre de la máquina en la que se ejecuta el servidor. En el caso de sistemas Linux aislados, el nombre de la máquina que ejecuta el servidor coincide con la que ejecuta los clientes. En ese caso se puede omitir el nombre por completo.
- El número del servidor de dicha máquina que debe responder a las peticiones de los clientes. Dado que una máquina puede estar corriendo diferentes servidores a la vez (es algo improbable pero posible en máquinas Linux) cada uno tiene que tener un número propio.
- El número del monitor. X permite que un servidor controle más de un monitor a la vez. Por ejemplo, puede que alguien quiera tener más espacio, de forma que use dos monitores a la vez. Para evitar que el rendimiento de la máquina sea muy malo, en vez de tener dos servidores, hacen que sea el mismo servidor el que controle las dos máquinas.

Si ponemos estas tres cosas juntas obtendremos: *máquina:número-de-servidor.número-de-pantalla*.

Por ejemplo, en `mousehouse`, todas mis aplicaciones tienen la presentación fijada en `:0.0`, lo que significa que la salida aparece en la primera pantalla del primer servidor local. Sin embargo si uso una computadora remota, tendría que ser `mousehouse:0.0`.

Figura 5.2 Opciones Estandar para Programas X

Nombre	Seguido de	Ejemplo
<code>-geometry</code>	geometría de la ventana	<code>xterm -geometry 80x24+0+90</code>
<code>-display</code>	adónde quiere que el programa envíe la salida	<code>xterm -display lionsden:0.0</code>
<code>-fg</code>	color primario en primer plano	<code>xterm -fg yellow</code>
<code>-bg</code>	color primario en el segundo plano	<code>xterm -bg blue</code>

Por defecto, la presentación se toma de una variable de entorno (ver Sección 9.1.4) llamada `DISPLAY`¹², pero puede ser fijada mediante una opción desde el intérprete de comandos (ver Figura 5.2). Para ver cuál es el valor de la variable `DISPLAY`, basta ejecutar `“echo $DISPLAY”`.

5.6 Características comunes

A pesar de que X es una interfaz gráfica de usuario, se trata de una interfaz muy desequilibrada. Resulta imposible predecir como va a comportarse cualquier elemento del sistema porque se pueden reconfigurar, cambiar o incluso sustituir muy fácilmente. Eso significa que es muy difícil explicar como se han de usar los distintos elementos de la interfaz. Una de las razones que ya conocemos es la posibilidad de utilizar diferentes gestores de ventanas que a su vez son altamente configurables.

Otra explicación para el desequilibrio de la interfaz es que las aplicaciones X se pueden construir mediante “conjuntos de chismes¹³”. La distribución X estándar incluye los “widgets Athena” desarrollados en el MIT. Las aplicaciones gratuitas suelen hacer uso de este conjunto. Tienen la desventaja de que su aspecto no es particularmente atrayente y resultan algo más difíciles de usar en un programa que los otros.

Otro conjunto de widget popular es “Motif”. Motif es un conjunto no gratuito similar a la interfaz de usuario usada por Windows de Microsoft. Muchas aplicaciones comerciales así como algunas gratuitas usan el conjunto de widget Motif. El popular Browser de World Wide Web `netscape`, por ejemplo, usa Motif.

Veamos cuales son algunas de las cosas más habituales que puede encontrar.


5.6.1 Botones

Los botones son generalmente los componentes de la interfaz más fáciles de usar. Un botón se invoca situando el cursor del ratón sobre el botón en cuestión y pinchando (pulsando y soltando inmediatamente) el botón izquierdo. Los botones Athena y Motif son funcionalmente equivalentes pero tienen diferencias de apariencia.

¹²N.T. `DISPLAY` significa presentación en inglés.

¹³N.T. Se trata de la traducción de la palabra inglesa *widget*.

Figura 5.3 `emacs` cambia la barra de menú en función del tipo de fichero que esté editando. Aquí se puede ver una posible barra de menú.



5.6.2 Barras de menú

Una barra de menú es una colección de comandos accesibles mediante el ratón. Por ejemplo, la Figura 5.3 muestra la barra de menú de `emacs`. Cada palabra es la cabecera de una categoría. `File`¹⁴ contiene comandos que permiten cargar o grabar ficheros. Por convención, esta categoría incluye también el comando que termina la aplicación.

Para utilizar un comando, mueva el cursor del ratón sobre la categoría correspondiente (como por ejemplo `File`) y apriete el botón izquierdo. Sin soltarlo, arrastre el ratón hacia abajo para ver los diferentes comandos de esa categoría. Para seleccionar uno, mueva el cursor sobre el mismo y suelte el botón del ratón. Algunas barras de menú le permiten pinchar sobre la categoría—si este es su caso, pinche sobre la categoría para ver el menú que se mantendrá hasta que pinche de nuevo para seleccionar un comando, otro menú, o pinche fuera (indicando que no está interesado en ningún comando en particular).

5.6.3 Barras de desplazamiento

Una **barra de desplazamiento** es un método que permite ver sólo una parte de un documento, mientras que mantiene el resto fuera de la pantalla. Por ejemplo, en la Figura 5.4, la ventana de `xterm` sólo presenta el tercio inferior del texto disponible. Es muy fácil ver cual parte del texto es el que se puede ver en cualquier momento: la franja oscura de la barra está situada y tiene una longitud relativa a la porción de texto que se puede ver. Cuando no hay más que lo que se puede ver, toda la barra aparece de color oscuro. Si se trata de la mitad, será la mitad la que aparezca oscurecida.

Puede tener una barra de desplazamiento vertical a la derecha o la izquierda de su texto y una horizontal en la parte superior o inferior, dependiendo de la aplicación de que se trate.

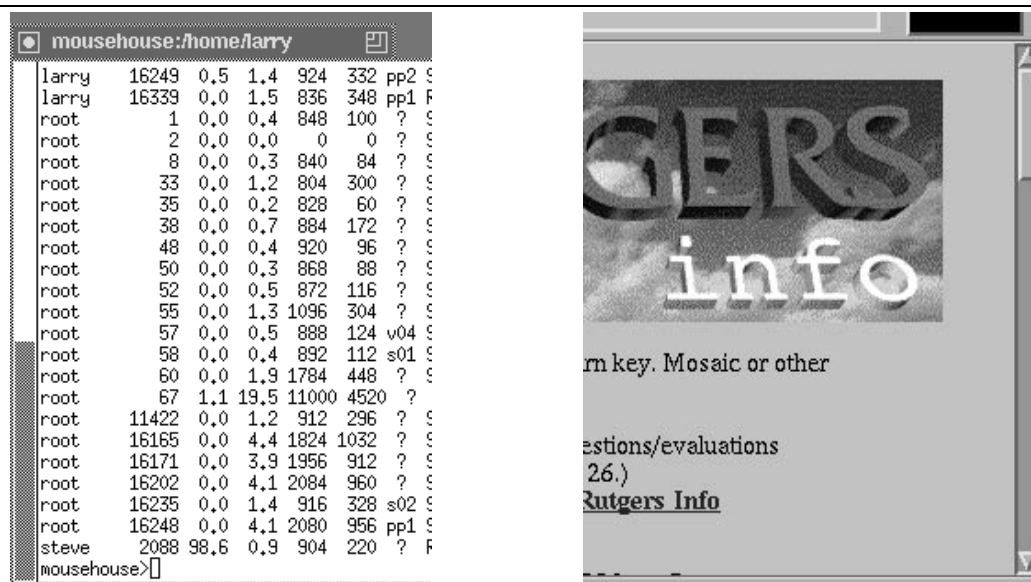
Barras de desplazamiento Athena

Las barras de desplazamiento Athena funcionan de forma diferente a las de otros sistemas de ventanas. Cada uno de los tres botones del ratón tiene una función diferente. Para desplazarse hacia arriba (es decir, para que ver el material situado por encima de lo que se puede ver en ese momento) puede pinchar en cualquier lugar de la barra con el botón derecho. Para desplazarse hacia abajo, puede usar el botón izquierdo de forma análoga.

También puede saltar a cualquier lugar en particular usando el botón central del ratón y pinchando en el lugar de la barra que corresponde a la posición del texto que le interesa. Esto hace a

¹⁴N.T. Del inglés *fichero* o *archivo*.

Figura 5.4 Barra de desplazamiento de tipo Athena en la parte izquierda de una ventana `xterm`. Junto a ella, una barra tipo Motif visible en una ventana con `netscape`.



la ventana presentar el material situado en ese punto del documento.

Barras de desplazamiento Motif

Una barra Motif actúa de un modo más parecido al acostumbrado en Windows de Microsoft o en un Macintosh. La parte derecha de la Figura 5.4 muestra un ejemplo. Debe notar que además de la barra en sí misma, hay unas flechas tanto en la parte superior como en la inferior. Estas flechas se pueden usar para ajuste fino de la pantalla: pinchando tanto el botón izquierdo como el central mueve la ventana ligeramente (por ejemplo una línea); el botón derecho no hace nada.

El comportamiento asociado a pinchar en la barra de desplazamiento es completamente diferente en las barras Motif del que exhiben las barras Athena. El botón derecho no tiene efecto. Asociado al botón izquierdo, la ventana se moverá hacia arriba si pincha sobre la posición actual o hacia abajo en el caso contrario. Si pincha *sobre* la barra en el lugar actual y la arrastra podrá moverse a cualquier posición a voluntad. Al soltar el botón se fija cuál es la posición final de la ventana.

Pinchando el botón central en cualquier lugar de la barra hace que la ventana se desplace hacia ese lugar de forma inmediata igual que ocurre en barras Athena. Sin embargo el punto elegido se convierte en el *centro* de la nueva pantalla de datos.

Capítulo 6

Trabajando con Unix

A UNIX saleslady, Lenore,
Enjoys work, but she likes the beach more.
 She found a good way
 To combine work and play:
She sells C shells by the seashore.

Unix es un potente sistema para aquellos que saben cómo dominar su poder. En este capítulo, intentaré describir varias maneras de usar el shell de Unix, **bash**, más eficientemente.

6.1 Comodines

En el capítulo anterior, se enseñaron los comandos para mantener ficheros **cp**, **mv**, y **rm**. A veces, se querrá tratar con más de un fichero a la vez—en realidad, con muchos a la vez. Por ejemplo, se quiere copiar todos los ficheros que empiecen por **data** en un directorio llamado **/backup**. Se podría hacer esto ejecutando muchos comandos **cp**, o escribiendo cada fichero en una línea de comando. Estos dos métodos llevan mucho tiempo, incluso, se tienen muchas posibilidades de cometer errores.

Una buena manera de hacer este trabajo es teclear:

```
/home/larry/report$ ls -F
1993-1      1994-1      data1      data5
1993-2      data-new    data2
/home/larry/report$ mkdir ~/backup
/home/larry/report$ cp data* ~/backup
/home/larry/report$ ls -F ~/backup
data-new    data1      data2      data5
/home/larry/report$
```

Como se puede observar, el asterisco indica a **cp** que tome todos los ficheros que empiecen por **data** y los copie a **/backup**. ¿Qué cree que “**cp d*w /backup**” puede haber hecho?

6.1.1 ¿Qué ocurre *realmente*?

Buena pregunta. De hecho, hay un par de caracteres especiales interceptados por el shell, **bash**. El carácter “*”, un asterisco, dice “cambia esta palabra con todos los ficheros que se ajusten a esta especificación”. Así, el comando “**cp data* /backup**”, como el de arriba, cambia a “**cp data-new data1 data2 data5 /backup**” antes de ejecutarse.

Para ilustrar esto, introduciré un comando nuevo, **echo**. **echo** es un comando extremadamente simple; repite, o muestra, cualquier parámetro. De este modo:

```
/home/larry$ echo Hola!
Hola!
/home/larry$ echo Como se encuentra?
Como se encuentra?
/home/larry$ cd report
/home/larry/report$ ls -F
1993-1          1994-1          data1          data5
1993-2          data-new          data2
/home/larry/report$ echo 199*
1993-1 1993-2 1994-1
/home/larry/report$ echo *4*
1994-1
/home/larry/report$ echo *2*
1993-2 data2
/home/larry/report$
```

Como se puede ver, el shell expande el comodín y pasa todos los ficheros al programa que se va a ejecutar. Esto plantea una pregunta interesante: ¿qué ocurre si *no* hay ficheros que se ajusten a la especificación del comodín? Pruebe “**echo /rc/fr*og**” y **bash** pasará literalmente la especificación del comodín al programa.

Otros shells, como **tcsh**, en vez de pasar el comodín literalmente, contestarán **No match**. Aquí está el mismo comando ejecutado bajo **tcsh**:

```
mousehouse>echo /rc/fr*og
echo: No match.
mousehouse>
```

La última pregunta que podría hacerse es, ¿qué pasa si quisiera mostrar **data***, en vez de la lista de nombres? Bien, tanto en **bash** como en **tcsh**, sólo se debe incluir la cadena entre comillas:

/home/larry/report\$ echo "data*"		mousehouse>echo "data*"
data*	Ó bien	data*
/home/larry/report\$		mousehouse>

6.1.2 El signo de interrogación

Además del asterisco, el shell también interpreta un signo de interrogación como un carácter especial. Un signo de interrogación coincidirá con un carácter, y sólo uno. Por ejemplo, “**ls /etc/??**” mostrará todos los ficheros de dos letras en el directorio **/etc**.

6.2 Ganar tiempo con bash

6.2.1 Editando la línea de comandos

A veces, escribes un comando largo a **bash** y, antes de pulsar **Intro**, se da cuenta de que ha cometido un error al escribirlo. Se puede simplemente borrar todo y volver a teclear correctamente, pero ¡es demasiado esfuerzo! En cambio, se pueden usar las flechas para moverse, borrar el/los carácter/es incorrecto/s, y escribir la información correctamente.

Hay muchas teclas especiales que ayudan a editar la línea de comandos, muchas de ellas similares a los comandos usados en GNU Emacs. Por ejemplo, **C-t** intercambia dos caracteres adyacentes¹. Se pueden encontrar muchos de los comandos en el capítulo sobre Emacs, Capítulo 8.

6.2.2 Completamiento de comandos y nombres de fichero

Otra peculiaridad de **bash** es la ejecución automática de las líneas de comando. Por ejemplo, veamos el siguiente ejemplo de un comando **cp** típico:

```
/home/larry$ ls -F
esto-es-un-fichero-largo
/home/larry$ cp esto-es-un-fichero-largo corto
/home/larry$ ls -F
corto                esto-es-un-fichero-largo
/home/larry$
```

Es una gran molestia tener que teclear cada letra de **esto-es-un-fichero-largo** cada vez que se quiere acceder a él, sucede lo mismo si queremos crear **esto-es-un-fichero-largo** copiando en él **/etc/passwd**². Ahora, aprenderemos a escribir el anterior comando **cp** más rápidamente y con menos posibilidad de error.

En vez de teclear el nombre del fichero entero, se escribe “**cp es**”, se pulsa y suelta la tecla **Tab**. Por arte de magia, el resto del nombre del fichero aparece en la línea de comandos, y se puede escribir **corto**. Desgraciadamente, **bash** no puede leer los pensamientos, por lo que se debe teclear **corto**.

Cuando se pulsa **Tab**, **bash** mira lo que hay escrito y busca un fichero que empiece como eso. Por ejemplo, si tecleo **/usr/bin/ema** y luego pulso **Tab**, **bash** encontrará **/usr/bin/emacs** ya que es el único fichero que empieza por **/usr/bin/ema** en mi sistema. En cambio, si tecleo **/usr/bin/ld** y pulso **Tab**, **bash** me avisará. Eso es porque tres ficheros, **/usr/bin/ld**, **/usr/bin/ldd**, y **/usr/bin/ld86** empiezan por **/usr/bin/ld** en mi sistema.

Si se intenta un completamiento y **bash** avisa, se puede pulsar inmediatamente **Tab** otra vez para conseguir una lista de todos los ficheros que coincidan con el patrón. De este modo, si no se está seguro del nombre exacto del fichero, podemos teclear los primeros caracteres del nombre y buscarlo en una lista más pequeña de ficheros.

¹**C-t** significa mantener pulsada la tecla marcada como “Ctrl”, y apretar la tecla “t”. Luego soltar ambas.

²“cp /etc/passwd esto-es-un-fichero-largo”

6.3 La entrada estándar y La salida estándar

Intentemos abordar un problema simple: conseguir un listado del directorio `/usr/bin`. Si hacemos `ls /usr/bin`, algunos de los nombres de los ficheros saldrán por arriba de la pantalla. ¿Como se pueden ver todos los ficheros?

6.3.1 Algunos conceptos de Unix

El sistema operativo Unix facilita mucho a los programas el uso del terminal. Cuando un programa escribe algo en la pantalla, está usando algo llamado **salida estándar**. Salida estándar, en inglés standard output o `stdout`, es la manera que tiene el programa de escribirle cosas al usuario. El nombre por el que se indica un programa es **entrada estándar** (`stdin`). Es posible que un programa se comunique con el usuario sin usar la entrada o salida estándar, pero la mayoría de los comandos que se tratan en este libro usan `stdin` y `stdout`.

Por ejemplo, el comando `ls` imprime una lista de los directorios en la salida estándar, que está normalmente “conectada” al terminal. Un comando interactivo, como el shell, `bash`, lee los comandos de la entrada estándar.

Un programa también puede escribir en el **error estándar**, ya que es muy fácil hacer que la salida estándar apunte a cualquier lugar aparte del terminal. El error estándar (`stderr`) está casi siempre conectado al terminal para que alguna persona pueda leer el mensaje.

En esta sección, examinaremos tres modos de enredarse con la entrada y salida estándar: redireccionar la salida, redireccionar la entrada, y las tuberías.

6.3.2 Redireccionar la salida

Un aspecto muy importante de Unix es la posibilidad de **redireccionar** la salida. Esto permite que, en vez de ver los resultados de un comando, los salvemos en un fichero o los enviemos directamente a una impresora. Por ejemplo, para redireccionar la salida del comando `ls /usr/bin`, se coloca un signo `>` al final de la línea, y se indica el fichero donde dejar la salida:

```
/home/larry$ ls
/home/larry$ ls -F /usr/bin > listado
/home/larry$ ls
listado
/home/larry$
```

Como se puede ver, en vez de escribir los nombres de todos los ficheros, el comando crea un fichero totalmente nuevo en el directorio actual. Echemos un vistazo a este fichero usando el comando `cat`. Si se recuerda, `cat` era un comando bastante inútil que copiaba lo que se escribía (entrada estándar) al terminal (salida estándar). `cat` también imprime un fichero en la salida estándar si se indica el fichero como parámetro:

```
/home/larry$ cat listado
```

```
..  
/home/larry$
```

La salida exacta del comando “`ls /usr/bin`” aparece en el contenido de **listado**. Por ahora todo bien, sin embargo no resuelve el problema original.³

A pesar de todo, `cat` hace algunas cosas interesantes cuando se redirecciona su salida. ¿Qué hace el comando “`cat listado > fichero`”? Normalmente, el “`> fichero`” dice “coge toda la salida del comando y ponla en **fichero**”. La salida del comando “`cat listado`” es el fichero **listado**. Así hemos inventado un nuevo (y no tan eficiente) método de copiar ficheros.

¿Qué ocurre con el comando “`cat > zorro`”? `cat` lee cada línea escrita en el terminal (entrada estándar) y la imprime de vuelta (salida estándar) hasta que lee `Ctrl-d`. En este caso, la salida estándar se ha redireccionado al fichero **zorro**. Ahora `cat` sirve como un editor rudimentario:

```
/home/larry$ cat > zorro  
El rapido zorro marron salta sobre el descuidado perro.  
pulsa Ctrl-d
```

Ahora se ha creado el fichero **zorro** que contiene la frase “El rapido zorro marron salta sobre el descuidado perro”. Un último uso del versátil comando `cat` es concatenar ficheros. `cat` imprimirá cada fichero dado como parámetro, uno despues de otro. El comando “`cat listado zorro`” imprimirá el listado del directorio `/usr/bin`, y luego la tonta frase. Así, el comando “`cat listado zorro > listyzorro`” creará un nuevo fichero conteniendo los contenidos de **listado** y **zorro**.

6.3.3 Redireccionar la entrada

Así como cuando se redirecciona la salida estándar, es posible redireccionar la entrada. En lugar de que un programa lea su entrada desde el teclado, la leerá desde un fichero. Como redireccionar la entrada está relacionado con redireccionar la salida, parece natural que “`<`” sea el carácter para redireccionar la entrada. Éste también se usa después del comando que se desee ejecutar.

Esto es generalmente útil si se tiene un fichero de datos y un comando que espera sus datos desde la entrada estándar. Muchos comandos permiten especificar un fichero sobre el cual operar, así que en las actividades diarias el “`<`” no se usa tanto como otras técnicas.

6.3.4 Las tuberías

Muchos comandos Unix producen gran cantidad de información. Por ejemplo, es normal que un comando como “`ls /usr/bin`” produzca más salida que la que se puede ver en pantalla. Para que se pueda ver toda la información de un comando como “`ls /usr/bin`”, es necesario usar otro comando Unix llamado **more**⁴. **more** parará cada vez que la pantalla se llene de información. Por

³Para lectores impacientes, el comando que se debe usar es **more**. Sin embargo, hay que hablar un poco sobre algo más antes de llegar ahí.

⁴Se llama **more** porque ese es el indicador que originalmente mostraba: “`--more--`”. En varias versiones de Linux el comando **more** es idéntico a un comando más avanzado que hace todo lo que **more** puede hacer y más aún. Como demostración de que los programadores de ordenadores son malos cómicos, llamaron a este nuevo programa “**less**”.

ejemplo, “`more < /etc/rc`” mostrará el fichero `/etc/rc` como lo haría “`cat /etc/rc`”, excepto que `more` permite leerlo. `more` también admite el comando “`more /etc/rc`”, y esa es la forma normal de invocarlo.

Sin embargo, eso no ayuda al problema de que “`ls /usr/bin`” muestre más información de la que se pueda ver. “`more < ls /usr/bin`” no funciona— ¡la redirección de entrada sólo funciona con ficheros, no comandos! Se *podría* hacer esto:

```
/home/larry$ ls /usr/bin > temp-ls
/home/larry$ more temp-ls
..
/home/larry$ rm temp-ls
```

Pero, Unix propone una forma más limpia de hacerlo. Se puede usar el comando “`ls /usr/bin | more`”. El carácter “|” es una **tubería**. Como una tubería de agua, una tubería Unix controla el flujo. En vez de agua, se controla el flujo de información.

Los **filtros** son programas muy útiles para usarse en conjunción con las tuberías. Un filtro es un programa que lee la entrada estándar, la cambia de alguna manera, y la saca por la salida estándar. `more` es un filtro—lee los datos que coge de la entrada estándar y los muestra por la salida estándar pantalla a pantalla, permitiendo leer el fichero. `more` no es un gran filtro porque su salida no se puede enviar a otro programa.

Otros filtros incluyen los programas `cat`, `sort`, `head`, y `tail`. Por ejemplo, si se quiere leer sólo las primeras diez líneas de la salida de `ls`, se puede usar “`ls /usr/bin | head`”.

6.4 Multitarea

6.4.1 Usando el control de trabajos

Control de trabajos se refiere a la habilidad de poner procesos (esencialmente, otra palabra para programas) en **background** (segundo plano) y ponerlos de vuelta en **foreground** (primer plano). Esto es como decir, que se quiere ser capaz de ejecutar algo mientras se hacen otras cosas, pero que estén ahí otra vez cuando se les quiera decir algo o pararlos. En Unix, la principal herramienta para el control de procesos es el shell—seguirá la pista de los procesos por usted, si se aprende como hablar su lenguaje.

Las dos palabras más importantes en ese lenguaje son **fg**, para primer plano, y **bg**, para segundo plano. Para entender como funcionan, use el comando `yes` en el indicador del sistema.

```
/home/larry$ yes
```

Esto produce el maravilloso efecto de desplazar una larga columna de `yes` por la parte izquierda de la pantalla, tan rápido que no se pueden seguir⁵. Para pararlo, se podría pulsar `Ctrl-c` y matarlo,

⁵ Hay buenas razones para que este extraño comando exista. Ciertos comandos esperan una confirmación —un “sí” ([y]es en inglés) a una pregunta. El comando `yes` permite al programador automatizar la respuesta a esas preguntas.

pero esta vez Ud. oprimirá **Ctrl-z**. Parece haberse detenido, pero habrá un mensaje antes del indicador de sistema, más o menos parecido a este:

```
[1]+  Stopped                  yes
```

Significa que el trabajo **yes** se ha **suspendido** en el segundo plano. Se puede hacer que siga ejecutándose tecleando **fg** en el indicador de sistema, que lo pondrá en primer plano otra vez. Si se desea, se pueden hacer otras cosas antes, mientras está suspendido. Pruebe unos cuantos **ls** o algo antes de ponerlo en primer plano nuevamente.

Una vez que ha vuelto al primer plano, las **yes** empezarán a salir otra vez, tan rápido como antes. No hay que preocuparse de que si mientras ha estado suspendido ha “almacenado” más **yes** para enviarlas a la pantalla: cuando un trabajo se suspende, el programa entero no se ejecuta hasta que se lo vuelva de vuelta a la vida. (Ahora pulse **Ctrl-c** para matarlo de veras).

Pongamos aparte el mensaje que obtuvimos del shell:

```
[1]+  Stopped                  yes
```

El número entre corchetes es el **número de trabajo** de este proceso, y se usará cuando se necesite referenciarlo específicamente. (Naturalmente, desde el momento que tengamos en ejecución múltiples trabajos, se necesita un modo de acceder a cada uno). El “+” siguiente indica que ése es el “trabajo actual” — esto es, el proceso más reciente que se ha movido del primer plano al segundo. Si se tecleara “**fg**”, se pondría el trabajo con el “+” en primer plano otra vez. (Más sobre esto después, cuando se discuta la ejecución de múltiples trabajos a la vez). La palabra **Stopped** significa que el trabajo está “parado”. El trabajo no está muerto, pero actualmente no se ejecuta. Linux lo ha guardado en un estado especial de suspendido, listo para saltar a la acción cuando alguien lo solicite. Finalmente, el **yes** es el nombre del trabajo que se ha detenido.

Antes de seguir adelante, matemos este trabajo y lo arrancamos otra vez de forma diferente. El comando se llama **kill** y se usa del siguiente modo:

```
/home/larry$ kill %1
[1]+  Stopped                  yes
/home/larry$
```

Ese mensaje sobre el proceso que indica “parado” otra vez induce a error. Para saber si aún está vivo (eso es, tanto en ejecución como congelado en un estado suspendido), teclee “**jobs**”:

```
/home/larry$ jobs
[1]+  Terminated             yes
/home/larry$
```

Ahora ya lo sabe: ¡el trabajo ha terminado! (Es posible que el comando **jobs** no muestre nada, lo que simplemente significa que no hay trabajos ejecutándose en segundo plano. Si se acaba de matar un trabajo, y al teclear **jobs** no muestra nada, se tiene la seguridad de que el comando **kill** se ha ejecutado correctamente. Normalmente indicará que el trabajo ha “terminado”).

Ahora, ejecute **yes** de nuevo, de esta forma:

```
/home/larry$ yes > /dev/null
```

Si lee la sección sobre la redirección de entrada y salida, sabrá que se está enviando la salida de `yes` a un fichero especial llamado `/dev/null`. `/dev/null` es un agujero negro que come cualquier salida que se le envíe (se puede imaginar ese torrente de `yes` saliendo por detrás del ordenador y perforando un agujero en la pared, si eso le hace feliz).

Después de teclear esto, no se recuperará el indicador de sistema, pero tampoco saldrá esa columna de `yes`. Sin embargo la salida se está enviando a `/dev/null`, el trabajo aún se ejecuta en primer plano. Como siempre, se puede suspender pulsando `Ctrl-z`. Hágalo ahora para volver al indicador del sistema.

```
/home/larry$ yes > /dev/null
["yes" se ejecuta, y solamente se ha pulsado Ctrl-z]
[1]+  Stopped                  yes >/dev/null

/home/larry$
```

Hmm... ¿hay alguna forma de ponerlo en ejecución en segundo plano, mientras deja el indicador del sistema para trabajar de forma interactiva? El comando para hacer eso es `bg`:

```
/home/larry$ bg
[1]+ yes >/dev/null  &
/home/larry$
```

Ahora, deberá tener confianza en mí sobre esto: después de teclear `bg`, el trabajo “`yes > /dev/null`” habrá continuado con su ejecución otra vez, pero esta vez en segundo plano. De hecho, si hace alguna cosa en el prompt, como `ls` u otros, se dará cuenta que su máquina se ha ralentizado un poco (¡generar y descargar continuamente una cadena preparada de “yes” lleva algo de tiempo, al fin y al cabo!) Aparte de esto, no hay ningún otro efecto. Se puede hacer lo que se desee en el indicador del sistema, y `yes` continuará felizmente enviando su salida al agujero negro.

Ahora hay dos formas diferentes de matarlo: con el comando `kill` que ya se explicó, o poniendo el trabajo en primer plano de nuevo e interrumpirlo con una interrupción, `Ctrl-c`. Probemos la segunda forma, sólo para entender la relación entre `fg` y `bg` un poco mejor;

```
/home/larry$ fg
yes >/dev/null

[ahora esta en primer plano.  Imagine que pulso Ctrl-c para terminarlo]

/home/larry$
```

Bueno, se acabó. Ahora, ejecute unos cuantos trabajos simultáneamente, como estos:

```
/home/larry$ yes > /dev/null &
[1] 1024
/home/larry$ yes | sort > /dev/null &
[2] 1026
```

```
/home/larry$ yes | uniq > /dev/null
[ly aqui, pulse Ctrl-z para suspenderlo, por favor]

[3]+  Stopped                  yes | uniq >/dev/null
/home/larry$
```

La primera cosa que le debe llamar la atención de estos comandos es la terminación “&” al final de los dos primeros. Poner un “&” después del comando indica al shell que los ejecute en segundo plano desde el principio. (Es una forma de evitarse tener que ejecutar el programa, pulsar Ctrl-z, y luego teclear “bg”). Así, estos dos comandos han empezado a ejecutarse en segundo plano. El tercero está suspendido e inactivo en este momento. Se puede dar cuenta de que la máquina se ha vuelto más lenta, ya que los que se están ejecutando requieren un poco de tiempo de CPU.

Cada uno indica su número de trabajo. Los dos primeros también muestran sus **números de identificación de proceso**, o PID, después del número de trabajo. Los PIDs normalmente no son algo que se necesite conocer, pero a veces viene bien.

Matemos el segundo, ya que creo que está ralentizando su máquina. Se puede teclear “kill %2”, pero eso sería demasiado fácil. Por el contrario, haga esto:

```
/home/larry$ fg %2
yes | sort >/dev/null
[pulse Ctrl-c para matarlo]

/home/larry$
```

Como esto demuestra, fg toma parámetros empezando con “%”. De hecho, se podría teclear sólo esto:

```
/home/larry$ %2
yes | sort >/dev/null
[pulse Ctrl-c para matarlo]

/home/larry$
```

Esto funciona por que el shell automáticamente interpreta un número de trabajo como una petición para poner ese trabajo en primer plano. Se puede indicar los números de trabajo con otros números precedidos por un “%”. Ahora teclee “jobs” para ver cuáles trabajos quedan en ejecución:

```
/home/larry$ jobs
[1]-  Running                  yes >/dev/null  &
[3]+  Stopped                  yes | uniq >/dev/null
/home/larry$
```

El “-” indica que ese trabajo número 1 es segundo en la lista para ser puesto en el primer plano, si sólo se teclea “fg” sin dar parámetros. El “+” indica que el trabajo especificado es el primero en la lista—un “fg” sin parámetros pondrá al trabajo número 3 en el primer plano. Sin embargo, se puede acceder a él llamándolo, si se desea, mediante:

```

/home/larry$ fg %1
yes >/dev/null
[ahora pulse Ctrl-z para suspenderlo]

[1]+  Stopped                  yes >/dev/null
/home/larry$

```

Al cambiar al trabajo número 1 y luego suspenderlo han cambiado las prioridades de todos los trabajos de usuario. Esto se puede ver con el comando `jobs`:

```

/home/larry$ jobs
[1]+  Stopped                  yes >/dev/null
[3]-  Stopped                  yes | uniq >/dev/null
/home/larry$

```

Ahora los dos están parados (porque los dos se han suspendido con `Ctrl-z`), y el número 1 es el siguiente en la lista a entrar en el primer plano por defecto. Esto es así porque se le puso en el primer plano manualmente, y luego fue suspendido. El “+” siempre se refiere al trabajo más reciente que ha sido suspendido del primer plano. Se puede continuar con su ejecución otra vez:

```

/home/larry$ bg
[1]+ yes >/dev/null  &
/home/larry$ jobs
[1]-  Running                  yes >/dev/null
[3]+  Stopped                  yes | uniq >/dev/null
/home/larry$

```

Fíjese que ahora está en ejecución, y el otro trabajo se ha movido en la lista y tiene el “+”. Ahora matémoslos para que el sistema no esté permanentemente ralentizado por procesos que no hacen nada.

```

/home/larry$ kill %1 %3
[3]   Terminated             yes | uniq >/dev/null
/home/larry$ jobs
[1]+  Terminated             yes >/dev/null
/home/larry$

```

Aparecerán varios mensajes sobre la terminación de los trabajos—nada muere tranquilamente, al parecer. La figura 6.1 de la página 59 muestra un breve resumen de lo que se debe saber acerca del control de trabajos.

6.4.2 Teoría del control de trabajos

Es importante entender que el control de procesos lo hace el shell. No hay ningún programa en el sistema llamado `fg`; por eso, `fg`, `bg`, `&`, `jobs`, y `kill` son internos al shell⁶. (A veces `kill` es un programa independiente; en el shell `bash` usado por Linux pertenece al shell). Esto es una forma

⁶N. del T.: En el original pone “shell-builtins”, eso es que se compilan dentro del shell.

lógica de hacerlo: ya que cada usuario quiere su propio espacio de control de trabajos, y cada usuario ya tiene su propio shell, es más fácil que el shell siga la pista de los trabajos usuario. Por otro lado, cada número de trabajo de usuario sólo tiene significado para ese usuario: mi trabajo número [1] y su trabajo número [1] son probablemente dos procesos totalmente diferentes. De hecho, si se está conectado más de una vez, cada uno de los shells tendrá datos únicos sobre el control de trabajos, así como también un usuario puede tener dos trabajos diferentes con el mismo número ejecutándose en dos shells diferentes.

La manera segura de referenciarlos es usar el número **ID**entificador de Proceso (PID). Estos números abarcan todo el sistema — cada proceso tiene su propio (y único) número. Dos usuarios diferentes pueden referenciar un proceso por su PID y saber que están hablando sobre el mismo proceso (¡asumiendo que están conectados en la misma máquina!)

Echémosle un vistazo a un comando más para entender que son los PIDs. El comando `ps` lista todos los procesos en ejecución, incluyendo el shell. Pruébalo. Tiene también unas cuantas opciones, de las cuales las más importantes (para mucha gente) son “a”, “u”, y “x”. La opción “a” lista los procesos pertenecientes a algún usuario, no sólo los suyos propios. La “x” lista los procesos que no tienen un terminal asociado a ellos⁷. Finalmente, la “u” da información adicional sobre los procesos que es frecuentemente útil.

Para hacerse una idea de lo que realmente está haciendo el sistema, se ponen todos juntos: “`ps -aux`”. Se pueden ver los procesos que usan más memoria mirando la columna %MEM, y más CPU mirando a la columna %CPU. (La columna TIME lista la cantidad *total* de tiempo de CPU usado.)

Otra nota rápida sobre los PIDs. `kill`, aparte de tomar opciones de la forma `%notrabajo`, toma opciones de los PIDs en crudo. Esto es, si pone un “`yes > /dev/null`” en segundo plano, se ejecuta “`ps`”, y se busca el `yes`. Luego se teclea “`kill PID`”⁸.

Si empieza a programar en C con su sistema Linux, pronto aprenderá que el control de trabajos del shell es sólo una versión interactiva de las llamadas a las funciones `fork` y `exec1`. Esto es demasiado complejo para explicarlo aquí, pero sería útil recordarlo después cuando se esté programando y se quieran ejecutar múltiples procesos desde un simple programa.

6.5 Consolas virtuales: como estar en varios lugares a la vez

Linux soporta **consolas virtuales**. Son una manera de hacer que su simple máquina aparezca como múltiples terminales, todos conectados al mismo núcleo Linux. Por fortuna, usar las consolas virtuales es una de las cosas más simples en Linux: hay “hot keys”⁹ para cambiar entre las consolas rápidamente. Para probarlo, hay que conectarse al sistema, pulsar la tecla `Alt` izquierda, y pulsar

⁷Esto sólo tiene sentido para ciertos programas que no tienen que hablar con el usuario a través del teclado.

⁸En general, es más fácil matar el número del trabajo en vez de usar PIDs.

⁹N.T.: No he creído apropiado traducir este término, ya que muchos usuarios saben a lo que se refiere. De todas formas para el que no lo sepa, esto es, una simple combinación de teclas que hace un cierto trabajo.

F2 (esto es, la tecla de función número 2)^{10 11}.

Se encontrará con otro indicador para conectarse. No se alarme: ahora está en la consola virtual (VC) número 2. Conéctese y haga algunas cosas — unos cuantos `ls` o lo que sea — para confirmar que es un shell real. Ahora puede volver a la VC número 1, pulsando el **Alt** izquierdo y **F1**. O se puede mover a una *tercera* VC, de la forma obvia (**Alt-F3**).

Generalmente los sistemas Linux vienen con cuatro VC activadas por defecto. Esto se puede incrementar a ocho; estos temas se cubrirán en *The Linux System Administrator's Guide*. Ello implica editar uno o dos ficheros en el directorio `/etc`. Sin embargo, cuatro deben ser suficientes para la mayoría de las personas.

Una vez las haya usado, las VC probablemente se convertirán en una herramienta indispensable para tener varias cosas ejecutándose a la vez. Por ejemplo, yo normalmente ejecuto Emacs en la VC 1 (y hago la mayor parte de mi trabajo ahí), mientras tengo un programa de comunicaciones en la VC 3 (así puede coger o dejar ficheros via modem mientras trabajo, o ejecutar programas en máquinas remotas), y mantengo un shell en la VC 2 sólo en caso de que tenga que ejecutar algo sin involucrar a la VC 1.

¹⁰ Hay que asegurarse de que esto se hace desde consolas en modo texto: si se está ejecutando X Window u otra aplicación gráfica, probablemente no funcionará, sin embargo corre el rumor de que pronto se podrá cambiar entre las consolas virtuales en X Window de Linux.

¹¹ N. del T.: De hecho al momento de la traducción, la combinación es **Ctrl-Alt-tecla-de-función**.

Figura 6.1 Resumen de comandos y teclas usados para el control de trabajos.

<code>fg %n^otrabajo</code>	Este es un comando del shell que devuelve un trabajo al primer plano. Para saber cuál es éste por defecto, se teclea “ <code>jobs</code> ” y se busca el que tiene el <code>+</code> . Parámetros: número de trabajo opcional. El trabajo por defecto se identifica con el <code>+</code> .
<code>&</code>	Cuando se añade un <code>&</code> al final de la línea de comandos, indica al comando que se ejecute en segundo plano automáticamente. Este trabajo está entonces sujeto a todos los métodos usuales para el control de trabajos aquí detallados.
<code>bg %n^otrabajo</code>	Este es un comando del shell que manda a un trabajo suspendido ejecutarse en segundo plano. Para saber cual es éste por defecto, se teclea “ <code>jobs</code> ” y se busca el que tiene el <code>+</code> . Parámetros: número de trabajo opcional. El trabajo por defecto se identifica con el <code>+</code> .
<code>kill %n^otrabajo PID</code>	Este es un comando del shell que obliga a un trabajo en segundo plano, ya sea suspendido o en ejecución, a terminar. Se debe siempre especificar el número de trabajo o PID, y si se están usando números de trabajo, no hay que olvidar poner el <code>%</code> . Parámetros: El número de trabajo (a continuación del <code>%</code>) o el PID (no es necesario el <code>%</code>). Se puede especificar más de un proceso o trabajo en una línea.
<code>jobs</code>	Este comando del shell lista información sobre los trabajos que están en ese momento en ejecución o suspendidos. A veces también dice cuáles son los que acaban de salir o han terminado.
<div>Ctrl-c</div>	Este es el carácter genérico de interrupción. Normalmente, si se pulsa mientras un programa se está ejecutando en primer plano, matará al programa (puede que haya que hacerlo varias veces). Sin embargo, no todos los programas responderán a este método de terminación.
<div>Ctrl-z</div>	Esta combinación de teclas normalmente suspende un programa, puede que algunos programas lo ignoren. Una vez suspendido, el trabajo se puede reiniciar en el segundo plano o se puede matar.

Capítulo 7

Pequeños programas potentes

```
better !pout !cry
better watchout
lpr why
santa claus <north pole >town

cat /etc/passwd >list
ncheck list
ncheck list
cat list | grep naughty >nogiftlist
cat list | grep nice >giftlist
santa claus <north pole > town

who | grep sleeping
who | grep awake
who | egrep 'bad|good'
for (goodness sake) {
    be good
}
```

7.1 El poder de Unix

El poder de Unix¹ se esconde en pequeños comandos que no parecen ser muy útiles cuando se utilizan por separado, pero combinados con otros (directa o indirectamente) proporcionan un entorno mucho más potente y flexible que la mayoría de los otros sistemas operativos. Los comandos de los que hablaremos en este capítulo son entre otros: **sort**, **grep**, **more**, **cat**, **wc**, **spell**, **diff**, **head**, y **tail**.

Veremos lo que cada una de las utilidades hace por separado y luego daremos algunos ejemplos de como utilizarlos conjuntamente².

¹N. del T.: El cookie de apertura de este capítulo es la traducción a comandos Unix de una canción de navidad: “Santa Claus is coming to town” (escrita por H. Gillespie, J. F. Coots)

²Nótese que los resúmenes de los comandos en este capítulo no son extensos. Para conocer cada una de las opciones,

7.2 Trabajando con ficheros

Junto a comandos como `cd`, `mv`, y `rm` que ya se vieron en el Capítulo 4, hay otros comandos que trabajan con ficheros pero no con los datos contenidos en ellos. Estos son `touch`, `chmod`, `du`, y `df`. Ninguno de ellos se preocupa por lo que *contiene* el fichero—simplemente modifican algunas de las informaciones que Unix guarda acerca del fichero.

Estos comandos manipulan:

- Los registros de tiempo. Todo fichero tiene tres fechas asociadas a él.³ Las tres fechas son las siguientes: creación del fichero, última modificación (cuando se produjo el último cambio en el fichero), y último acceso (cuando se produjo la última lectura del fichero).
- El dueño. Cada fichero en Unix pertenece a un único propietario.
- El grupo. Cada fichero tiene también asociados un grupo de usuarios. El grupo más usual para los ficheros de usuario se denomina `users`, que es al cual, por lo general, pertenecen todos los usuarios con cuenta en el sistema.
- Los permisos. Cada fichero tiene ciertos permisos (también llamados “privilegios”) asociados a él, que indican a Unix quién puede acceder a ese fichero, cambiarlo o en el caso de ser un programa, ejecutarlo. Cada uno de estos permisos puede ser establecido por separado para el dueño, el grupo, y el resto de usuarios.

`touch` *fichero1 fichero2 ... ficheroN*

`touch` actualiza los registros de fecha y hora con la fecha y hora actual de los ficheros indicados en la línea de comandos. Si el fichero no existe, `touch` lo creará. También es posible especificar la fecha y hora a registrar en la información de los ficheros—consulte la página del manual (comando `man`) para más información sobre `touch`.

`chmod` [-Rfv] *modo fichero1 fichero2 ... ficheroN*

El comando que se utiliza para modificar los permisos de un fichero es el `chmod`⁴. Antes de meternos en como utilizar este comando, veamos primero que permisos hay en Unix. Cada fichero tiene asociados un grupo de permisos. Estos permisos le indican al sistema operativo quien puede leer, escribir o ejecutar como programa el fichero. (A continuación, se explica como puede un usuario hacer estas cosas. Un programa ejecutado por un usuario puede hacer las mismas cosas que las que

vea la página del manual correspondiente con el comando `man`.

³Los antiguos sistemas de ficheros de Linux sólo almacenaban una fecha, ya que derivaban de Minix. Si se utiliza alguno de estos sistemas de ficheros, parte de la información no estará disponible—y las operaciones que se realicen sobre ellas serán inútiles.

⁴N. del T.: Abreviatura de **change mode**, cambiar modo.

le estén permitidas al propio usuario; esto puede suponer un problema de seguridad, si se desconoce lo que hace cada programa en particular.)

Unix reconoce tres tipos diferentes de individuos: primero, el propietario del fichero (y la persona que puede utilizar el comando `chmod` sobre ese fichero). Segundo, el “grupo”. El grupo de la mayoría de los ficheros normales de un usuario del sistema será “users”. (Para ver el grupo de un fichero en particular, utilizar “`ls -l fichero`”). Por último, está el “resto” que no son ni propietarios ni pertenecen al grupo, denominados “otros”⁵

De esta forma, por ejemplo, un fichero puede tener permisos de lectura y escritura para el propietario, permiso de lectura para el grupo y ningún tipo de permisos para otros. O por alguna razón, un fichero puede tener permisos de lectura/escritura para el grupo y otros, pero ¡ningún privilegio para el propietario!

Veamos algunos ejemplos de como cambiar permisos utilizando `chmod`. Primero creamos un fichero utilizando `cat`, `emacs`, o cualquier otro programa. Por defecto, son posibles las operaciones de lectura y de escritura sobre el fichero. (Los permisos dados a otros usuarios dependen de como estén establecidas las variables del sistema y de la cuenta del usuario). Podemos asegurarnos de que tenemos permiso de lectura utilizando el comando `cat`, a continuación eliminamos el privilegio de lectura para nosotros mismos utilizando “`chmod u-r fichero`”⁶. Ahora si intentamos leer el fichero, obtendremos el mensaje de error `Permission denied`. Devolvemos el privilegio de lectura utilizando “`chmod u+r fichero`”.

Los permisos de directorio utilizan las tres mismas ideas anteriores: leer, escribir, y ejecutar, pero con ligeras variaciones. El privilegio de lectura permite al usuario (o al grupo, o a otros) leer el directorio—listar los nombres de los ficheros. El permiso de escritura permite añadir o borrar ficheros. El permiso de ejecución permite al usuario acceder a ficheros en el directorio o a cualquier subdirectorio. (Si un usuario no tiene permisos de ejecución para un directorio, entonces no puede siquiera hacer un `cd` al mismo).

Para utilizar `chmod`, reemplazar el *modo* por el **usuario**, **grupo**, **otro** o todos (**all**) y la operación a realizar. (Es decir, utilizar un signo más “+” para añadir privilegios y un menos “-” para quitarlos. Un signo igual “=” indica los mismos permisos para los usuarios involucrados en el modo). Los permisos posibles son lectura (**read**), escritura (**write**), y ejecución (**execute**).

La opción “-R” al utilizar `chmod` cambiará los permisos de un directorio, de todos los ficheros contenidos en él, y de forma recursiva los permisos del árbol de subdirectorios a partir de él. La opción “-f” fuerza a que `chmod` intente cambiar los permisos incluso si el usuario no es el dueño del fichero. (Si se utiliza el parámetro “f” al ejecutar `chmod` no se mostrará un mensaje de error en caso de que no pueda cambiar los permisos del fichero.), por último la opción “v” hace que la ejecución del `chmod` informe de todo lo que hace.

⁵N. del T.: En ingles: “owner” (con “u” de “user”), “group” and “others” respectivamente.

⁶N. del T.: El parámetro `u-r` indica “user minus read”, es decir: usuario menos lectura.

7.3 Estadísticas del sistema

Los comandos de esta sección son de utilidad a la hora de mostrar las estadísticas del sistema operativo, o de una parte de él.

du [-abs] [*trayectoria1 trayectoria2 ... trayectoriaN*]

El comando **du**⁷ contabilizará el espacio de disco ocupado por un subdirectorio *y todos sus subdirectorios*. El uso de este comando sin utilizar parámetros devolverá una lista de cuanto disco consume cada subdirectorio del directorio actual, y al final del informe, cuanto disco utiliza el directorio (y todos sus subdirectorios). Si se le pasa uno o más parámetros, devolverá la cantidad de espacio utilizado por esos ficheros o directorios en lugar de la del directorio actual.

La opción “**a**” mostrará además del espacio de los directorios, el de los ficheros. Utilizando “**b**” como opción presentará el total en bytes, en lugar de kilobytes (1 kilobyte = 1024 caracteres)⁸. Finalmente el parámetro “**s**” informará sólo acerca de los directorios explícitos en la línea de comando y *no* de sus subdirectorios.

df

El comando **df**⁹ resume la cantidad de espacio utilizada en el disco. Para cada sistema de ficheros (recuerde que sistemas de ficheros diferentes son o bien unidades físicas o particiones diferentes) muestra el espacio total de disco, la cantidad utilizada, la cantidad disponible y la capacidad total del sistema de ficheros que se utiliza.

Un caso extraño que puede darse es la posibilidad de tener una capacidad superior al 100%, o que la cantidad utilizada más la disponible no sea igual a la total. Esto es debido a que Unix reserva parte del espacio de cada sistema de ficheros para el directorio **raíz**. De esta forma aunque algún usuario accidentalmente sature el disco, el sistema todavía tendrá un poco de espacio para seguir operativo.

Este comando no ofrece opciones que puedan ser de utilidad al usuario.

uptime

El comando **uptime**¹⁰ informa sobre el tiempo en el que el sistema ha estado activo, es decir el tiempo transcurrido desde que Unix arrancó por última vez.

Este comando también devuelve la hora actual y el promedio de carga que soporta el sistema. El promedio de carga es el número medio de procesos esperando a ejecutar en un determinado periodo

⁷N. del T.: Abreviatura de **disk usage**, utilización del disco.

⁸Un byte es el equivalente de una letra en un documento de texto.

⁹N. del T.: Abreviatura de **disk filling**, espacio disponible en el disco.

¹⁰N. del T.: Tiempo en estado activo.

de tiempo. `uptime` muestra el promedio de carga del último minuto y de los cinco y diez últimos minutos. Si este promedio de carga se aproxima a cero indica que el sistema ha estado relativamente desocupado; por el contrario si el promedio es cercano al uno indica que el sistema ha estado casi completamente utilizado pero en ningún momento sobrecargado. Los promedios de carga altos son el resultado de la ejecución simultánea de varios programas.

Extraordinariamente, `uptime` es uno de los pocos comandos de Unix que *no* tienen opciones!

`who`

El comando `who` muestra los usuarios activos en ese momento en el sistema y cuando han iniciado su respectiva sesión. Si le damos los parámetros “`am i`” (es decir : “`who am i`”), nos devuelve la información relativa a nosotros mismos.¹¹

`w [-f] [nombre_usuario]`

El programa `w` devuelve los usuarios actuales del sistema y que están haciendo. (Básicamente combina la funcionalidad de `uptime` y `who`. La cabecera del informe que presenta `w` es exactamente la misma que `uptime`, siendo la información de cada una de las líneas la siguiente: el nombre del usuario, hora de inicio de la sesión (y cuanto tiempo ha estado ocioso). `JCPU` es la cantidad total de tiempo de CPU utilizada por ese usuario, mientras que `PCPU` es la cantidad total de tiempo utilizada por sus tareas actuales.

Si al comando se le pasa la opción “`f`”, mostrará los sistemas remotos desde los que los usuarios acceden, si los hay. Puede indicarse un nombre de usuario como parámetro para mostrar sólo información relativa a él.

7.4 ¿Qué hay en un fichero?

Principalmente hay dos comandos en Unix para listar ficheros, `cat` y `more`. Sobre ambos ya se habló en el Capítulo 6.

`cat [-nA] [fichero1 fichero2 ... ficheroN]`

`cat` no es un comando de uso amigable—no espera a que el usuario acabe de leer el fichero, y se utiliza generalmente en contextos de tuberías. Aunque pueden utilizarse algunas opciones muy útiles del comando, por ejemplo, “`n`” numera todas las líneas del fichero, y “`A`” mostrará los caracteres de control como caracteres normales en lugar de hacer (posiblemente) cosas extrañas en la pantalla. (Recuerde que para ver las opciones más curiosas y quizá “menos útiles” utilice el comando `man`:

¹¹N. del T.: literalmente es como preguntar al sistema ¿quién soy yo?

“**man cat**”). Si no se especifican ficheros en la línea de comandos, **cat** aceptará la entrada desde **stdin**¹².

more [-l] [+*número_línea*] [*fichero1 fichero2 ... ficheroN*]

more es más útil, y además es el comando recomendado para ver ficheros de texto ASCII. La única opción interesante es “**l**”, que indica al comando que no se desea interpretar el carácter **Ctrl-L** como carácter de “nueva página”. El comando comenzará en la línea *número_línea* especificada.

Al ser **more** un comando interactivo, hemos resumido a continuación las órdenes más comunes:

Barra espaciadora Pasar a la siguiente pantalla de texto.

d Pasar 11 líneas de pantalla, o aproximadamente la mitad de una pantalla normal: 25 líneas.

/ Busca una expresión regular. La construcción de una expresión regular puede ser muy complicada por lo que es recomendable teclear simplemente el texto a buscar. Por ejemplo, **/sapo** **Intro** buscará la primera ocurrencia de “sapo” en el fichero a partir de la posición actual. La misma tecla seguida por un **Intro** buscará la siguiente ocurrencia de la última expresión buscada.

n Buscará la próxima aparición de la expresión regular especificada.

:n Pasar al siguiente fichero, en caso de que se especifique más de un fichero en la línea de comandos.

:p Pasar al fichero anterior.

q Terminar **more**.

head [-líneas] [*fichero1 fichero2 ... ficheroN*]

head mostrará las primeras diez líneas de los ficheros especificados, o las primeras diez líneas de la **stdin** si no se especifica ningún fichero en la línea de comandos. Cualquier opción numérica se tomará como el número de líneas a mostrar, por ejemplo “**head -15 rana**” mostrará las primeras quince líneas del fichero **rana**.

tail [-líneas] [*fichero1 fichero2 ... ficheroN*]

Como **head**, **tail** mostrará solo una parte del fichero, en este caso el final del fichero, las últimas diez líneas del fichero, o que provengan de la **stdin**. **tail** también acepta la opción de especificar el número de líneas, como en el caso anterior.

¹²N. del T.: **standard input**, entrada estándar, generalmente entrada de datos por teclado.

`file [fichero1 fichero2 ... ficheroN]`

El comando `file` intenta identificar que tipo de formato tiene un fichero en particular. Debido a que no todos los ficheros tienen extensión o otras formas de identificarlos fácilmente, este comando realiza algunas comprobaciones rudimentarias para intentar comprender exactamente que contiene el fichero.

Hay que tener cuidado ya que es bastante posible que `file` realice una identificación incorrecta.

7.5 Comandos de edición

Esta sección trata de los comandos que alteran un fichero, realizando sobre el fichero operaciones concretas o mostrando estadísticas del mismo.

`grep [-nvwx] [-número] expresión [fichero1 fichero2 ... ficheroN]`

Uno de los comandos más útiles de Unix es el `grep`¹³, utilizado para buscar expresiones en un fichero de texto. La forma más sencilla de usarlo es:

```
/home/larry$ cat animales

Los animales son unas criaturas muy interesantes. Uno de mis animales favoritos es
el tigre, una temible bestia con grandes colmillos.
Tambien me gusta el leon--- realmente increible!
/home/larry$ grep igre animales
el tigre, una temible bestia con grandes colmillos.
/home/larry$
```

Una de los problemas de esta forma de usarlo, es que simplemente muestra las líneas que contienen la palabra a buscar, no aporta información acerca de donde buscar en el fichero, es decir el número de línea. Pero dependiendo de lo que se pretenda puede ser hasta más que suficiente, por ejemplo si se buscan los errores de la salida de un programa, se puede probar “`a.out | grep error`”, donde “`a.out`” es el nombre del programa.

Cuando es necesario conocer donde están las palabras a buscar, es decir el número de línea, hay que utilizar la opción “`n`”. Si lo que se desea es ver todas las líneas donde *no* se encuentra la expresión especificada, entonces utilice la opción “`v`”.

Otra posibilidad que ofrece el `grep` es la búsqueda de partes de una palabra, como en el ejemplo anterior que “`igre`” se equiparó con “`tigre`”. Para buscar sólo palabras completas hay que pasarle al `grep` la opción “`w`”, y para líneas completas la opción es “`x`”.

Si no se especifica ningún fichero (por ejemplo: “`grep igre`”), `grep` examinará la `stdin`.

¹³N. del T.: Abreviatura de `generalized regular expression parser`, analizador general de expresiones regulares.

wc [-clw] [*archivo1 archivo2 ... archivoN*]

El comando **wc**¹⁴ simplemente cuenta el número de palabras, líneas y caracteres en los archivos pasados como parámetros. Si no se especifica ningún archivo, operará sobre la **stdin**.

Los tres parámetros, “**clw**”, indican el elemento a contar: **c**aracteres, **l**íneas y **w** para palabras. Por ejemplo, “**wc -cw**” contará el número de caracteres y palabras, pero no el número de líneas. Si no se indica ningún parámetro **wc** cuenta todo: palabras, líneas y caracteres.

Se puede utilizar **wc** para saber el número de archivos de un directorio: “**ls | wc -w**”. Si se desea saber el número de archivos que acaban en **.c**, entonces ejecute “**ls *.c | wc -w**”.

spell [*archivo1 archivo2 ... archivoN*]

spell es el corrector ortográfico más sencillo de Unix generalmente para inglés americano¹⁵. **spell** es un filtro, igual que la mayoría de los comandos que hemos comentado antes, que toma el archivo de texto ASCII y devuelve todas las palabras que considera erróneas. **spell** opera sobre los archivos especificados, y si no se incluye ninguno, entonces utiliza la **stdin**.

También es posible que esté disponible en su máquina **ispell**, que es un corrector ortográfico un poco más sofisticado. **ispell** ofrece diversos vocabularios y un menú más manejable en caso de ejecutarlo con un archivo en la línea de comandos. También puede ejecutarse como un filtro si no se especifica ningún archivo.

La forma de trabajar con **ispell** es bastante obvia, pero si necesita más ayuda consulte la página del manual correspondiente: “**man ispell**”.

cmp *archivo1* [*archivo2*]

cmp **com**para dos archivos. El primero debe ser obligatoriamente pasado como parámetro, mientras que el segundo puede ser pasado como un segundo parámetro o leído desde la entrada estándar. **cmp** es muy sencillo, y simplemente dice donde se diferencian los dos archivos.

diff *archivo1 archivo2*

Uno de los comandos estándar más complejos de Unix es el **diff**. La versión GNU de **diff** tiene hasta ¡veinte opciones! en la línea de comandos. Es una versión mucho más potente que **cmp** y muestra cuales son las diferencias en lugar de decir simplemente donde está la primera.

Ya que una buena parte de las opciones del **diff** están más allá del alcance de este manual,

¹⁴N. del T.: Abreviatura de **w**ord **c**ount, contar palabras.

¹⁵Aunque hay versiones del corrector para diversas lenguas europeas, la copia que puede encontrarse en su máquina Linux es con toda probabilidad para inglés americano.

simplemente hablaremos del funcionamiento básico. Brevemente, `diff` toma dos parámetros y muestra las diferencias entre ellos línea a línea. Por ejemplo:

```
/home/larry$ cat rana
Los animales son una criaturas muy interesantes. Uno de mis animales favoritos es
el tigre, una temible bestia con grandes colmillos.
Tambien me gusta el leon--- realmente increible!
/home/larry$ cp rana sapo
/home/larry$ diff rana sapo
/home/larry$ cat perro
Los animales son una criaturas muy nterесantes. Uno de mis animales favoritos es

el tigre, una temible bestia con grandes colmillos.
Tambien me          gusta el leon--- realmente increible!
/home/larry$ diff rana perro
1c1,2
< Los animales son una criaturas muy interesantes. Uno de mis animales favoritos es
----
> Los animales son una criaturas muy nterесantes. Uno de mis animales favoritos es
>
3c4
< Tambien me gusta el leon--- realmente increible!
----
> Tambien me          gusta el leon--- realmente increible!
/home/larry$
```

Como puede observarse, `diff` no devuelve nada cuando los dos ficheros son iguales. Pero cuando hemos comparado dos ficheros diferentes, se muestra una cabecera de sección, “1c1,2” indicando que ha comparado la línea 1 del fichero de la izquierda, **rana**, con las líneas 1–2 de **perro** y las diferencias que ha encontrado. Luego ha comparado la línea 3 de **rana** con la línea 4 de **perro**. Aunque pueda parecer extraño que compare diferentes números de línea, es bastante eficiente ya que es posible que pueda haber algún retorno de línea de más en alguno de los ficheros.

```
gzip [-v#] [fichero1 fichero2 ... ficheroN]
gunzip [-v] [fichero1 fichero2 ... ficheroN]
zcat [fichero1 fichero2 ... ficheroN]
```

Estos tres programas se utilizan para comprimir y descomprimir datos. `gzip`, o GNU Zip, es un programa que lee de los ficheros originales y devuelve ficheros más pequeños, borra los ficheros especificados en la línea de comandos y los reemplaza con ficheros que tienen el mismo nombre pero con **.gz** añadido a su nombre.

```
tr cadena1 cadena2
```

El comando de “traducción de caracteres” opera sobre la entrada estándar—no acepta un nombre de fichero como parámetro. Reemplaza todas las ocurrencias de *cadena1* en la entrada con la *cadena2*. En comparación con llamadas tan sencillas como “`tr rana sapo`”, `tr` puede aceptar comandos más complejos. Por ejemplo, hay una forma rápida de convertir los caracteres en minúscula por otros en mayúscula :

```
/home/larry$ tr {:lower:} [:upper:]  
Esta es una frase RARA.  
ESTA ES UNA FRASE RARA.
```

`tr` es relativamente complejo y se usa normalmente en pequeños programas shell, como filtro.

Capítulo 8

Editando archivos con Emacs

FUNNY SOMETHING OR OTHER

8.1 ¿Qué es Emacs?

Para obtener algo en una computadora, necesita una forma de introducir texto en los archivos, y una manera de cambiar el texto que ya está en los archivos. Un **editor** es un programa para este tipo de tareas. **emacs** es uno de los editores más populares, en parte porque es muy fácil para un principiante hacer trabajos con él. (El editor de Unix clásico, el **vi**, se trata en el Apéndice A.)

Para aprender **emacs**, tiene que encontrar un archivo de texto (letras, números, etc.) cópielo a su directorio de usuario ¹ (no queremos modificar el archivo original, si éste contiene información importante), y luego llame a Emacs con el archivo:

```
/home/larry$ emacs LEAME
```

(Por supuesto, si decide copiar **/etc/rc**, **/etc/inittab**, o cualquier otro archivo, sustituya ese nombre de archivo por **LEAME**. Por ejemplo, si “**cp /etc/rc ~/rc**”, entonces “**emacs rc**”).

X

“Llamar” a Emacs puede tener efectos diferentes dependiendo en dónde lo haga. Desde una consola que muestra sólo caracteres de texto Emacs se apoderará de toda la consola. Si lo llama desde X, Emacs abrirá su propia ventana. Asumiré que lo está haciendo desde una consola de texto, pero todo sucede de la misma manera en la versión de X, lógicamente —simplemente sustituya la palabra “ventana” en los lugares en donde lea “pantalla”. ¡Además, recuerde que debe mover el puntero del ratón a la ventana de Emacs para escribir!

Su pantalla (o ventana, si está usando X) debería parecerse a la Figura 8.1. La mayor parte de la pantalla contiene su documento de texto, pero las dos últimas líneas son especialmente interesantes si está tratando de aprender Emacs. La penúltima línea (la que tiene una cadena larga de guiones) se denomina **línea de modo** (“mode line” en inglés).

¹ Por ejemplo, “**cp /usr/src/linux/LEAME ./LEAME**”

Figura 8.1 Emacs se inició con “emacs LEAME”

```
Linux kernel release 1.0

These are the release notes for linux version 1.0.  Read them carefully,
as they tell you what this is all about, explain how to install the
kernel, and what to do if something goes wrong.

WHAT IS LINUX?

Linux is a Unix clone for 386/486-based PCs written from scratch by
Linus Torvalds with assistance from a loosely-knit team of hackers
across the Net.  It aims towards POSIX compliance.

It has all the features you would expect in a modern fully-fledged
Unix, including true multitasking, virtual memory, shared libraries,
demand loading, shared copy-on-write executables, proper memory
management and TCP/IP networking.

It is distributed under the GNU General Public License - see the
accompanying COPYING file for more details.

INSTALLING the kernel:
-----Emacs: README                      (Fundamental)--Top-----
```

En mi línea de modo, ve “Top”. Debería decir “All”, y puede haber otras pequeñas diferencias. (A muchas personas les aparece la hora actual en la línea de modo). La línea inmediatamente inferior a la línea de modo se denomina **minibuffer**, o a veces el **área de eco**. Emacs usa el minibuffer para enviar mensajes al usuario, y ocasionalmente cuando es necesario, para leer información que introduce el usuario. De hecho, ahora mismo Emacs le está diciendo “Para obtener información acerca del Proyecto GNU y sus objetivos, teclee C-h C-p.” (sólo que en inglés). Ignórelo por ahora; no vamos a usar mucho el minibuffer por un tiempo.

Antes de realizar cualquier cambio en el texto del archivo, necesita aprender cómo moverse. El cursor deberá estar al principio del archivo, en la esquina superior-izquierda de la pantalla. Para avanzar, presione **C-f** (reteniendo la tecla **Control** mientras presiona “f”, para “forward”(avanzar)). Avanzará un carácter cada vez, y si retiene ambas teclas, la repetición automática de teclas de su sistema deberá surtir efecto en medio segundo aproximadamente. Notará como, cuando llega al fin de la línea, el cursor automáticamente se mueve a la próxima línea. **C-b** (para “backward”(retroceder)) tiene el comportamiento opuesto. Y, ya que estamos en ello, **C-n** y **C-p** le llevan a las líneas siguiente y anterior, respectivamente. ²

Usar las teclas de control es comúnmente la manera más rápida de moverse cuando está editando. El objetivo de Emacs es mantener sus manos sobre las teclas alfa-numéricas del teclado, donde se realiza la mayoría del trabajo. Sin embargo, si quiere, las teclas de movimiento deberán funcionar también.

De hecho, cuando usa X, debería ser capaz de ubicar el puntero del ratón y hacer “click” con el botón izquierdo para mover el cursor donde quiera. Sin embargo, esto es muy lento—¡tiene que mover la mano hasta el ratón! La mayoría de la gente que usa Emacs usa principalmente el teclado para moverse por el texto.

²En caso de que aún no lo haya notado, muchos de los comandos de movimiento de emacs consisten en combinar **Control** con una única letra mnemotécnica.

Use `C-p` y `C-b` para ir a la esquina superior-izquierda. Ahora mantenga `C-b` un poco más. Debería oír un sonido molesto de campana, y ver el mensaje “Beginning of buffer” (Principio del buffer) que aparece en el minibuffer. En este punto se puede preguntar, “¿Pero qué es un buffer?”

Cuando Emacs trabaja sobre un archivo, no trabaja realmente sobre el archivo en sí. En vez de eso, copia los contenidos del archivo en un área de trabajo especial de Emacs llamada **buffer**, donde puede modificar el contenido. Cuando ha acabado de trabajar, debe decirle a Emacs que guarde los buffers — en otras palabras, que escriba el contenido de los buffers en el archivo correspondiente. Hasta que haga esto, el archivo permanece sin cambiar, y el contenido de los buffers existe únicamente dentro de Emacs.

Con esto en mente, prepárese a insertar su primer carácter en el buffer. Hasta ahora, todo lo que hemos hecho ha sido “no destructivo”, este es un gran momento. Puede escoger cualquier carácter que quiera, pero si quiere hacer esto con estilo, yo sugiero usar una bonita y sólida, “X” mayúscula. Mientras lo teclea, eche un vistazo al principio de la línea de modo al pie de la pantalla. Cuando cambia los buffer de modo que sus contenidos no sean iguales que los del archivo sobre el disco, Emacs muestra dos asteriscos a principios de la línea de modo, para hacerle saber que el buffer ha sido modificado:

```
--*- Emacs:  algun_archivo.txt (Fundamental)--Top-----
```

Estos dos asteriscos se muestran tan pronto como modifica el buffer, y permanecen visibles hasta que guarde el buffer. Puede guardar los buffer muchas veces durante una sesión de edición —el comando para hacerlo es simplemente `C-x C-s` (presione `Control` y pulse “x” y “s” mientras la mantiene apretada, ¡probablemente ya se lo imaginó!). Es deliberadamente fácil de escribir, porque lo mejor es salvar sus buffers al principio y frecuentemente.

Ahora voy a enumerar más comandos, además de los que ya ha aprendido, y los puede practicar del modo que prefiera. Yo sugiero familiarizarse con ellos antes de proseguir:

<code>C-f</code>	Avanza un carácter.
<code>C-b</code>	Retrocede un carácter.
<code>C-n</code>	Va a la próxima línea.
<code>C-p</code>	Va a la línea anterior.
<code>C-a</code>	Va al comienzo de la línea.
<code>C-e</code>	Va al final de la línea.
<code>C-v</code>	Va a la próxima página/pantalla de texto.
<code>C-l</code>	Redibuja la pantalla, con la línea actual en el centro.
<code>C-d</code>	Borra este carácter (practica este).
<code>C-k</code>	Borra el texto desde aquí hasta el fin de línea.
<code>C-x C-s</code>	Salva el buffer en su archivo correspondiente.
<code>Retroceso</code>	Borra el carácter anterior (el último que escribiste).

8.2 Comenzar rápidamente en X



Si está interesado en editar unos archivos rápidamente, un usuario de X no tiene que ir mucho más allá de los menús en la parte superior de la pantalla:

Estos menús no están disponibles en el modo texto.

Cuando ejecute por primera vez Emacs, habrá cuatro menús en la parte superior de la pantalla: **Buffers**, **File**, **Edit**, y **Help**. Para usar un menú, simplemente mueva el puntero del ratón sobre el nombre (como **File**, haga click y retenga el botón izquierdo). Entonces, mueva el puntero a la acción que quiere y libere el botón del ratón. Si cambia de idea, retire el puntero del ratón del menú y libere el botón.

El menú **Buffers** enumera los diferentes archivos que han sido editados en esta instancia de Emacs. El menú **Files** muestra un grupo de comandos para cargar y guardar archivos —muchos de ellos se describirán más adelante. El menú **Edit** muestra algunos comandos para editar un buffer, y el menú **Help** debería dar la documentación en línea.

Notará que las equivalencias del teclado se enumeran junto a las opciones del menú. Puesto que, a largo plazo, éstas serán más rápidas, podría quererlas aprender. También, para bien o para mal, la mayoría de la funcionalidad de Emacs está *únicamente* disponible mediante el teclado —así que puede que le interese leer el resto de este capítulo.

8.3 Editando varios archivos al mismo tiempo

Emacs puede trabajar sobre más de un archivo a la vez. De hecho, el único límite sobre cuantos buffers puede contener Emacs es la cantidad real de memoria disponible en la máquina. El comando para traer un nuevo archivo a un buffer de Emacs es **C-x C-f**. Cuando lo teclee, se le pedirá un nombre de archivo en el minibuffer:

```
Find file (Buscar archivo):~/
```

La sintaxis, aquí, es la misma que la usada para especificar archivos desde la línea de comandos; las barras representan subdirectorios, `~` es su directorio de usuario. También consigue **terminación automática de nombre de archivo**, significa que si ha escrito suficiente de un nombre de archivo en la línea de comandos para identificar el archivo singularmente, puede simplemente presionar **Tab** para completarlo (o para ver las terminaciones posibles, si hay más de una). **Espacio** también tiene un papel en la terminación automática de nombres de ficheros en el minibuffer, parecido a **Tab**, pero dejaré que experimente para que averigüe cómo difieren las dos. Una vez que tiene el nombre completo en el minibuffer, presione **Intro**, y Emacs creará un buffer mostrando el archivo. En Emacs, este proceso es conocido como **encontrar** un archivo. Siga adelante y busque ahora algún otro archivo de texto sin importancia y tráigalo a Emacs (haga esto desde nuestro buffer original `algun_archivo.txt`). Ahora tiene un nuevo buffer; Supondré que se llama `otro_archivo.txt`, ya que no puedo ver su línea de modo.

Su buffer original parece haber desaparecido —probablemente se pregunta dónde fue. Está todavía dentro de Emacs, y puede volver a él con **C-x b**. Cuando teclee esto, verá que en el

minibuffer le pide el nombre un buffer al que cambiar, y nombra uno por defecto. El buffer por defecto lo consigue sólo con presionar `Intro` en la línea de comandos, sin escribir un nombre de buffer. El buffer por defecto al que cambiar, es siempre el más recientemente usado, para que cuando esté haciendo mucho trabajo entre dos buffers, `C-x b` tenga por defecto el “otro” buffer (salvándole de tener que escribir el nombre del buffer). Incluso si el buffer por defecto es el que quiere, debería probar a teclear su nombre de todos modos.

Note que consigue el mismo tipo de terminación automática que obtuvo al buscar un archivo: al pulsar `Tab` completa todo lo que puede del nombre de un buffer y así sucesivamente. Cada vez que se le pida algo en el minibuffer, es una buena idea ver si Emacs hace terminación automática. Aprovechando la terminación automática cuando se le ofrezca, ahorrará teclear mucho. Emacs comúnmente hace la terminación automática cuando elige un elemento de alguna lista predefinida.

Todo lo que ha aprendido para moverse y editar texto en el primer buffer se aplica a los nuevos. Siga adelante y cambie algún texto en el nuevo buffer, pero no lo guarde (es decir, no teclee `C-x C-s`).

Déjeme asumir que quiere desechar los cambios sin guardarlos en el archivo. El comando para esto es `C-x k`, que “mata” (kill) el buffer. Tecléelo ahora. Primero se le preguntará qué buffer matar, pero por defecto es el buffer actual, y casi siempre es el que se quiere matar, simplemente presione `Intro`. Entonces le preguntará si *realmente* quiere matar el buffer — Emacs siempre controla antes de matar un buffer que tiene cambios sin salvar. Simplemente escriba “yes” (sí) y presione `Intro`, si quiere matarlo.

Siga adelante y practique cargar archivos, modificarlos, guardarlos, y matar sus buffers. Por supuesto, cerciórese de no modificar ningún archivo de sistema importante de una forma que cause problemas ³, pero trate de tener por lo menos cinco buffers abiertos al mismo tiempo, para que se pueda dar el gusto de moverse entre ellos.

8.4 Terminando una sesión de edición

Cuando haya hecho su trabajo en Emacs, asegúrese de que se guarden todos los buffers que deben guardarse, y salga de Emacs con `C-x C-c`.

A veces `C-x C-c` le hará una pregunta o dos en el minibuffer antes de dejarle salir —no se alarme, simplemente conteste en las maneras obvias. Si piensa que podría volver a Emacs luego, no use `C-x C-c`; use `C-z`, que suspenderá Emacs. Puede volver luego con el comando “fg” del shell. Esto es más eficiente que detener a Emacs y comenzar varias veces, especialmente si tiene que editar los mismos archivos nuevamente.



Bajo X, presionar `C-z` reducirá a icono la ventana. Mire la sección sobre minimizar en el Capítulo 5. Esto le da dos formas de minimizar Emacs —la manera normal que ofrece el gestor de ventanas, y `C-z`. Recuerde, cuando minimice, un simple “fg” no traerá la ventana anterior —tendrá que usar el gestor de ventanas.

³De cualquier manera, si no es el usuario “root” de la máquina, no debería ser capaz de dañar el sistema, pero tenga cuidado igualmente.

8.5 La tecla Meta

Ha aprendido ya sobre una “tecla modificadora” en Emacs, la tecla `Control`. Hay una segunda, llamada la tecla **Meta**, que se usa casi tan frecuentemente. Sin embargo, no todos los teclados tienen su tecla Meta en el mismo lugar, y algunos ni siquiera la tienen. Lo primero que necesita hacer es encontrar dónde se encuentra su tecla Meta. Es probable que las teclas `Alt` de su teclado sean también teclas Meta, si usa un PC IBM o algún otro teclado que tenga una tecla `Alt`.

La forma de probar esto es mantener presionada una tecla que crea que puede ser una tecla Meta y teclear “x”. Si ve que un pequeño prompt aparece en el minibuffer (como esto: `M-x`) entonces la ha encontrado. Para librarse del prompt y regresar al buffer de Emacs, teclee `C-g`.

Si no consigue un prompt, entonces todavía queda una solución. Puede usar la tecla `Escape` como una tecla Meta. Pero en vez de mantenerla pulsada mientras teclea la próxima letra, tiene que pulsarla y soltarla rápidamente, y *entonces* teclee la letra. Este método funcionará tenga o no una tecla Meta verdadera, también es la manera más segura para hacerlo. Intente ahora pulsar ligeramente `Escape` y entonces teclee “x”. Debería conseguir otra vez ese pequeño prompt. Simplemente use `C-g` para salir. `C-g` es la manera general en Emacs para salir de algún lugar donde no quiere estar. Los fastidiosos y comunes pitidos son para hacerle saber que ha interrumpido algo, pero está bien, porque es lo que quería hacer cuando tecleó `C-g`.⁴

La notación `M-x` es análoga a `C-x` (ponga cualquier carácter en el lugar de la “x”). Si ha encontrado una verdadera tecla Meta, use ésta, de otra manera simplemente use la tecla `Escape`. Yo escribiré simplemente `M-x` y ud. tendrá que usar su propia tecla Meta.

8.6 Cortar, pegar, destruir y tirar

Emacs, como cualquier buen editor, le permite cortar y pegar bloques de texto. A fin de hacer esto, necesita una forma de definir el comienzo y fin del bloque. En Emacs, se hace esto estableciendo dos ubicaciones en el buffer, conocidas como **marca** y **puntero**. Para colocar la marca, vaya al lugar donde quiere que comience el bloque y teclee `C-SPC` (“SPC” significa `Espacio`, por supuesto). Debería ver el mensaje “Mark set” (Marca establecida) que aparece en el minibuffer⁵. Ahora la marca ha sido establecida en ese lugar. No habrá ningún indicador especial destacando este hecho, pero Ud. sabe dónde la ha puesto, y eso es lo que importa.

¿Y qué hay del **puntero**? Bien, resulta que ha colocado un puntero cada vez que ha movido el cursor, porque “puntero” simplemente se refiere a su ubicación actual en el buffer. En términos formales, el puntero es el punto donde se insertará el texto si escribe algo. Al colocar la marca, y luego moverse al final del bloque, ha definido un bloque de texto. Este bloque es conocido como la **región**. La región siempre significa el área entre la marca y el puntero.

El sólo hecho de definir la región no la deja disponible para pegar. Tiene que decirle a Emacs

⁴Ocasionalmente, un `C-g` no es suficiente para persuadir a emacs que realmente quiere interrumpir lo que hace. Simplemente insista, y normalmente Emacs volverá a un modo más cuerdo.

⁵Sobre algunos terminales, `C-SPC` no funciona. Para estas máquinas, debe usar `C-@`.

que lo copie para poder ser capaz de pegarlo. Para copiar la región, asegúrese de que la marca y el puntero están correctamente establecidos, y teclee **M-w**. Ahora ha sido grabada por Emacs. Para pegarlo en alguna otra parte, simplemente vaya allí y teclee **C-y**. Esto es conocido como **tirar** el texto en el buffer.

Si quiere mover el texto de la región a alguna otra parte, teclee **C-w** en vez de **M-w**. Esto **matará** la región —todo el texto dentro de ella desaparecerá. De hecho, se ha guardado del mismo modo que si hubiera usado **M-w**. Puede tirar de nuevo con **C-y**, como siempre. El lugar donde emacs guarda todo este texto es conocido como el **círculo de muerte**. Algunos editores lo llaman el “porta papeles” o el “buffer de pegado”.

Existe otra manera para cortar y pegar: cuando usa **C-k** para matar hasta el final de una línea, el texto matado se guarda en el círculo de muerte. Si mata más de una línea seguida, se guardarán todas juntas en el círculo de muerte, para que la próxima tirada pegue todas las líneas al mismo tiempo. Por ello, casi siempre es más rápido usar repetidas veces **C-k** para matar algún texto, que establecer la marca y el puntero y usar **C-w**. Sin embargo, de una u otra manera funcionará. Es realmente una cuestión de preferencia personal cómo lo hace.

8.7 Buscar y reemplazar

Hay varias maneras para buscar texto en Emacs. Muchas son más bien complejas, y no merece la pena tratarlas aquí. La más fácil y la más entretenida es usar **isearch**.

“Isearch” se refiere a “incremental search” (búsqueda incremental). Supongamos que quiere buscar la cadena “tábano” en el siguiente buffer:

```
Yo estaba temeroso que nos quedáramos sin gasolina, cuando mi tácito pasajero
exclamó ‘‘Auch un aguijón! Hay un tábano aquí dentro! ’’.
```

Debería moverse al comienzo del buffer, o por lo menos a algún punto que sabe que está antes de la primera aparición de la palabra, “tábano”, y teclear **C-s**. Eso le pondrá en el modo de búsqueda isearch. Ahora comience a escribir la palabra que está buscando, “tábano”. Pero tan pronto como escribe la “t”, ve que Emacs ha saltado a la primera aparición de “t” en el buffer. Si la cita de arriba es todo el contenido del buffer, entonces la primera “t” es de la palabra “temeroso”. Ahora escriba la “á” de “tábano”, y Emacs saltará sobre “tácito”, que contiene la primer ocurrencia de “tá”. Y finalmente, “b” consigue “tábano”, sin haber tenido que escribir la palabra entera.

Lo que hace en una isearch es definir una cadena para buscarla. Cada vez que agrega un carácter al final de la cadena, el número de posibles cadenas se reduce, hasta que haya escrito lo suficiente para definir la cadena singularmente. Una vez que ha encontrado la palabra, puede salir de la búsqueda con **Intro** o cualquiera de los comandos normales de movimiento. Si piensa que la cadena que buscas esta atrás en el buffer, entonces debería usar **C-r**, que hace isearch hacia atrás.

Si encuentra una palabra, pero no es la que buscaba, entonces presione **C-s** nuevamente mientras todavía esté en la búsqueda. Esto le moverá hasta la próxima palabra coincidente, cada vez que lo haga. Si no existe una próxima palabra, dirá que la búsqueda fracasó, pero si presiona **C-s**

nuevamente en este punto, la búsqueda volverá a comenzar desde el principio del buffer. Se puede decir lo opuesto de `C-r` —comienza al final del buffer.

Intente introducir un buffer de texto en inglés y haga un isearch para la cadena “the”. Primero teclee todos los “the” que quiera y luego use `C-s` para ir a todas las apariciones. Note que también aparecerán palabras como “them”, dado que también contiene la subcadena “the”. Para buscar un único “the”, deberá agregar un espacio al final de la cadena de búsqueda. Puede agregar nuevos caracteres a la cadena en cualquier punto de la búsqueda, después tiene que presionar `C-s` repetidamente para encontrar las próximas palabras coincidentes. Puede usar también la `Retroceso` o `Supr` para quitar caracteres de la cadena en cualquier punto de la búsqueda, y presionando `Intro` sale de la búsqueda, dejándole en la última coincidencia.

Emacs también permite reemplazar todas las apariciones de una cadena con alguna nueva cadena — esto es conocido como **query-replace** (preguntar-reemplazar). Para invocarlo, teclee `query-replace` y `Intro`.

Como se hace terminación automática sobre el nombre del comando, una vez que has escrito “query-re”, puede simplemente presionar `Tab` para terminarlo. Digamos que desea reemplazar todas las ocurrencias de “tábano” por “mosca”. En el prompt “Query replace:” (preguntar-reemplazar), escriba “tábano”, y presione `Intro`. Entonces aparecerá el prompt nuevamente, y deberá introducir “mosca”. Entonces Emacs recorrerá el buffer, parando a cada aparición de la palabra “tábano”, y preguntando si quiere reemplazarla. Simplemente presione en cada instancia “y” o “n”, por “Yes” o “No”, hasta que termine. Si no entiende esto mientras lo lee, pruébelo.

8.8 ¿Qué es lo que ocurre realmente?

Realmente, todas estas **teclas ligadas** que ha aprendido son los atajos a funciones de Emacs. Por ejemplo, `C-p` es una manera abreviada de decirle a Emacs que ejecute la función interna `previous-line` (línea previa). Sin embargo, todas estas funciones internas pueden ser llamadas por el nombre, usando `M-x`. Si olvida que `previous-line` está ligado a `C-p`, puede escribir simplemente `M-x previous-line Intro`, y se moverá una línea hacia arriba. Pruebe esto ahora, para que comprenda como `M-x previous-line` y `C-p` son realmente la misma cosa.

El diseñador de Emacs comenzó desde la base hacia arriba, primero definió un lote completo de funciones internas, y entonces les asoció o ligó ciertas teclas a las más comúnmente usadas. A veces es más fácil llamar a una función explícitamente con `M-x` que recordar a qué tecla está ligada. La función `query-replace`, por ejemplo, está ligada a `M-%` en algunas versiones de Emacs. Pero ¿quién puede recordar tan rara combinación? A menos que use `query-replace` muy frecuentemente, es más fácil simplemente llamarla con `M-x`.

La mayoría de las teclas que pulsa son letras, cuya función es ser insertadas en el texto del buffer. Cada una de esas teclas está **ligada** a la función `self-insert-command`, que no hace nada más que insertar la letra en el buffer. Las combinaciones que usan la tecla `Control` con una letra generalmente están ligadas a funciones que hacen otras cosas, como mover el cursor. Por ejemplo, `C-v` está ligada a una función llamada `scroll-up` (avanzar página), que mueve el buffer una

pantalla hacia arriba (lo que quiere decir que su posición en el buffer se mueve hacia *abajo*, por supuesto).

Si alguna vez quisiera realmente insertar un carácter de Control en el buffer, entonces, ¿cómo lo haría? Después de todo, los caracteres de Control son caracteres `ASCII`, aunque rara vez usados, y puede querer tenerlos en un archivo. Hay una manera para impedir que los caracteres de Control sean interpretados como comandos por Emacs. La tecla `C-q`⁶ está ligada a una función especial llamada `quoted-insert` (insertar lo citado). Todo lo que `quoted-insert` hace es leer la próxima tecla e insertarla literalmente en el buffer, sin tratar de interpretarla como un comando. Así es cómo puede poner los caracteres de Control en sus archivos usando Emacs. ¡Naturalmente, la manera de insertar un `C-q` es presionar `C-q` dos veces!

Emacs también tiene muchas funciones que no están ligadas a ninguna tecla. Por ejemplo, si escribe un mensaje largo, y no quiere tener que presionar `Intro` al final de cada línea. Puede hacer que emacs lo haga por Ud. (de hecho puede hacer que Emacs haga cualquier cosa por Ud.) —el comando para hacerlo se llama `auto-fill-mode` (modo de auto llenado, pero no está ligado a ninguna tecla por defecto. A fin de invocar este comando, debe escribir “`M-x auto-fill-mode`”. “`M-x`” es la tecla usada para llamar a funciones por el nombre. Podría usarlo para llamar a funciones como `next-line` y `previous-line`, pero eso sería muy ineficaz, ya que esas funciones están ligadas a `C-n` y `C-p`.

A propósito, si mira su línea de modo después de invocar `auto-fill-mode`, notará que la palabra “Fill” se ha agregado al lado derecho. Mientras esté allí, Emacs llenará (rellenará) el texto automáticamente. Puede desactivarlo escribiendo “`M-x auto-fill-mode`” nuevamente —es un comando de palanca.

La incomodidad de escribir largos nombres de función en el minibuffer disminuye porque Emacs hace terminación automática en los nombres de funciones de la misma manera que lo hace en los nombres de archivo. Por lo tanto, rara vez debería encontrarse escribiendo el nombre entero de la función letra a letra. Si no está totalmente seguro de si puede o no usar la terminación automática, simplemente presione `Tab`. No puede hacer daño: lo peor que puede suceder es que consiga un carácter Tab, y si es afortunado, resultará que puede usar la terminación automática.

8.9 Pidiendo ayuda a Emacs

Emacs tiene extensas facilidades de ayuda —tan extensas de hecho, que sólo podemos comentarlas un poco aquí. A las facilidades de ayuda más básicas se accede tecleando `C-h` y luego una única letra. Por ejemplo, `C-h k` muestra la ayuda sobre una tecla (le pide que presiones una tecla, y entonces le dice lo que esa tecla hace). `C-h t` abre un breve manual sobre Emacs. Más importante aún, `C-h C-h C-h` le da ayuda sobre la ayuda, para decirle que está disponible una vez que ha tecleado `C-h` por primera vez. Si sabe el nombre de una función de Emacs (`save-buffer` (grabar el buffer), por ejemplo), pero no puede recordar cuál es la combinación de teclas a la que está ligada, use `C-h w`, para “`where-is`” (dónde está), y escriba el nombre de la función. O, si quiere conocer

⁶Llamamos a `C-q` una “tecla”, aunque se produce manteniendo presionada la tecla `Control` y presionando “q”, porque es un único carácter `ASCII`.

qué hace una función con detalle, use `C-h f`, que preguntará por un nombre de función.

Recuerde, que como Emacs hace terminación automática del nombre de la función, realmente no tiene que estar seguro de cómo se llama para pedir ayuda sobre ella. Si piensa que puedes adivinar la palabra con la que podría comenzar, teclee ésa y presione `Tab` para ver si se completa. Si no, vuelva atrás e intente otra cosa. Lo mismo ocurre con los nombres de archivo: aún cuando no pueda recordar del todo como nombró cierto archivo al que no ha accedido en tres meses, puede probar y usar la terminación automática para averiguar si está en lo cierto. Usa la terminación automática como una forma de preguntar, y no sólo como una manera de teclear menos.

Hay otros caracteres que puede pulsar después de `C-h`, y con cada uno consigue ayuda de una manera diferente. Los que usará más frecuentemente son `C-h k`, `C-h w`, y `C-h f`. Una vez que esté más familiarizado con Emacs, otra para probar es `C-h a`, que le pregunta por una cadena y le comenta todas las funciones que tienen esa cadena como parte de su nombre (la “a” sería para “apropos” [a propósito], o “about” [acerca]).

Otra fuente de información es el lector de documentación **Info**. Info es demasiado complejo para tratarlo aquí, pero si está interesado en explorarlo por si mismo, teclee `C-h i` y lea el párrafo en la parte superior de la pantalla. Le dirá cómo conseguir más ayuda.

8.10 Especializando buffers: Modos

Los buffers de Emacs tienen **modos** asociados⁷. La razón para esto, es que sus necesidades cuando escribe un mensaje de correo son muy diferentes de sus necesidades cuando, por ejemplo, escribe un programa. Mejor que tratar de crear un editor que satisfaga cada necesidad particular en todo momento (que sería imposible), el diseñador de Emacs⁸ eligió hacer que Emacs se comporte de manera diferente dependiendo de qué hace Ud. en cada buffer individual. Así, los buffers tienen modos, cada uno diseñado para alguna actividad específica. Los aspectos principales que distinguen un modo de otros son las combinaciones de teclas, pero también pueden existir otras diferencias.

El modo más básico es el modo **Fundamental**, que realmente no tiene ningún comando especial. De hecho, esto es todo lo que Emacs dice sobre el Modo Fundamental:

Modo Fundamental:

Modo mayor no especializado para nada en particular. Los otros modos mayores son definidos por comparación con este.

Obtuve así esa información: Teclee `C-x b`, que es **switch-to-buffer** (cambiar al buffer), e introduje “foo” cuando se me preguntó por un nombre de buffer al que cambiar. Como no había anteriormente ningún buffer llamado “foo”, Emacs creó uno y me cambió a él. Estaba en el modo **fundamental** por defecto, pero si no lo hubiese estado, podría haber tecleado “M-x fundamental-mode” para que estuviese. Todos los nombres de modo tienen un comando

⁷para colmo de males, hay “Modos Mayores” y “Modos Menores”, pero en este momento no necesita conocer nada acerca de esto.

⁸Richard Stallman, a veces también conocido como “rms”, porque es su login.

llamado `<nombre-de-modo>-mode` que pone el buffer actual en ese modo. Entonces, para averiguar más sobre este modo mayor, tecleé `[C-h m]`, que consigue ayuda sobre el modo mayor actual del buffer en que está.

Hay un modo ligeramente más útil llamado `text-mode` (modo `texto`, que tiene los comandos especiales `[M-S]`, para `centrar párrafo`, y `[M-s]`, que invoca `centrar línea`. `[M-S]`, a propósito, significa exactamente lo que piensa: mantenga pulsadas la `[Meta]` y la tecla `[Shift]`, y presiona “S”.

Pero no me tome la palabra en esto — cree un nuevo buffer, póngalo en modo `texto`, y teclee `[C-h m]`. Puede que no entienda todo lo que Emacs le diga cuando lo haga, pero debería ser capaz de conseguir sacar alguna información útil de ello.

Esto es una introducción a algunos de los modos más comúnmente usados. Si los usa, asegúrese de que teclea en cada uno `[C-h m]` alguna vez, para averiguar más sobre cada modo.

8.11 Modos de programación

8.11.1 Modo C

Si usa Emacs para programar en el lenguaje C, puede conseguir que él le haga toda la indentación automáticamente. Los archivos cuyos nombres terminan en `.c` o `.h` se abrirán automáticamente en el **modo c**. Esto significa que ciertos comandos especiales de edición, útiles para escribir programas en C, están disponibles. En el modo C, `[Tab]` está ligado a `c-indent-command` (indentar comandos c). Esto significa que presionando la tecla `[Tab]` no inserta realmente un carácter de Tabulación. En cambio, si presiona `[Tab]` en cualquier parte de una línea, Emacs automáticamente indenta esta línea correctamente para su ubicación en el programa. Esto implica que Emacs sabe algo sobre la sintaxis de C, (aunque nada sobre semántica — ¡no puede asegurar que su programa no tenga errores!).

Para hacer esto, asuma que las líneas anteriores están indentadas correctamente. Esto significa que si en la línea anterior falta un paréntesis, un punto y coma, llaves, o cualquier otra cosa, Emacs indentará la línea actual de una manera peculiar inesperada. Cuando vea que hace esto, sabrá que debe buscar un error de puntuación en la línea anterior.

Puede usar esta característica para verificar que ha puntuado correctamente sus programas — en vez de leer el programa entero buscado problemas, simplemente comience a indentar las líneas desde arriba hasta abajo con `[Tab]`, y cuando alguna se indenta de forma rara, verifique las líneas inmediatamente anteriores. En otras palabras, ¡deje a Emacs hacer el trabajo por Ud.!

8.11.2 Modo Scheme

Este es un modo mayor que no le servirá de nada a menos que tenga un compilador o un interprete para el lenguaje de programación Scheme en su sistema. Tener uno no es tan normal como, digamos, un compilador de C, pero se está haciendo cada vez más común, así que lo trataremos también. Mucho de lo que es cierto para el modo Scheme es también cierto para el modo Lisp, si prefiere escribir en Lisp .

Bien, para complicar las cosas, Emacs viene con dos modos Scheme diferentes, porque la gente no podía decidir cómo querían que funcionara. El que estoy describiendo se llama `cmuscheme`, y luego, en la sección personalizando a Emacs, hablaré de cómo puede haber dos modos Scheme diferentes y qué hacer acerca de ello. Por ahora, no se preocupe si las cosas en su Emacs no coinciden con las que yo digo aquí. Un editor personalizable significa un editor impredecible, ¡y no hay vuelta de hoja!

Puede ejecutar un proceso Scheme interactivo en Emacs con el comando `M-x run-scheme`. Esto crea un buffer llamado “`*scheme*`”, que tiene el prompt habitual de Scheme. Puede teclear expresiones de Scheme en el prompt, presionar `[Intro]`, y Scheme las evaluará y mostrará la respuesta. Así, a fin de interactuar con el proceso de Scheme, podrá simplemente escribir todas sus aplicaciones y definiciones de función en el prompt. Es posible que haya escrito previamente código fuente Scheme en un algún archivo, y sería más fácil hacer su trabajo en el archivo y enviar las definiciones al buffer de proceso Scheme según sea necesario.

Si el archivo fuente termina en `.ss` o `.scm`, automáticamente se abrirá en el **modo Scheme** cuando lo encuentre con `[C-x C-f]`. Si por alguna razón, no surge en el modo Scheme, puede hacerlo a mano con `M-x scheme-mode`. Este modo `scheme` no es lo mismo que el buffer que ejecuta el proceso Scheme; más bien, el que el buffer de código fuente esté en modo `scheme` significa que tiene comandos especiales para comunicar con el buffer de proceso.

Si está dentro de la definición de una función en el buffer de código fuente Scheme y teclea `[C-c C-e]`, entonces esa definición será “enviada” al buffer de proceso —exactamente como si lo hubiera tecleado Ud. mismo. `[C-c M-e]` envía la definición y entonces le lleva al buffer de proceso para hacer algo de trabajo interactivo. `[C-c C-l]` carga un archivo de código Scheme (éste funciona desde el buffer de proceso o el buffer de código fuente). Y como otros modos de lenguajes de programación, al presionar `[Tab]` en cualquier lugar de una línea de código se indentará correctamente esa línea.

Si está en el prompt del buffer de proceso, puede usar `[M-p]` y `[M-n]` para moverse entre sus comandos anteriores (también conocido como la **historia de entrada**). Así que si está depurando la función ‘`rotar`’, y ya lo ha aplicado a los argumentos en el buffer de proceso, como:

```
>(rotar '(a b c d e))
```

entonces puede recuperar ese comando anterior tecleando `[M-p]` en el prompt. Aquí no debería ser necesario volver a escribir expresiones largas en el prompt de Scheme —hábituese a usar la historia de entrada y ahorrará mucho tiempo.

Emacs conoce bastantes lenguajes de programación: C, C++, Lisp, y Scheme son simplemente algunos. Generalmente, sabe cómo indentarlos de forma intuitiva.

8.11.3 Modo de correo

También puede editar y enviar correo en Emacs. Para entrar en un buffer de correo, teclee `[C-x m]`. Necesita llenar los campos **To:** (A:) y **Subject:** (Asunto:), y entonces use `[C-n]` para ir, por debajo de la línea de separación, al cuerpo del mensaje (que está vacío cuando comienza por primera vez). No cambie o borre la línea de separación, o sino Emacs no será capaz de enviar su correo —use

esa línea para distinguir el encabezamiento del correo, que le dice dónde enviar el correo, de los contenidos del mensaje.

Puede escribir lo que quiera por debajo de la línea de separación. Cuando esté listo para enviar el mensaje, simplemente teclee `C-c C-c`, y Emacs lo enviará y hará que el buffer de correo desaparezca.

8.12 Como ser más eficiente aún

Los usuarios experimentados de Emacs son fanáticos de la eficiencia. ¡De hecho, frecuentemente acaban derrochando mucho tiempo buscando formas para ser más eficientes!. No quiero que le suceda esto, aunque hay algunas cosas fáciles con las que puede llegar a ser un mejor usuario de Emacs. A veces los usuarios experimentados hacen que los novatos se sientan tontos por no saber todos estos trucos —por alguna razón, la gente llega a hacerse religiosas sobre el uso “correcto” de Emacs. Allá vamos:

Cuando se mueve de un lado a otro, usa los medios más rápidos disponibles. Ud. sabe que `C-f` es **forward-char** (un carácter hacia adelante) —¿suponía que `M-f` es **forward-word** (una palabra hacia adelante)? `C-b` es **backward-char** (un carácter hacia atrás). ¿Supone qué hace `M-b`? Sin embargo, esto no es todo, puede avanzar una frase cada vez con `M-e`, siempre que escriba sus frases de modo que haya siempre dos espacios después del punto final (de otra manera Emacs no puede distinguir donde termina una frase y comienza la siguiente). `M-a` es **backward-sentence** (una frase atrás).

Si ve que usa repetidamente `C-f` para llegar al final de la línea, avergüéncese, y asegúrese de usar `C-e` en su lugar, y `C-a` para ir al principio de la línea. Si usa muchos `C-n` para bajar pantallas de texto, avergüéncese mucho, y usa `C-v` siempre. Si usa repetidamente `C-p` para avanzar pantallas, no se atreva a enseñar la cara, y use `M-v` en su lugar.

Si se está acercando al final de una línea y se da cuenta de que hay una palabra mal tecleada o de que se ha olvidado alguna en algún lugar anterior de la línea, *no* use la `Retroceso` o `Supr` para volver a ese punto. Eso requeriría volver a escribir porciones enteras de texto perfecto. En vez de eso, use combinaciones de `M-b`, `C-b`, y `C-f` para moverse a la ubicación precisa del error, arréglole, y entonces use `C-e` para moverse al fin de la línea nuevamente.

Cuando tiene que escribir un nombre de archivo, nunca teclee el nombre completo. Solamente escriba lo suficiente para identificarlo singularmente, y deje que Emacs termine el trabajo presionando `Tab` o `Espacio`. ¿Por qué teclear de más cuando puede derrochar ciclos de CPU en su lugar?

Si escribe algún tipo de texto simple, y de algún modo su auto-llenando (auto-filling) lo ha fastidiado, use `M-q`, que es **rellenado de párrafo** en los modos de texto comunes. Esto “ajustará” el párrafo en el que está, como si hubiese sido llenado línea a línea, pero sin tener que liarse haciéndolo a mano. `M-q` trabajará desde dentro del párrafo, o desde su comienzo o final.

A veces es útil usar `C-x u`, (**undo** (deshacer)), que tratará de “deshacer” el (los) último(s) cambio(s) que hizo. Emacs decidirá cuanto deshacer; habitualmente decide muy inteligentemente. Llamándolo repetidamente deshará más y más, hasta que Emacs no pueda recordar qué cambios se

hicieron.

8.13 Personalizando Emacs

¡Emacs es *tan* grande, y *tan* complejo, que de hecho tiene ¡su propio lenguaje de programación! No bromeo: para personalizar Emacs ajustándolo a sus necesidades, tiene que escribir programas en este lenguaje. Se llama Emacs Lisp, y es un dialecto de Lisp, así que si tiene experiencia previa en Lisp, le parecerá bastante amistoso. Si no, no se preocupe: no voy a profundizar mucho, porque definitivamente se aprende mejor practicando. Para aprender realmente a programar Emacs, deberá consultar las páginas de información de Emacs Lisp, y leer mucho código fuente de Emacs Lisp.

La mayor parte de la funcionalidad de Emacs está definida en archivos de código de Emacs Lisp⁹. La mayoría de estos archivos se distribuyen con Emacs y colectivamente son conocidos como la “Biblioteca de Emacs Lisp”. La ubicación de esta biblioteca depende de cómo se instaló Emacs en su sistema —son ubicaciones comunes `/usr/lib/emacs/lisp`, `/usr/lib/emacs/19.19/lisp/`, etc. El **19.19** es el número de versión de Emacs, y podría ser diferente en su sistema.

No necesita hurgar por su sistema de archivos buscando la biblioteca de lisp, porque Emacs tiene la información almacenada internamente, en una variable llamada **load-path** (trayectoria de carga). Para averiguar el valor de esta variable, es necesario **evaluarla**; esto es, hacer que el intérprete de lisp de Emacs consiga su valor. Hay un modo especial para evaluar las expresiones de Lisp en Emacs, llamado **modo lisp interactivo** (lisp-interaction-mode). Comúnmente, hay un buffer llamado “*scratch*” que está ya en este modo. Si no lo puede encontrar, cree un nuevo buffer con cualquier nombre, y escriba `M-x lisp-interaction-mode` dentro de él.

Ahora tiene un espacio de trabajo para interactuar con el intérprete Lisp de Emacs. Teclee esto:

```
load-path
```

y entonces presione `[C-j]` al finalizar. En el modo Lisp interactivo, `[C-j]` está ligado a `eval-print-last-sexp` (evaluar-imprimir-última-sexp). Una “sexp” es una “**s-expresión**”, lo que significa un grupo balanceado de paréntesis, incluido el caso de que no haya ninguno. Bueno, esto es simplificarlo un poco, pero irá entendiendo que son según programe con Emacs Lisp. De cualquier manera, al evaluar `load-path` debería conseguir algo como esto:

```
load-path [C-j]
("/usr/lib/emacs/site-lisp/vm-5.35" "/home/kfogel/elithp"
"/usr/lib/emacs/site-lisp" "/usr/lib/emacs/19.19/lisp")
```

Por supuesto, no tendrá el mismo aspecto en cada sistema, puesto que es dependiente de cómo se instaló Emacs. El ejemplo de arriba viene de mi PC 386 que funciona con Linux. Como indica lo anterior, `load-path` es una lista de cadenas. Cada cadena nombra un directorio que podría contener archivos de Emacs Lisp. Cuando Emacs necesita cargar un archivo de código Lisp, va buscándolo en cada uno de estos directorios, en orden. Si un directorio se nombra pero no existe en el sistema de archivos, Emacs simplemente lo ignora.

⁹ A veces llamados no oficialmente “Elisp”.

Cuando Emacs arranca, automáticamente trata de cargar el archivo **.emacs** desde su directorio de usuario. Por lo tanto, si quiere hacer personalizaciones en Emacs, deberá ponerlas en **.emacs**. La personalización más común son las teclas ligadas, así que aquí está cómo hacerlo:

```
(global-set-key "\C-cl" 'goto-line)
```

global-set-key (fijar teclas globalmente) es una función de dos argumentos: la tecla a la que ha de ser ligada, y la función a la que ligarla. La palabra “**global**” significa que esta tecla ligada tendrá efecto en todos los modos mayores (hay otra función, **local-set-key** (fijar teclas localmente), que liga una tecla en un único buffer). Arriba, he ligado **C-c l** a la función **goto-line** (ir a tal línea). La tecla se describe usando una cadena. La sintaxis especial “**\C-<carácter>**” significa mantener pulsada la tecla **Control** mientras se presiona **<carácter>**. Así mismo, “**\M-<carácter>**” indica la tecla **Meta**.

Todo eso está muy bien, ¿pero cómo supe que el nombre de la función era “**goto-line**”? Puedo saber que quiero ligar **C-c l** a alguna función que pregunta por un número de línea y mueve el cursor a esa línea, pero ¿cómo hice para averiguar el nombre de esa función?

Aquí es donde intervienen las facilidades de ayuda de Emacs. Una vez que ha decidido qué tipo de función busca, puede usar Emacs para rastrear su nombre exacto. He aquí una manera rápida y sucia para hacerlo: puesto que Emacs completa los nombres de función, simplemente escriba **C-h f** (que es **describe-function** (describir función), recuérdelo), y entonces presione **Tab** sin escribir nada más. Esto pide a Emacs que complete la cadena vacía —en otras palabras, ¡la terminación automática se corresponderá con cada una de las funciones!. Puede tardar un momento en construir la lista de funciones, ya que Emacs tiene muchas funciones internas, pero mostrará todo lo que entre en la pantalla cuando esté listo.

En este momento presione **C-g** para abandonar la función **describir función**. Habrá un buffer llamado “***Completions***”, que contiene la lista de terminaciones automáticas que acaba de generar. Cambie a este buffer. Ahora puede usar **C-s**, **isearch**, para buscar las funciones probables. Por ejemplo, es una suposición segura que una función que pregunta por un número de línea y entonces va a esa línea contendrá la cadena “**line** (línea)” en su nombre. Por lo tanto, simplemente comience buscando la cadena “**line**”, y acabará encontrando lo que busca.

Si quiere otro método, puede usar **C-h a**, **command-apropos**, para mostrar todas las funciones cuyos nombres se ajustan a la cadena dada. La salida de **command-apropos** es un poco más difícil de clasificar, que simplemente buscar una lista de terminación automática, en mi opinión, pero puede encontrar que tiene distintas sensaciones. Pruebe ambos métodos y a ver que opina.

Siempre existe la posibilidad de que Emacs no tenga ninguna función predefinida para hacer lo que está buscando. En esta situación, tiene que escribir la función Ud. mismo. No voy a hablar de cómo hacer eso —debería buscar en la biblioteca de Emacs Lisp ejemplos de definiciones de función, y leer las paginas Info sobre Emacs Lisp. Si resulta que conoce a un gurú local de Emacs, pregúntele cómo hacerlo. Definir sus propias funciones de Emacs no es un gran asunto —para darle una idea, yo he escrito 131 de ellas durante más o menos el último año. Requiere un poco de práctica, pero la curva de aprendizaje no es empinada.

Otra cosa que la gente hace a menudo en su **.emacs** es asignar a ciertas variables los valores preferidos. Por ejemplo, ponga esto en su **.emacs** y entonces inicie un nuevo Emacs:

```
(setq inhibit-startup-message t)
```

Emacs verifica el valor de la variable `inhibit-startup-message` (bloquear mensaje de arranque) para decidir si muestra la información sobre la versión y la falta de garantía cuando arranca. La expresión de Lisp de arriba usa el comando `setq` para asignar a esa variable el valor ‘t’, que es un valor especial de Lisp que significa **true** (verdadero). Lo contrario de ‘t’ es ‘nil’ (nada o nulo), que es el valor **false** (falso) designado en Emacs Lisp. He aquí dos cosas que están en mi `.emacs` que podría encontrar útiles:

```
(setq case-fold-search nil);causa la insensibilidad a mayúsculas y minúsculas para
la búsqueda
; ;; Hacer indentar los programas C de la manera que me gusta:
(setq c-indent-level 2)
```

La primera expresión hace que las búsquedas (incluyendo `isearch`) sean insensibles a mayúsculas y minúsculas; esto es, la búsqueda encontrará versiones de un mismo carácter sea mayúscula o minúscula aunque la cadena de búsqueda contenga únicamente la versión en minúscula. La segunda expresión establece que la indentación por defecto para las sentencias en lenguaje C sea un poco menor de lo normal—esto es solamente una preferencia personal; encuentro que esto hace el código C más legible.

El carácter de comentario en Lisp es “;”. Emacs ignora cualquier cosa que siga a uno de éstos, a menos que aparezca dentro de una cadena literal, como esta:

```
;;estas dos líneas son ignoradas por el intérprete de Lisp, pero la
; ;; s-expression que le sigue se evaluará totalmente:
(setq alguna-cadena-literal "Una pausa torpe; sin ningún propósito.")
```

Es una buena idea comentar sus cambios en los archivos Lisp, porque seis meses después no se acordará en qué estaba pensando cuando los modificó. Si el comentario aparece sólo en una línea, precédalo con dos punto y coma. Esto ayuda a Emacs a indentar los archivos Lisp correctamente.

Puede encontrar información sobre las variables internas de Emacs de las mismas formas que con las funciones. Use `C-h v`, `describe-variable` para hacer una lista de terminación automática, o use `C-h C-a`, `apropos`. `apropos` difiere de `C-h a`, `command-apropos`, en que muestra variables y funciones en vez de solamente funciones.

La extensión por defecto para los archivos de Emacs Lisp es `.el`, como en `c-mode.el`. Sin embargo, para hacer que el código Lisp se ejecute más rápido, Emacs permite que sea **byte-compiled** (compilado a un formato interno), y estos archivos de código Lisp compilado terminan en `.elc` en vez de `.el`. La excepción a esto es su archivo `.emacs`, que no necesita la extensión `.el` porque Emacs sabe buscarlo para arrancar.

Para cargar un archivo de código Lisp interactivamente, use el comando `M-x load-file` (cargar archivo). Le preguntará por el nombre del archivo. Para cargar archivos Lisp desde dentro de otros archivos Lisp, haga esto:

```
(load "c-mode");fuerza a Emacs a cargar el contenido de c-mode.el o .elc
```

Emacs añadirá la extensión **.elc** al nombre del archivo e intentará encontrarlo en algún lugar del `load-path`. Si falla, lo intenta con la extensión **.el**; si falla esto, usa la cadena literal tal y como es pasada a `load`. Puede compilar (byte-compile) un archivo con el comando `M-x byte-compile-file`, pero si modifica el archivo a menudo, probablemente no merezca la pena. Sin embargo no debería compilar de esa manera su **.emacs**, ni siquiera darle la extensión **.el**.

Después de que **.emacs** se ha cargado, Emacs busca un archivo llamado **default.el** para cargarlo. Comúnmente se ubica en un directorio en la trayectoria de carga (`load-path`) llamado **site-lisp** o **local-elisp** o algo parecido (ver el ejemplo `load-path` que di hace un rato). La gente que mantiene Emacs en un sistema multiusuario usa **default.el** para hacer cambios que afectarán los Emacs de todos, puesto que todos los Emacs lo cargan después de los **.emacs** personales. **default.el** no debería ser compilado, ya que tiende a ser modificado frecuentemente.

Si el **.emacs** de una persona contiene algún error, Emacs no intentará cargar **default.el**, sino que simplemente se detendrá, destellando un mensaje diciendo “**Error in init file.**” (Error en el archivo de inicio) o algo similar. Si ve este mensaje, probablemente algo vaya mal con su **.emacs**.

Hay un tipo más de expresión que a menudo va en un **.emacs**. La librería de Emacs Lisp a veces ofrece múltiples paquetes para hacer lo mismo de diferentes formas. Esto significa que tiene que especificar cuál quiere usar (o tendrá el paquete por defecto, que no es siempre el mejor para todos los propósitos). Un área donde esto sucede es en las características de interacción del Scheme de Emacs. Hay dos interfaces diferentes de Scheme distribuidos con Emacs (al menos en la versión 19): **xscheme** y **cmuscheme**.

```
prompt> ls /usr/lib/emacs/19.19/lisp/*scheme*
/usr/lib/emacs/19.19/lisp/cmuscheme.el
/usr/lib/emacs/19.19/lisp/cmuscheme.elc
/usr/lib/emacs/19.19/lisp/scheme.el
/usr/lib/emacs/19.19/lisp/scheme.elc
/usr/lib/emacs/19.19/lisp/xscheme.el
/usr/lib/emacs/19.19/lisp/xscheme.elc
```

Resulta que el interfaz ofrecido por **cmuscheme** me gusta mucho más que el que ofrece **xscheme**, pero el que Emacs usará por defecto es **xscheme**. ¿Cómo puedo hacer que Emacs actúe de acuerdo con mi preferencia?. Puse esto en mi **.emacs**:

```
;; note cómo la expresión puede quebrarse en dos líneas.  Lisp
;; ignora los espacios en blanco, generalmente:
(autoload 'run-scheme "cmuscheme"
"Corre un Scheme inferior, de la forma que me gusta." t)
```

La función **autoload** (auto carga) toma el nombre de una función (citada con “’”, por razones que tienen que ver con cómo funciona Lisp) y le dice a Emacs que esta función está definida en un determinado archivo. El archivo es el segundo argumento, una cadena (sin la extensión **.el** o **.elc**) indicando el nombre del archivo a buscar en la trayectoria de carga `load-path`.

Los argumentos restantes son opcionales, pero necesarios en este caso: el tercer argumento es una cadena de documentación para la función, de modo que si llama a **describe-function** (describir-

función), consigue alguna información útil. El cuarto argumento le dice a Emacs que esta función autocargable puede ser llamada interactivamente (esto es, usando `M-x`). Esto es muy importante en este caso, porque uno debería poder teclear `M-x run-scheme` para comenzar un proceso de scheme que se ejecuta bajo Emacs.

Ahora que `run-scheme` ha sido definido como una función autocargable, ¿qué sucede cuando tecleo `M-x run-scheme`? Emacs mira la función `run-scheme`, ve que está establecida para ser autocargable, y carga el archivo nombrado por la autocarga (en este caso, `cmuscheme`). El archivo compilado `cmuscheme.elc` existe, así que Emacs lo cargará. Ese archivo *debe* definir la función `run-scheme`, o habrá un error de autocarga. Por suerte, define `run-scheme`, así que todo va sin tropiezos, y consigo mi interfaz preferida de Scheme¹⁰.

Una **autocarga** es como una promesa a Emacs, de que cuando llegue el momento, puede encontrar la función especificada en el archivo en el que le dice que mire. A cambio, consigue algún control sobre lo que se carga. También, la autocarga ayuda a reducir el tamaño de Emacs en la memoria, al no cargar ciertas características hasta que se pidan. Muchos comandos no están definidos realmente como funciones cuando Emacs se inicia. Más bien, están simplemente preparados para autocargarse desde cierto archivo. Si nunca invoca el comando, nunca se carga. Este ahorro de espacio es vital para el funcionamiento de Emacs: si cargara todos los archivos disponibles en la biblioteca Lisp, Emacs tomaría veinte minutos simplemente para arrancar, y una vez hecho, podría ocupar la mayor parte de la memoria disponible en su máquina. No se preocupe, no tiene que establecer todas estas autocargas en su `.emacs`; ya se tomaron en cuenta cuando Emacs se desarrolló.

8.14 Averiguando más

No le he contado todo lo que se puede saber sobre Emacs. De hecho, no creo haberle contado siquiera un 1% de lo que se puede saber sobre Emacs. Aunque sabe suficiente para proseguir, todavía hay montones de comodidades y trucos que ahorran tiempo que debería averiguar. La mejor forma de hacerlo es esperar hasta que vea que necesita algo, y buscar entonces una función que lo haga.

La importancia de estar cómodo con las facilidades de ayuda en línea de Emacs no puede enfatizarse lo suficiente. Por ejemplo, suponga que quiere poder insertar los contenidos de algún archivo en un buffer que ya está trabajando sobre un archivo diferente, para que el buffer contenga a ambos. Si intuyese que hay un comando llamado `insert-file` (insertar archivo), tendría razón. Para verificar su acertada suposición, teclee `C-h f`. En el prompt del minibuffer, introduzca el nombre de una función sobre la que quiera ayuda. Puesto que sabe que hay terminación automática en los nombres de funciones, y puede suponer que el comando que busca comienza con “insert”, escriba `insert` y presiona `Tab`. Esto le muestra todos los nombres de función que comienzan con “insert”, e “insert-file” es uno de ellos.

De este modo completa el nombre de función y lee sobre como trabaja, y entonces usa `M-x insert-file`. Si se está preguntando si también está ligado a una tecla, escribe `C-h w insert-file` `Intro`, y averígüelo. Cuanto más sepa de las facilidades de ayuda de Emacs, más fácilmente podrá

¹⁰A propósito, `cmuscheme` era la interfaz de la que hablaba antes, en la sección sobre el trabajo con Scheme, así que si quiere usar algo de este manual, necesita asegurarse de que ejecuta `cmuscheme`.

hacer preguntas a Emacs acerca de sí mismo. La capacidad de hacerlo, combinada con un espíritu de exploración y un deseo de aprender nuevas formas de hacer las cosas, puede acabar por ahorrarle mucho tecleado.

Para pedir una copia del manual de usuario de Emacs (Emacs user's manual) y/o el manual de Programación en Emacs Lisp (Emacs Lisp Programming manual), debe escribir a:

Free Software Foundation
675 Mass Ave
Cambridge, MA 02139
USA

Ambos manuales se distribuyen electrónicamente con Emacs, en una forma legible usando el lector de documentación Info (**C-h i**), pero puede encontrar más fácil tratar con freeware que con las versiones en línea. Además, sus precios son bastantes razonables, y el dinero va a una buena causa—¡software gratuito de calidad!. En algún momento, debería teclear **C-h C-c** para leer las condiciones de copyright de Emacs. Es más interesante de lo que puede pensar, y le ayudará a aclarar el concepto de software libre. Si cree que el termino “free software” simplemente significa que el programa no cuesta nada, por favor ¡lea el copyright en cuanto tenga tiempo!.

Capítulo 9

¡Tengo que ser yo mismo!

Si Dios hubiera sabido que necesitaríamos previsión, nos la habría dado.

9.1 Personalización del bash

Una de las cosas que distinguen la filosofía de Unix es que los diseñadores de sistemas no intentaron predecir cada necesidad que los usuarios podrían tener; en lugar de eso, intentaron hacer fácil para cada usuario individual la configuración del entorno a sus necesidades particulares. Esto se consigue principalmente a través de **ficheros de configuración**. También son conocidos como “ficheros de inicio”, “ficheros rc” (por “run control”, control de arranque), o incluso “ficheros punto (dot files)”, porque los nombres de los ficheros siempre empiezan con “.”. Recordemos que los ficheros que empiezan por “.” no se visualizan normalmente con `ls`.

Los ficheros de configuración más importantes son los usados por el shell. El shell por defecto de Linux es el **bash**, y éste es el shell que cubre este capítulo. Antes de empezar a explicar cómo personalizar el **bash**, tenemos que saber cuales son los archivos que mira.

9.1.1 Arranque del shell

Hay diferentes modos de funcionamiento del **bash**. Puede funcionar como **shell de ingreso**, que es el modo en que arranca cuando se ingresa por primera vez. El shell de ingreso debería ser el primer shell que vea.

Otro modo en que puede funcionar **bash** es como **shell interactivo**. Este es un shell que presenta un prompt a un humano y espera una entrada de datos. Un shell de ingreso también es un shell interactivo. Una manera de conseguir un shell interactivo sin ingresar en el sistema es, por ejemplo, un shell dentro de **xterm**. Cualquier shell que sea creado por otro medio distinto del ingreso registrado en el sistema es un shell de no-ingreso.

Finalmente, hay **shells no interactivos**. Estos shells se usan para ejecutar un archivo de comandos, muy parecidos a los ficheros de procesamiento por lotes del MS-DOS —los archivos que

acaban en **.BAT**. Estas **macros de shell** funcionan como mini-programas. Aunque son usualmente mucho más lentos que un programa normal compilado, suele ser cierto también que son mucho más fáciles de escribir. Dependiendo del tipo de shell, se usarán distintos tipos de archivo al arrancarlo:

Tipo de Shell	Acción
Ingreso interactivo	Se lee y ejecuta el archivo .bash_profile .
Interactivo	Se lee y ejecuta el archivo .bashrc .
No interactivo	Se lee y ejecuta la macro de shell.

9.1.2 Ficheros de arranque

Como muchos usuarios quieren tener mayormente el mismo entorno, sin importar que tipo de shell interactivo acaben teniendo, y sea o no un shell de ingreso, empezaremos nuestra configuración poniendo un comando muy simple en nuestro archivo **.bash_profile**: “**source ~/.bashrc**”. El comando **source** ordena al shell que interprete el argumento como una macro de shell. Lo que significa para nosotros es que cada vez que **.bash_profile** se ejecuta, *también* se ejecuta **.bashrc**.

Ahora, sólo añadiremos comandos a nuestro archivo **.bashrc**. Si alguna vez queremos que se ejecute un comando únicamente cuando hemos hecho un ingreso registrado, lo añadiremos a nuestro **.bash_profile**.

9.1.3 Creando alias

¿Cuales son algunas de las cosas que interesaría personalizar? Esto es algo que creo que el 90% de los usuarios de Bash han puesto en su **.bashrc**:

```
alias ll="ls -l"
```

Este comando define un **alias** de shell llamado **ll** que “expande” el comando normal de shell “**ls -l**” cuando se invoca por el usuario. De modo que, asumiendo que Bash ha leído este comando del fichero **.bashrc**, podemos teclear **ll** para conseguir el efecto de “**ls -l**” con solo la mitad de pulsaciones. Lo que ocurre es que cuando tecleamos “**ll**” y pulsamos Intro, Bash lo intercepta, lo reemplaza por “**ls -l**”, y ejecuta éste en su lugar. No hay ningún programa llamado **ll** en el sistema, pero el shell automáticamente traduce el alias a un programa válido.

Hay algunos alias de ejemplo en la figura 9.1.3. Puede ponerlos en su propio **.bashrc**. Uno especialmente interesante es el primero. Con él, cada vez que alguien teclea “**ls**”, automáticamente tiene una opción “**-F**” añadida. (El alias no intenta expandirse a sí mismo otra vez). Este es un modo muy común de añadir opciones que se usan constantemente al llamar a un programa.

Nótese los comentarios con el caracter “**#**” en la figura 9.1.3. Cada vez que aparece un “**#**”, el shell ignora el resto de la línea.

Quizá haya notado unas cuantas cosas sueltas sobre los ejemplos. Primero, me he dejado las comillas en algunos de los alias—como **pu**. Estrictamente hablando, las comillas no son necesarias cuando sólo se tiene una palabra a la derecha del signo igual.

Figura 9.1 Algunos alias de ejemplo para bash.

```
alias ls="ls -F"           # muestra los caracteres al final del listado
alias ll="ls -l"           # ls especial
alias la="ls -a"
alias ro="rm *~; rm .*~"   # este borra las copias de seguridad creadas por Emacs
alias rd="rmdir"           # ahorra teclas!
alias md="mkdir"
alias pu=pushd             # pushd, popd, y dirs no estan incluidos en este
alias po=popd             # manual---quizá quiera echarles un vistazo
alias ds=dirs              # en la página man de Bash
# estos solo son atajos de teclado
alias to="telnet cs.oberlin.edu"
alias ta="telnet altair.mcs.anl.gov"
alias tg="telnet wombat.gnu.ai.mit.edu"
alias tko="tpalk kold@cs.oberlin.edu"
alias tjo="talk jimb@cs.oberlin.edu"
alias mroe="more"         # corrección ortográfica!
alias moer="more"
alias email="emacs -f rmail" # mi lector de correo
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\" \"\" # así llamo a Emacs
```

Pero no hace daño poner comillas, así que no me dejéis crearos malos hábitos. Ciertamente habrá que usarlas si se va a crear un alias de un comando con opciones y/o argumentos:

```
alias rf="refrobnicate -verbose -prolix -wordy -o foo.out"
```

Además, el último alias tiene algún entrecomillado gracioso:

```
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\" \"\""
```

Como es fácil suponer, he querido pasar entrecomillados dentro de las opciones, así que tengo que anteponerles una contrabarra para que `bash` no crea que ya ha llegado al final del alias.

Finalmente, he creado dos alias de dos errores de escritura comunes, “mroe” y “moer”, apuntando al comando que pretendía escribir, `more`. Los alias no interfieren con el paso de argumentos a un programa. Lo siguiente funciona bien:

```
/home/larry$ mroe hurd.txt
```

De hecho, saber como crear sus propios alias es probablemente por lo menos la mitad de las personalizaciones que hará jamás. Experimente un poco, descubra cuáles son los comandos largos que teclea frecuentemente, y cree alias para ellos. De esta manera, le resultará más confortable trabajar con el prompt del shell.

9.1.4 Variables de entorno

Otra cosa que uno hace frecuentemente en `.bashrc` es definir **variables de entorno**. Y ¿qué son las variables de entorno? Vamos a mirarlo desde otra dirección: Supongamos que está leyendo la

documentación del programa **fruggle**, y se encuentra con una de estas expresiones:

Fruggle normalmente busca su fichero de configuración, **.fruggerc**, en el directorio raíz del usuario. Sin embargo si la variable de entorno **FRUGGLEPATH** indica un nombre de archivo diferente, mirará ahí en su lugar.

Cada programa se ejecuta en un **entorno**, y ese entorno es definido por el shell que llamó al programa¹. Se puede decir que el entorno existe “dentro” del shell. Los programadores tienen una rutina especial para interrogar al entorno, y el programa **fruggle** usa esa rutina. Comprueba el valor de la variable de entorno **FRUGGLEPATH**. Si esa variable no está definida, sencillamente usará el archivo **.fruggerc** del directorio raíz. Si está definida, usará el valor de esa variable (que debe ser el nombre de un archivo que **fruggle** pueda usar) en lugar del archivo por defecto **.fruggerc**.

Así es como se puede cambiar el entorno en **bash**:

```
/home/larry$ export PGPPATH=/home/larry/secrets/pgp
```

Hay que pensar en el comando **export** con este significado: “Por favor exporta esta variable fuera del entorno donde estaré llamando programas, de modo que su valor sea visible para ellos.” Realmente hay razones para llamarlo **export**, como se verá después.

Esta variable en particular es usada por el infame programa de encriptación mediante clave pública de Phil Zimmerman, **pgp**. Por defecto, **pgp** usa el directorio raíz como lugar para encontrar determinados archivos que necesita (que contienen claves de encriptación), y también un lugar para guardar archivos temporales que crea mientras está en marcha. Al darle este valor a la variable **PGPPATH**, le he dicho que use el directorio **/home/larry/secrets/pgp** en lugar de **/home/larry**. He tenido que leer el manual de **pgp** para encontrar el nombre exacto de la variable y lo que hace, pero es bastante estándar el uso del nombre del programa en mayúsculas, seguido del sufijo “PATH”.

También es útil saber como preguntar al entorno:

```
/home/larry$ echo $PGPPATH
/home/larry/.pgp
/home/larry$
```

Nótese el “\$”; se precede la variable de entorno con un signo de dólar para extraer el valor de la variable. Si lo hubiera escrito sin el signo de dólar, **echo** simplemente habría mostrado su(s) argumento(s):

```
/home/larry$ echo PGPPATH
PGPPATH
/home/larry$
```

El “\$” se usa para *evaluar* variables de entorno, pero sólo lo hace dentro del contexto del shell—o sea, cuando el shell está interpretando. ¿Cuándo está el shell interpretando? Bueno, cuando se

¹ Ahora se puede comprobar porqué los shells son tan importantes. ¡No hay que andar pasando un entorno completo a mano cada vez que se llame a un programa!

Figura 9.2 Algunas variables de entorno importantes.

Nombre	Contenido	Ejemplo
HOME	Directorio principal del usuario	<code>/home/larry</code>
TERM	Tipo de terminal del usuario	<code>xterm</code> , <code>vt100</code> , o <code>console</code>
SHELL	Path del shell del usuario	<code>/bin/bash</code>
USER	Nombre de ingreso	<code>larry</code>
PATH	Lista de búsqueda de programas	<code>/bin:/usr/bin:/usr/local/bin:/usr/bin/X11</code>

escriben comandos en el prompt, o cuando `bash` está leyendo comandos de un archivo como `.bashrc`, se puede decir que está “interpretando” los comandos.

Hay otro comando muy útil para preguntar al entorno: `env`. `env` enseñará un listado de todas las variables de entorno. Es posible, especialmente si se usa X, que la lista se salga de la pantalla. Si esto ocurre, hay que canalizar `env` a través de `more`: “`env | more`”.

Unas cuantas de estas variables pueden ser muy útiles, así que las comentaremos. Mire la Figura 9.1.4. Estas cuatro variables están definidas automáticamente cuando se ingresa en el sistema: no se definen en `.bashrc` o `.bash_login`.

Vamos a echar un vistazo más de cerca a la variable `TERM`. Para comprenderla, vamos a mirar hacia atrás en la historia del Unix: El sistema operativo necesita conocer ciertos datos sobre su consola para poder realizar funciones básicas, como escribir un carácter en la pantalla, mover el cursor a la línea siguiente, etc. En los primeros días de los ordenadores, los fabricantes estaban continuamente añadiendo nuevas características a sus terminales: primero el vídeo inverso, luego quizás juegos de caracteres europeos, eventualmente incluso primitivas funciones de dibujo (hay que recordar que éstos eran los tiempos anteriores a los sistemas de ventanas y el ratón). Pero todas estas funciones representaban un problema para los programadores: ¿Cómo iban a saber lo que un terminal podía soportar y lo que no? Y ¿cómo podían emplear nuevas características sin convertir los viejos terminales en inservibles?

En Unix, la respuesta a estas cuestiones fue `/etc/termcap`. `/etc/termcap` es una lista de todos los terminales que un sistema conoce, y como controlan el cursor. Si un administrador de sistema consigue un terminal nuevo, todo lo que tiene que hacer es añadir una entrada para ese terminal en `/etc/termcap` en lugar de recompilar todo el Unix. A veces es incluso más simple. Al pasar el tiempo, el `vt100` de Digital Equipment Corporation se convirtió en un pseudo-estándar, y muchos nuevos terminales fueron construidos para que pudieran emularlo, o comportarse como si fueran un `vt100`.

Bajo Linux, el valor de `TERM` es a veces `console`, un clónico de `vt-100` con algunas características añadidas.

Otra variable, `PATH`, es también crucial para el funcionamiento correcto del shell. Aquí está la mía:

```
/home/larry$ env | grep ^PATH
PATH=/home/larry/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/TeX/bin
/home/larry$
```

El PATH es una lista, separada por el carácter dos puntos “:”, de los directorios donde el shell buscará el nombre del programa a ejecutar. Cuando yo tecleo “ls” y pulso Intro, por ejemplo, Bash primero busca en `/home/larry/bin`, un directorio que he hecho para guardar los programas que escribo. Pero yo no he escrito `ls` (de hecho, ¡creo que se escribió antes de que yo naciera!). Como no lo encuentra en `/home/larry/bin`, Bash mira después en `/bin`—¡y ahí hay una coincidencia! `/bin/ls` existe y es ejecutable, de modo que Bash deja de buscar un programa llamado `ls` y lo arranca. Podría haber habido perfectamente otro `ls` esperando en el directorio `/usr/bin`, pero `bash` nunca lo ejecutará si no lo pido especificando una ruta de directorios explícita:

```
/home/larry$ /usr/bin/ls
```

La variable PATH existe para no tener que teclear rutas de directorio completas para cada comando. Cuando se escribe un comando, Bash lo busca en los directorios nombrados en el PATH, en orden, y si lo encuentra, lo ejecuta. Si no lo encuentra, devuelve un descortés mensaje de error:

```
/home/larry$ clubly
clubly: command not found
```

Notar que en mi PATH no existe el directorio actual, “.”. Si estuviera, tendría este aspecto:

```
/home/larry$ echo $PATH
./home/larry/bin:/bin:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/TeX/bin
/home/larry$
```

Esto es asunto de debate en los círculos de Unix (de los cuales ahora es miembro, le guste o no). El problema es que tener el directorio actual en el path puede ser un agujero en la seguridad. Supongamos que entramos en un directorio en el que alguien ha dejado un “Caballo de Troya” llamado `ls`, y hacemos un `ls`, como es natural después de entrar en un directorio nuevo. Como el directorio actual, “.”, viene primero en nuestro PATH, el shell encontrará esta versión de `ls` y la ejecutará. Sea cual sea el daño que hayan puesto en ese programa, lo acabamos de activar (y puede ser un montón de daño). Quien fuera no necesita permisos de root para hacerlo; sólo hace falta permisos de escritura en el directorio donde se encuentra el “falso” `ls`. Incluso podría ser su propio directorio home, si sabe que vamos a estar husmeando por ahí en algún momento.

En su propio sistema, es muy improbable que las personas se estén dejando trampas unas a otras. Pero en un gran sistema multiusuario (como muchos ordenadores de universidades), hay un montón de programadores hostiles con los que nunca se ha encontrado. Que quiera tentar a la suerte o no teniendo “.” en el path depende de la situación; no voy a ser dogmático en ningún sentido, sólo quiero informar de los riesgos implícitos². Los sistemas multiusuario son verdaderas comunidades, donde la gente puede hacerles cosas a los demás en todo tipo de maneras nunca vistas.

El modo en que he dejado mi PATH incluye la mayoría de lo que se ha aprendido hasta ahora sobre variables de entorno. Esto es lo que hay actualmente en mi **.bashrc**:

```
export PATH=${PATH}:/usr/bin:/usr/local/bin:/usr/bin/X11:/usr/TeX/bin
```

²Recuérdese que siempre se puede ejecutar un programa en el directorio actual siendo explícito, p.ej.: “./foo”.

Me aprovecho de que la variable `HOME` se activa antes de que Bash lea el fichero `.bashrc`, usando su valor en la construcción del `PATH`. Las llaves (“{...}”) son un nivel añadido de cita; delimitan el tamaño de lo que va a evaluar “\$”, de modo que el shell no se confunda por culpa del texto inmediatamente posterior (“/bin” en este caso). Aquí hay otro ejemplo del efecto que tienen:

```
/home/larry$ echo ${HOME}foo
/home/larryfoo
/home/larry$
```

Sin las llaves, no conseguiría nada, porque no hay ninguna variable de entorno llamada `HOMEfoo`.

```
/home/larry$ echo $HOMEfoo

/home/larry$
```

Quisiera aclarar una cosa en este path: el significado de “\$PATH”. Lo que hace es incluir el valor de cualquier variable `PATH` *previamente* activada en mi nuevo `PATH`. ¿Dónde se habrá activado la variable anterior? El archivo `/etc/profile` sirve como una especie de `.bash_profile` global, común a todos los usuarios. Tener un archivo centralizado como este hace más sencillo para el administrador del sistema añadir un directorio nuevo al `PATH` de todo el mundo, sin que cada uno tenga que hacerlo individualmente. Si se incluye el path antiguo en el nuevo, no se perderán ninguno de los directorios que el sistema ya haya preparado.

También se puede controlar como aparece el prompt. Esto se consigue mediante la variable de entorno `PS1`. Personalmente, quiero un prompt que me indique el path del directorio actual—así es “como lo hago en mi `.bashrc`”:

```
export PS1=' $PWD\ $ '
```

Como se puede ver, en realidad se usan *dos* variables. La que se activa es `PS1`, y toma el valor de `PWD`, que puede ser interpretada como “Print Working Directory” (imprime el directorio de trabajo) o “Path to Working Directory” (trayectoria al directorio de trabajo). Pero la evaluación de `PWD` tiene lugar entre apóstrofes agudos. Estos apóstrofes sirven para evaluar la expresión en su interior, la cual a su vez evalúa la variable `PWD`. Si sólo hubieramos hecho “`export PS1=$PWD`”, nuestro prompt nos habría mostrado constantemente el path del directorio de trabajo *en el momento en que PS1 fué activada*, en lugar de actualizarse constantemente mientras cambiamos de directorios. Bueno, es un poco confuso, y no muy importante en realidad. Sólo hay que tener en cuenta que se necesita esa clase de apóstrofes si se quiere mostrar el directorio actual en el prompt.

Quizá se prefiera “`export PS1=' $PWD> '`”, o incluso el nombre del sistema:

```
export PS1='hostname'>'
```

Disecionemos este ejemplo un poco más. Aquí se usa un *nuevo* tipo de literal, el apóstrofo grave. Esta clase de literal no protege nada —de hecho, ‘hostname’ no aparece en ninguna parte del prompt cuando intentamos arrancarlo. Lo que sucede realmente es que se evalúa el comando dentro de los apóstrofes graves, y el resultado se guarda en lugar del nombre del comando entrecomillado.

Probemos con “echo ‘ls’” o “wc ‘ls’”. A medida que se consiga más experiencia usando el shell, esta técnica se hace más y más potente.

Hay mucho más por comentar de como se configura el fichero **.bashrc**, y aquí no hay bastante espacio para hacerlo. Se puede aprender más leyendo la página **man** de **bash**, o preguntando a usuarios experimentados. Aquí hay un **.bashrc** completo para poder estudiarlo; es razonablemente estándar, aunque el path es un poco largo.

```
# Algunas cosas al azar:
ulimit -c unlimited
export history_control=ignoredups
export PS1='$PWD>'
umask 022

# paths específicos de aplicaciones:
export MANPATH=/usr/local/man:/usr/man
export INFOPATH=/usr/local/info
export PGPPATH=${HOME}/.pgp

# PATH principal:
homepath=${HOME}:/bin
stdpath=/bin:/usr/bin:/usr/local/bin:/usr/ucb:/etc:/usr/etc:/usr/games
pubpath=/usr/public/bin:/usr/gnuft/bin:/usr/local/contribs/bin
softpath=/usr/bin/X11:/usr/local/bin/X11:/usr/TeX/bin
export PATH=.:${homepath}:${stdpath}:${pubpath}:${softpath}
# Técnicamente, las llaves no eran necesarias, porque los dos puntos son
# delimitadores válidos; pero las llaves son una buena costumbre,
# y no hacen daño.

# alias
alias ls="ls -CF"
alias fg1="fg %1"
alias fg2="fg %2"
alias tba="talk sussman@tern.mcs.anl.gov"
alias tko="talk kold@cs.oberlin.edu"
alias tji="talk jimb@totoro.bio.indiana.edu"
alias mroe="more"
alias moer="more"
alias email="emacs -f vm"
alias pu=pushd
alias po=popd
alias b=~/.b"
alias ds=dirs
alias ro="rm *~; rm .*~"
alias rd="rmdir"
alias ll="ls -l"
alias la="ls -a"
alias rr="rm -r"
```

```
alias md="mkdir"
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\"

function gco
{
    gcc -o $1 $1.c -g
}
```

9.2 Los ficheros de inicio de X Window



Mucha gente prefiere trabajar dentro de un entorno gráfico, y para las máquinas bajo Unix eso suele significar usar X. Si se está acostumbrado al Macintosh o al Microsoft Windows, puede costar un poco acostumbrarse al sistema X Window, especialmente al modo en que se configura.

Con Macintosh o Microsoft Windows, se personaliza el entorno *desde dentro*: si se quiere cambiar el fondo, por ejemplo, se pulsa con el ratón sobre el nuevo color en algún programa gráfico especial de configuración. En X, las opciones por defecto del sistema se controlan mediante ficheros de texto, que se editan directamente—en otras palabras, se escribe el nombre del color en un fichero para cambiar el color del fondo.

No se puede negar que este método no es tan llano como el de algunos sistemas de ventanas comerciales. Creo que esta tendencia a permanecer basado en texto, incluso en un entorno gráfico, tiene mucho que ver con el hecho de que X fué creado por un puñado de programadores que simplemente no estaban tratando de escribir un software que pudieran usar sus abuelos. Esta tendencia puede cambiar en futuras versiones de X (por lo menos así lo espero³), pero por ahora, hay que aprender como entenderse con unos cuantos ficheros de texto más. Por lo menos nos da un control muy flexible y preciso de nuestra configuración.

Aquí están los ficheros más importantes para configurar X:

- .xinitrc** Un script arrancado por X cuando se inicia.
- .twmrc** Leído por el gestor de X Window, **twm**.
- .fvwmrc** Leído por el gestor de X Window, **fvwm**.

Todos estos ficheros deberían estar en el directorio **\$HOME**, si es que existen.

.xinitrc es una macro de shell muy simple que se ejecuta cuando invocamos a X. Puede hacer cualquiera de las cosas que pueden hacer otras macros de shell, pero por supuesto su uso principal es arrancar varios programas de X y establecer parámetros del sistema de ventanas. El último comando en **.xinitrc** es usualmente el nombre de un gestor de ventanas a ejecutar, por ejemplo **/usr/bin/X11/twm**.

¿Qué tipo de cosas nos gustaría poner en **.xinitrc**? Quizá algunas llamadas al programa **xsetroot**, para hacer que la ventana de fondo (escritorio) y el ratón tengan el aspecto que deseamos. Llamadas a **xmodmap**, que informa al servidor⁴ de como interpretar las señales del teclado.

³N. del T.: Yo también ;-)

⁴El “servidor” sólo se refiere al proceso principal en la máquina, con el que todos los demás programas de X tienen que comunicar para usar la pantalla. Estos otros programas se conocen como “clientes”, y todo el paquete se llama

O cualquier otro programa que se quiera iniciar cada vez que se arranque X (por ejemplo, `xclock`).

Aquí está una parte de mi `.xinitrc`; el suyo seguramente será distinto, así que hay que tomárselo sólo como ejemplo:

```
#!/bin/sh
# La primera línea le dice al sistema operativo que shell usar para
# interpretar esta macro. La macro en sí misma debe estar marcada como
# ejecutable; se puede hacer con "chmod +x ~/.xinitrc".

# xmodmap es un programa que informa al servidor X de como interpretar las
# señales del teclado. Es *definitivamente* aconsejable aprender como
# funciona. Se puede probar con "man xmodmap", "xmodmap -help",
# "xmodmap -grammar", y otras opciones.
# No garantizo que las expresiones abajo escritas signifiquen nada en tu
# sistema (ni siquiera garantizo que signifiquen algo en el mío):
xmodmap -e 'clear Lock'
xmodmap -e 'keycode 176 = Control_R'
xmodmap -e 'add control = Control_R'
xmodmap -e 'clear Mod2'
xmodmap -e 'add Mod1 = Alt_L Alt_R'

# xset es un programa para establecer otros parámetros del servidor X:
xset m 3 2 &          # parámetros del ratón
xset s 600 5 &         # preferencias del salvapantallas
xset s noblank &       # lo mismo
xset fp+ /home/larry/x/fonts # para cterm
# Para aprender más, "xset -help".

# Ordena al servidor X que sobreponga fish.cursor sobre fish.mask, y use
# el patrón resultante como cursor del ratón:
xsetroot -cursor /home/lab/larry/x/fish.cursor /home/lab/larry/x/fish.mask &

# Un placentero patrón y color de fondo:
xsetroot -bitmap /home/lab/larry/x/pyramid.xbm -bg tan

# para hacer: xrdb aquí? Y que hay del fichero .Xdefaults?

# Hay que mirar "man xsetroot", o "xsetroot -help" para más
# información sobre el programa usado arriba.

# Un programa cliente, el imponente reloj circular de color de Jim Blandy:
/usr/local/bin/circles &

# Quizá os gustaría tener un reloj en la pantalla a todas horas?
/usr/bin/X11/xclock -digital &
```

“sistema cliente-servidor”.


```

# Permite a programas cliente de X ejecutándose en occs.cs.oberlin.edu
# mostrarse a sí mismos aquí, y lo mismo para juju.mcs.anl.gov:
xhost occs.cs.oberlin.edu
xhost juju.mcs.anl.gov

# Se puede simplemente decirle al servidor X que permita usar aquí
# a clientes que estén ejecutándose en cualquier otra máquina,
# pero esto es un agujero en la seguridad -- <estos clientes pueden ser
# usados por cualquier otro, que puede vigilar lo que se teclea cuando se
# entra una contraseña, o algo así!
# Si se quiere hacerlo de todos modos, se puede usar un "+" para indicar
# cualquier nombre de host posible, en lugar de un nombre específico,
# así:
# xhost +

# Y finalmente arrancar el gestor de ventanas:
/usr/bin/X11/twm

# Alguna gente prefiere otros gestores de ventanas. Yo uso twm, pero fvwm se
# distribuye también muchas veces con Linux:
# /usr/bin/X11/fvwm

```

Nótese que algunos comandos son iniciados en segundo plano (p.ej.: los que van seguidos de “&”), mientras que otros no lo son. La distinción está en que algunos programas se iniciarán cuando arranquemos X y seguirán en marcha hasta que salgamos. Los otros se ejecutan una vez y se cierran inmediatamente. `xsetroot` es uno de éstos; sólo establece la ventana principal o el cursor a lo que sea, y luego se cierra.

Una vez que el gestor de ventanas ha arrancado, leerá su propio fichero de inicio, que controla cosas como la composición de los menús, en qué posiciones se presentan las ventanas cuando se abren, control de iconos, y otros asuntos terriblemente importantes. Si se usa `twm`, el fichero es `.twmrc` en nuestro directorio home. Si se usa `fvwm`, el fichero es `.fvwmrc`, etc. Explicaremos sólo estos dos, ya que son los gestores de ventanas que más frecuentemente se encontrarán con Linux.

9.2.1 Configuración de twm

`.twmrc` no es una macro de shell—está escrito en un lenguaje especialmente hecho para `twm`, ¡créalo o no!⁵ Principalmente, a la gente en su `.twmrc` le gusta jugar con los estilos de la ventana (colores y demás), y hacer interesantes menús, así que ahí va un `.twmrc` de ejemplo:

```

# Indica colores para varias partes de una ventanas. Tiene un gran impacto
# en la "sensación" que transmite el entorno.
Color
{

```

⁵Este es uno de los problemas de los ficheros de configuración: normalmente usan su propio lenguaje de comandos. Esto significa que los usuarios se vuelven muy buenos a la hora de aprender lenguajes nuevos rápidamente. Supongo que habría estado bien que los primeros programadores de Unix se hubieran puesto de acuerdo en algún estándar de fichero de inicio, pero para ser justo es difícil de predecir que tipo de informaciones necesitarán los programas.

```

    BorderColor "OrangeRed"
    BorderTileForeground "Black"
    BorderTileBackground "Black"
    TitleForeground "black"
    TitleBackground "gold"
    MenuForeground "black"
    MenuBackground "LightGrey"
    MenuTitleForeground "LightGrey"
    MenuTitleBackground "LightSlateGrey"
    MenuShadowColor "black"
    IconForeground "DimGray"
    IconBackground "Gold"
    IconBorderColor "OrangeRed"
    IconManagerForeground "black"
    IconManagerBackground "honeydew"
}

# Espero que no tengas un sistema monocromo, pero si lo tienes...
Monochrome
{
    BorderColor "black"
    BorderTileForeground "black"
    BorderTileBackground "white"
    TitleForeground "black"
    TitleBackground "white"
}

# He creado beifang.bmp con el programa "bitmap". Aquí le digo a twm
# que lo use como el patrón por defecto para destacar las barras de
# título de las ventanas:
Pixmaps
{
    TitleHighlight "/home/larry/x/beifang.bmp"
}

# No os preocupéis por estas cosas, son sólo para usuarios avanzados :-)
BorderWidth      2
TitleFont         "-adobe-new century schoolbook-bold-r-normal--14-140-75-75-p-87-iso8859-1"
MenuFont          "6x13"
IconFont          "lucidasans-italic-14"
ResizeFont        "fixed"
Zoom 50
RandomPlacement

# Estos programas no tendrán una barra de título por defecto
NoTitle
{
    "stamp"

```

```

"xload"
"xclock"
"xlogo"
"xbiff"
"xeyes"
"oclock"
"xoid"
}

# "AutoRaise" significa que una ventana aparece al frente cuando el ratón
# entra en ella. Lo encuentro desconcertante, así que lo he desactivado.
# Como puedes ver, he heredado mi .twmrc de otra gente a quien tampoco le
# gustaba Autoraise.
AutoRaise
{
    "nothing"      # No me gusta auto-raise  # Ni a mi  # A mi tampoco
}

# Aquí es donde se define la función de los botones del mouse.
# Nótese el patrón: Un botón del ratón pulsado en la ventana
# principal, si no se está pulsando ninguna tecla modificadora, siempre
# hace aparecer un menú. Otras posiciones usualmente producen
# modificaciones de la ventana en algún modo, y las teclas modificadoras
# se usan junto con los botones del ratón para acceder a las funciones
# más avanzadas de manipulación de ventanas.
#
# No hay porqué seguir este patrón en .twmrc -- el cómo
# establecer el entorno es un asunto totalmente personal

# Button = KEYS : CONTEXT : FUNCTION
# -----
Button1 =      : root      : f.menu "main"
Button1 =      : title     : f.raise
Button1 =      : frame     : f.raise
Button1 =      : icon      : f.iconify
Button1 = m    : window    : f.iconify

Button2 =      : root      : f.menu "stuff"
Button2 =      : icon      : f.move
Button2 = m    : window    : f.move
Button2 =      : title     : f.move
Button2 =      : frame     : f.move
Button2 = s    : frame     : f.zoom
Button2 = s    : window    : f.zoom

Button3 =      : root      : f.menu "x"
Button3 =      : title     : f.lower
Button3 =      : frame     : f.lower

```

```

Button3 =      : icon      : f.raiselower

# Puedes escribir tus propias funciones; esta se usa en el menú
# "windowops" cerca del final de este fichero:
Function "raise-n-focus"
{
    f.raise
    f.focus
}

# Muy bien, abajo están los menús a los que me refería en la
# sección de los botones del ratón. Nótese que muchas de estas
# entradas de menú llaman a submenús a su vez. Se pueden tener tantos
# niveles como se quiera, pero no se pueden tener menús recursivos. Ya
# lo he probado.

menu "main"
{
    "Vanilla"      f.title
    "Emacs"        f.menu "emacs"
    "Logins"       f.menu "logins"
    "Xlock"        f.menu "xlock"
    "Misc"         f.menu "misc"
}

# Esto me permite llamar a emacs desde varias máquinas diferentes. Ver
# la sección sobre ficheros .rhosts para más información sobre
# como funciona:
menu "emacs"
{
    "Emacs"        f.title
    "here"         !"/usr/bin/emacs &"
    ""            f.nop
    "phylo"        !"rsh phylo \"emacs -d floss:0\" &"
    "geta"         !"rsh geta \"emacs -d floss:0\" &"
    "darwin"       !"rsh darwin \"emacs -d floss:0\" &"
    "ninja"        !"rsh ninja \"emacs -d floss:0\" &"
    "indy"         !"rsh indy \"emacs -d floss:0\" &"
    "oberlin"      !"rsh cs.oberlin.edu \"emacs -d floss.life.uiuc.edu:0\" &"
    "gnu"          !"rsh gate-1.gnu.ai.mit.edu \"emacs -d floss.life.uiuc.edu:0\" &"
}

# Esto me permite llamar a xterms desde varias máquinas diferentes. Ver
# la sección sobre ficheros .rhosts para más información sobre
# como funciona:
menu "logins"
{
    "Logins"       f.title

```

```

"here"      !"/usr/bin/X11/xterm -ls -T 'hostname' -n 'hostname' &"
"phylo"     !"rsh phylo \"xterm -ls -display floss:0 -T phylo\" &"
"geta"      !"rsh geta \"xterm -ls -display floss:0 -T geta\" &"
"darwin"    !"rsh darwin \"xterm -ls -display floss:0 -T darwin\" &"
"ninja"     !"rsh ninja \"xterm -ls -display floss:0 -T ninja\" &"
"indy"      !"rsh indy \"xterm -ls -display floss:0 -T indy\" &"
}

```

```

# El salvapantallas xlock, llamado con varias opciones (cada una de
# ellas da una bonita imagen):

```

```

menu "xlock"
{
"Hop"      !"xlock -mode hop &"
"Qix"      !"xlock -mode qix &"
"Flame"    !"xlock -mode flame &"
"Worm"     !"xlock -mode worm &"
"Swarm"    !"xlock -mode swarm &"
"Hop NL"   !"xlock -mode hop -nolock &"
"Qix NL"   !"xlock -mode qix -nolock &"
"Flame NL" !"xlock -mode flame -nolock &"
"Worm NL"  !"xlock -mode worm -nolock &"
"Swarm NL" !"xlock -mode swarm -nolock &"
}

```

```

# Programas de todo tipo que uso ocasionalmente:

```

```

menu "misc"
{
"Xload"      !"/usr/bin/X11/xload &"
"XV"         !"/usr/bin/X11/xv &"
"Bitmap"     !"/usr/bin/X11/bitmap &"
"Tetris"     !"/usr/bin/X11/xtetris &"
"Hextris"    !"/usr/bin/X11/xhextris &"
"XRoach"     !"/usr/bin/X11/xroach &"
"Analog Clock" !"/usr/bin/X11/xclock -analog &"
"Digital Clock" !"/usr/bin/X11/xclock -digital &"
}

```

```

# Esto es para lo que uso el botón central del ratón:

```

```

menu "stuff"
{
"Chores"     f.title
"Sync"       !"/bin/sync"
"Who"        !"who | xmessage -file - -columns 80 -lines 24 &"
"Xhost +"    !"/usr/bin/X11/xhost + &"
"Rootclear"  !"/home/larry/bin/rootclear &"
}

```

```

# Funciones de X que a veces conviene usar:

```

```

menu "x"
{
  "X Stuff"           f.title
  "Xhost +"           !"xhost + &"
  "Refresh"           f.refresh
  "Source .twmrc"      f.twmrc
  "(De)Iconify"        f.iconify
  "Move Window"        f.move
  "Resize Window"      f.resize
  "Destroy Window"     f.destroy
  "Window Ops"         f.menu "windowops"
  ""                  f.nop
  "Kill twm"           f.quit
}

# Este es un submenú del de arriba:
menu "windowops"
{
  "Window Ops"         f.title
  "Show Icon Mgr"       f.showiconmgr
  "Hide Icon Mgr"       f.hideiconmgr
  "Refresh"             f.refresh
  "Refresh Window"      f.winrefresh
  "twm version"         f.version
  "Focus on Root"       f.unfocus
  "Source .twmrc"       f.twmrc
  "Cut File"            f.cutfile
  "(De)Iconify"         f.iconify
  "DeIconify"           f.deiconify
  "Move Window"         f.move
  "ForceMove Window"    f.forcemove
  "Resize Window"       f.resize
  "Raise Window"        f.raise
  "Lower Window"        f.lower
  "Raise or Lower"      f.raiselower
  "Focus on Window"     f.focus
  "Raise-n-Focus"       f.function "raise-n-focus"
  "Destroy Window"      f.destroy
  "Kill twm"           f.quit
}

```

¡Guau! Creedme, este no es el **.twmrc** más largo que he llegado a ver. Es probable que con vuestra versión de X vinieran algunos ejemplos de archivos **.twmrc** bastante decentes. Se pueden buscar dentro del directorio `/usr/lib/X11/twm/` o en `/usr/X11/lib/X11/twm`.

Un defecto de programa que hay que vigilar en los ficheros **.twmrc** es no olvidarse de poner el “&” después de un comando en un menú. Si se nota que X se cuelga cuando se usan ciertos

comandos, lo más probable es que ésta sea la causa. Hay que salir de X con **Ctrl**-**Alt**-**Retroceso**, editar **.twmrc**, y probar otra vez.

9.2.2 Configuración de fvwm

Si se usa **fvwm**, el directorio **/usr/lib/X11/fvwm/** (o **/usr/X11/lib/X11/fvwm/**) tiene algunos ejemplos de ficheros de configuración.

[Compañeros: No sé nada sobre **fvwm**, aunque supongo que sacaré algo en claro de los ficheros de ejemplo, igual que el lector :-). Además, dado el pequeño pero decente **system.twmrc** en el directorio arriba mencionado, me pregunto si vale la pena que de este largo ejemplo con mi propio **.twmrc**. De momento está aquí, pero no sé si lo dejaremos o no.-Karl]

9.3 Otros ficheros de inicio

Otros ficheros de inicio dignos de mención son:

.emacs	Leído por el editor de texto Emacs cuando arranca.
.netrc	Da los nombres y contraseñas por defecto para ftp.
.rhosts	Hace una cuenta accesible remotamente.
.forward	Para redirección automática del correo.

9.3.1 El fichero de configuración de Emacs

Si se usa **emacs** como editor principal, entonces el fichero **.emacs** es muy importante. Se discute con extensión en el Capítulo 8.

9.3.2 Configuración por defecto del FTP

El fichero **.netrc** permite tener ciertas configuraciones por defecto cada vez que se arranca **ftp**. Aquí hay un pequeño **.netrc** de ejemplo:

```
machine floss.life.uiuc.edu login larry password fishSticks
machine darwin.life.uiuc.edu login larry password fishSticks
machine geta.life.uiuc.edu login larry password fishSticks
machine phylo.life.uiuc.edu login larry password fishSticks
machine ninja.life.uiuc.edu login larry password fishSticks
machine indy.life.uiuc.edu login larry password fishSticks

machine clone.mcs.anl.gov login fogel password doorm@
machine osprey.mcs.anl.gov login fogel password doorm@
machine tern.mcs.anl.gov login fogel password doorm@
machine altair.mcs.anl.gov login fogel password doorm@
machine dalek.mcs.anl.gov login fogel password doorm@
machine juju.mcs.anl.gov login fogel password doorm@
```

```
machine sunsite.unc.edu login anonymous password larry@cs.oberlin.edu
```

Cada línea de **.netrc** especifica un nombre de máquina, un nombre de cuenta de acceso a usar por defecto con esa máquina, y una contraseña. Esto es muy útil si se usa **ftp** continuamente y se está harto de teclear el nombre de acceso y la contraseña en cada uno. El programa de **ftp** intentará acceder automáticamente usando los datos del fichero **.netrc**, si se hace **ftp** a una de las máquinas listadas allí.

Se puede obligar a **ftp** a ignorar el fichero **.netrc** y no intentar acceder de manera automática, arrancándolo con la opción “-n”: “**ftp -n**”.

Hay que asegurarse de que el fichero **.netrc** *sólo* es legible por el usuario. Se puede usar el programa **chmod** para establecer los permisos de lectura. Si otra gente puede leerlo, significa que pueden encontrar su contraseña de acceso a varios otros ordenadores. Esto es un agujero en la seguridad tan grande como uno pueda tenerlo. Como recomendación para ser cuidadoso, **ftp** y otros programas que miran el fichero **.netrc** no funcionarán si los permisos de lectura del fichero son incorrectos.

Hay mucho más sobre el fichero **.netrc** de lo que se ha explicado aquí. Cuando se tenga una oportunidad, hay que probar “**man .netrc**” o “**man ftp**”.

9.3.3 Permitiendo un acceso remoto sencillo a su cuenta

Si tiene un fichero **.rhosts** en su directorio **\$HOME**, le permitirá usar programas en esta máquina de forma remota. Por ejemplo, se puede estar registrado en la máquina **cs.oberlin.edu**, pero con un **.rhosts** correctamente configurado en **floss.life.uiuc.edu**, se puede usar un programa en **floss** y presentar el resultado en **cs**, sin tener que registrarse o teclear una contraseña frente a **floss**.

El archivo **.rhosts** se ve más o menos así:

```
frobnozz.cs.knowledge.edu jsmith  
aphrodite.classics.hahvaahd.edu wphilps  
frobbo.hoola.com trixie
```

El formato es muy directo: un nombre de máquina, seguido de un nombre de usuario. Supongamos que este ejemplo es en realidad mi archivo **.rhosts** en **floss.life.uiuc.edu**. Esto significaría que puedo usar programas en **floss**, y ver el resultado en cualquiera de las máquinas listadas, siempre y cuando yo esté también registrado como el usuario correspondiente a cada máquina cuando intento ejecutar el programa.

El mecanismo exacto con el que uno usa un programa remoto es normalmente el programa **rsh**. Significa “remote shell”(shell remoto), y lo que hace es iniciar un shell en la máquina remota y ejecutar el comando especificado. Por ejemplo:


```
frobbo$ whoami
trixie
frobbo$ rsh floss.life.uiuc.edu "ls ~"
foo.txt    mbox    url.ps    snax.txt
frobbo$ rsh floss.life.uiuc.edu "more ~/snax.txt"
[snax.txt aparece pagina a pagina aqui]
```

El usuario “trixie” en “floss.life.uiuc.edu”, que tenía el fichero **.rhosts** de ejemplo mostrado antes, permite explícitamente a “trixie” en “frobbo.hoola.com” usar programas como si fuera “trixie” de “floss”.

No hay porqué tener el mismo nombre de usuario en todas las máquinas para que **.rhosts** funcione. Se puede usar la opción “-l” con **rsh** para especificar a la máquina remota cual nombre de usuario se quiere usar para registrarse. Si ese nombre existe en la máquina remota, y tiene un fichero **.rhosts** con el nombre de la máquina local y el nombre de usuario correspondiente, entonces la orden incluida con **rsh** se ejecutará.

```
frobbo$ whoami
trixie
frobbo$ rsh -l larry floss.life.uiuc.edu "ls ~"
[Inserta aqui un listado de mi directorio en floss]
```

Esto funcionará si el usuario “larry” en “floss.life.uiuc.edu” tiene un fichero **.rhosts** que permite a “trixie” de “frobbo.hoopla.com” usar programas en su cuenta. Si son o no la misma persona es irrelevante: la única cosa importante son los nombres de usuario, los nombres de las máquinas y la fila referente a “floss” en el **.rhosts** de Larry. Nótese que el fichero **.rhosts** de Trixie en **frobbo** no importa, sólo cuenta el de la máquina remota.

Hay otras combinaciones que se pueden poner en un fichero **.rhosts** —por ejemplo, se puede dejar en blanco el nombre de usuario detrás del nombre de una máquina remota, para permitir a cualquier usuario de esa máquina que ejecute programas en la máquina local. Esto es, por supuesto, un riesgo en la seguridad: alguien podría ejecutar remotamente un programa que borre mis archivos, por el simple hecho de tener una cuenta en cierta máquina. Si se va a hacer cosas como esta, entonces hay que asegurarse de que el fichero **.rhosts** es legible sólo por el usuario implicado, y por nadie más.

9.3.4 Redirección de correo

También se puede tener un fichero **.forward**, el cual no es un fichero de configuración estrictamente hablando. Si contiene direcciones de e-mail, entonces todo el correo que llegue al usuario se redirigirá a esas direcciones. Esto es útil cuando se tienen cuentas en varios sistemas, pero se quiere leer el correo desde un solo sitio.

Hay una gran cantidad de otros ficheros de configuración posibles. La cantidad exacta variará de sistema a sistema, y depende de los programas instalados. Una manera de saber más es mirar todos los archivos en el directorio **\$HOME** que empiecen por “.”. No es seguro que todos sean ficheros de configuración, pero seguro que la mayoría sí lo es.

9.4 Veamos algunos ejemplos

El ejemplo definitivo que puedo dar es mostrar un sistema Linux en funcionamiento. Así que, si tenéis acceso a Internet, estáis invitados a hacer `telnet` a `floss.life.uiuc.edu`. Registráos como “guest”, contraseña “explorer”, y husmead por dentro. Muchos de los archivos de ejemplo se pueden encontrar en `/home/kfogel`, pero también hay otros directorios de usuarios. Sois libres de copiar cualquier cosa que se pueda leer. Por favor tened cuidado: `floss` no es un sitio terriblemente seguro, y ciertamente se puede conseguir acceso como ‘root’ si uno lo intenta lo suficiente. Prefiero creer en la confianza, más que en la vigilancia constante, para mantener la seguridad.

Capítulo 10

Hablando con los demás

“Una noción fundamental de Usenet es que es una cooperativa.”

Cuando ya van para diez los años que llevo en Usenet, discrepo con lo anterior. La noción fundamental de Usenet es el insulto airado.

Chuq Von Rospach

Los modernos sistemas operativos tipo Unix son muy buenos a la hora de hablar con otras máquinas y funcionar en red. Dos máquinas Unix diferentes pueden intercambiar información de muchas, muchas maneras distintas. Este capítulo va a intentar contar cómo se puede sacar provecho de esta gran facilidad para la comunicación.

Trataremos de cubrir el correo electrónico, las noticias de Usenet, y diversas utilidades básicas de comunicaciones de Unix.

10.1 Correo electrónico

Una de las facilidades más populares que ofrece Unix de forma estándar es el correo electrónico. Con él evitamos el habitual follón de buscar un sobre, una hoja de papel, un lápiz, un sello, y un servicio postal. A cambio debemos enfrentarnos al follón de pelear con el ordenador.

10.1.1 Enviar correo

Todo lo que hay que hacer es teclear “`mail nombre`” y escribir el mensaje.

Por ejemplo, supongamos que queremos enviar correo a un usuario que se llama **sam**:

```
/home/larry$ mail sam
Subject: Documentacion del usuario
Solo estoy probando el sistema de correo.
[Ctrl-d]
```

```
/home/larry$  
/home/larry$
```

El programa **mail** es muy simple. Como **cat**, acepta texto de la entrada estándar, línea a línea, hasta que recibe un carácter de fin-de-texto como único carácter en una línea: **Ctrl-d**. Así, para enviar nuestro mensaje, tuvimos que pulsar **Intro** y después **Ctrl-d**.

Usar **mail** es la manera más rápida de mandar correo, y es bastante útil cuando se usa con redirección de entrada y/o salida. Por ejemplo, si quisiéramos enviar por correo electrónico el fichero **report1** a “Sam”, podríamos escribir “**mail sam < report1**”, o incluso “**sort report1 | mail sam**”.

Sin embargo, la desventaja de usar **mail** para enviar correo es su primitivísimo editor. ¡No se puede cambiar ni una línea una vez se ha pulsado **Intro**! Así que yo recomiendo mandar correo (cuando no se use redirección) con el modo mail del Emacs. El tema se trata en la Sección 8.10.

10.1.2 Leer el correo

mail [*nombre*]

El programa “**mail**” ofrece una forma algo liosa de leer el correo. Si se teclea “**mail**” sin parámetros, aparecerá lo siguiente:

```
/home/larry$ mail  
No mail for larry  
/home/larry$
```

“No mail for larry” significa: No hay correo para **larry**.

Me voy a enviar correo a mí mismo para poder utilizar el lector:

```
/home/larry$ mail larry  
Subject: Sapos!  
Y ranas!  
EOT  
/home/larry$ echo "culebras" | mail larry  
/home/larry$ mail  
Mail version 5.5 6/1/90. Type ? for help.  
"/usr/spool/mail/larry": 2 messages 2 new  
>N  1 larry                Tue Aug 30 18:11  10/211  "Sapos!"  
   N  2 larry                Tue Aug 30 18:12   9/191  
&
```

El símbolo petición de órdenes del programa **mail** es un “et a la inglesa” (“&”). Acepta un par de órdenes sencillas, y muestra una breve pantalla de ayuda si se pulsa “?” seguido de **Intro**.

Las órdenes básicas de **mail** son:

- t** *lista-de-mensajes* Mostrar los mensajes en la pantalla.
- d** *lista-de-mensajes* Borrar los mensajes.
- s** *lista-de-mensajes fichero* Guardar los mensajes en *fichero*.
- r** *lista-de-mensajes* Responder a los mensajes—esto es, comenzar a escribir un nuevo mensaje dirigido a quien fuera que envió los mensajes referidos.
- q** Salir y guardar los mensajes que no hayan sido borrados en un fichero llamado **mbox** en el directorio raíz del usuario.

¿Qué es una *lista-de-mensajes*? Consiste en una lista de números separados por espacios, o incluso un rango de los mismos como “2-4” (lo que es equivalente a “2 3 4”). Así mismo se puede introducir el nombre del remitente, de este modo la orden **t sam** mostrará todos los mensajes provenientes de Sam. Si se omite la *lista-de-mensajes*, se supone que la orden se refiere al último mensaje mostrado.

Leer el correo con el programa **mail** presenta varios inconvenientes. El primero es que si un mensaje es más largo que la pantalla, ¡el programa **mail** no se detiene! Habrá que guardar el mensaje en un fichero y usar **more** sobre él después. El segundo es que no ofrece una buena interfaz para manejar el correo antiguo —cuando se quiere guardar el correo para leerlo más tarde.

Emacs también permite leer el correo con **rmail**, pero este libro no lo cubre. Adicionalmente la mayoría de los sistemas Linux vienen con algunos lectores de correo adicionales además de **mail**, como **elm**, o **pine**.

10.2 Noticias más que de sobra

10.3 Localizar a la gente

El programa **finger** permite obtener información de otros usuarios del sistema y de todo el mundo. Sin duda el programa **finger** recibió este nombre por los anuncios de la AT&T que exhortaban a la gente a “salir y tocar a alguien”. Dado que el Unix tiene sus orígenes en la AT&T, probablemente eso le hiciera gracia al autor.

```
finger [-slpm] [usuario][@maquina]
```

Los parámetros opcionales de **finger** pueden resultar algo liosos. La verdad es que no es tan difícil. Se puede solicitar información de un usuario local (“**sam**”), información de otra máquina (“@**lionsden**”), información sobre un usuario remoto (“**sam@lionsden**”), o simplemente información sobre la máquina local (nada).

Otra característica es que si se pide información sobre un usuario y no hay ningún nombre de cuenta que coincida exactamente con el dado, tratará de emparejar el nombre real con lo que se

haya especificado. Eso significa que si ejecutamos “finger Greenfield”, nos contestará que existe la cuenta sam de *Sam Greenfield*.

```
/home/larry$ finger sam
Login: sam                               Name: Sam Greenfield
Directory: /home/sam                     Shell: /bin/tcsh
Last login Sun Dec 25 14:47 (EST) on tty2
No Plan.
/home/larry$ finger greenfie@gauss.rutgers.edu
[gauss.rutgers.edu]
Login name: greenfie                      In real life: Greenfie
Directory: /gauss/u1/greenfie             Shell: /bin/tcsh
On since Dec 25 15:19:41 on ttyp0 from tiptop-slip-6439
13 minutes Idle Time
No unread mail
Project: Debes estar bromeando! de que proyecto me hablas?
No Plan.
/home/larry$ finger
Login      Name                Tty  Idle  Login Time   Office   Office Phone
larry     Larry Greenfield      1    3:51  Dec 25 12:50
larry     Larry Greenfield      p0           Dec 25 12:51
/home/larry$
```

La opción “-s” hace que **finger** utilice siempre la forma abreviada (la que se obtiene normalmente cuando se hace **finger** a una máquina), y la opción “-l” hace que use siempre la forma larga, incluso al hacer **finger** a una máquina. La opción “-p” hace que **finger** no muestre los ficheros **.forward**, **.plan**, ni **.project**, y la “-m” hace que, cuando se ha pedido información sobre un usuario, sólo de información de un nombre de cuenta —que no intente emparejar el nombre que se le da con un nombre real.

10.3.1 Planes y proyectos

Pero, ¿qué es eso de **.plan** y **.project**? Son ficheros que existen en el directorio raíz de un usuario y que se muestran cuando se le hace **finger**. Cada uno puede crear sus propios ficheros **.plan** y **.project** —la única restricción es que sólo se muestra la primera línea del fichero **.project**.

Además, todo el mundo debe tener permisos de búsqueda (la “x” de ejecución) del directorio raíz del usuario en cuestión (“**chmod a+x ~/**”) y debe ser posible para todo el mundo leer los ficheros **.plan** y **.project** (“**chmod a+r ~/.plan ~/.project**”).

10.4 Uso remoto de sistemas

telnet *sistema-remoto*

La forma más habitual de usar un sistema Unix remoto es con **telnet**. **telnet** es normalmente un programa muy sencillo de usar:

```
/home/larry$ telnet lionsden
Trying 128.2.36.41...
Connected to lionsden
Escape character is '^]'.
```

```
lionsden login:
```

Como puede verse, después de teclear “**telnet**”, se nos presenta un mensaje de acceso al sistema remoto. Podemos introducir cualquier nombre de usuario (¡siempre que conozcamos la contraseña!) y después usar el sistema remoto casi como si estuviéramos sentados allí.

La manera normal de salir de **telnet** es hacer “**logout**” en el sistema remoto. Otra forma es tecleando la secuencia de escape, que (como en el ejemplo de arriba) es habitualmente **Ctrl-]**. Esto nos presenta un nuevo símbolo de petición de órdenes: **telnet>**. Entonces podemos teclear “**quit**” y **Intro** y la conexión con el otro sistema será cerrada y el programa **telnet** finalizará. (Si cambiáramos de idea, simplemente pulsando **Intro** volveríamos al sistema remoto).



Si se están usando las X, se puede crear un nuevo **xterm** para el otro sistema al que estamos saltando. Con la orden “**xterm -title "lionsden" -e telnet lionsden &**” crearemos una nueva ventana **xterm** que automáticamente ejecute **telnet**. (Si eso se hace con frecuencia, probablemente será deseable crear un alias o un programa de shell que lo haga automáticamente.)

10.5 Intercambio de ficheros

ftp sistema-remoto

La manera usual de intercambiar ficheros entre sistemas Unix es **ftp**, acrónimo que proviene de **file transfer protocol** (protocolo de transferencia de ficheros). Tras ejecutar la orden **ftp**, se nos pedirá que nos identifiquemos, de forma muy parecida a con **telnet**. Después de hacerlo aparece un símbolo de petición de órdenes especial: el símbolo **ftp**.

La orden “**cd**” funciona de la manera habitual, pero en el sistema remoto: cambia de directorio en el *otro* sistema. De la misma forma, la orden “**ls**” lista los ficheros en el sistema remoto.

Las dos órdenes más importantes son “**get**” y “**put**”. **get** transfiere un fichero del sistema remoto al local, y **put** lleva un fichero del sistema local al remoto. Ambas órdenes tienen efecto en el directorio local en el que se haya arrancado **ftp** y en el directorio remoto en el que se esté en ese momento (que se puede haber cambiado con la orden “**cd**”).

Un problema habitual con **ftp** es la distinción entre ficheros de texto y binarios. **ftp** es un protocolo muy antiguo, y antes tenía ventajas suponer que los ficheros que se transferían eran de texto. El comportamiento por omisión de algunas versiones de **ftp** es ése, lo que significa que los

programas que se envíen o reciban se corromperán. Por seguridad conviene usar la orden “binary” antes de empezar a transferir ficheros.

Para salir de `ftp` se usa la orden “bye”.

10.6 Viajando por la telaraña

La Telaraña Mundial (World Wide Web), o WWW, es un uso muy popular de la Internet. Consiste en un conjunto de **páginas**, cada una con su propia URL—*Localizador Uniforme de Recursos* asociado. Los URLs son simpáticas secuencias de caracteres de la forma `<http://www.rutgers.edu/>`. Las páginas están por lo general escritas en HTML (*Lenguaje de Composición de Hipertexto*).

El HTML permite al autor de un documento enlazar ciertas palabras, frases, o imágenes, con otros documentos que se hallen en cualquier lugar de la telaraña. Cuando un usuario lee un documento, puede moverse rápidamente a otro picando en una palabra clave o en un botón y se le mostrará el otro documento en cuestión—que posiblemente esté a miles de kilómetros del primero.

`netscape [url]`



El navegador más popular de Linux es **netscape**; se trata de un navegador comercial que vende (y regala) Netscape Communications Corporation. **netscape** sólo funciona con las X.

netscape trata de ser tan fácil de usar como sea posible y utiliza el conjunto de widgets Motif para lograr una apariencia muy a lo Microsoft Windows. El conocimiento básico para poder usar **netscape** es que las palabras azules subrayadas son enlaces, así como muchas de las imágenes. (Se puede averiguar cuales imágenes son enlaces picando en ellas.) Picando en esas palabras con el botón izquierdo del ratón, aparece una nueva página.

Linux soporta muchos navegadores más, incluyendo el navegador original **lynx**. **lynx** es un navegador orientado a texto—no muestra ninguna imagen que es con lo que actualmente se relaciona al Web—pero funciona sin las X.

`lynx [url]`

Es un poco más complicado aprender a usar **lynx**, pero normalmente se le coge el truco después de jugar un rato con las flechas del cursor. Las flechas hacia arriba y hacia abajo nos mueven entre los enlaces de una página. La flecha hacia la derecha sigue el enlace seleccionado (resaltado). La flecha hacia la izquierda vuelve a la página anterior. Para salir de **lynx**, se debe pulsar **q**. **lynx** tiene muchas órdenes más—consúltase la página del manual.

Capítulo 11

Comandos divertidos

Bueno, la mayoría de la gente que tiene que ver con los comandos de UNIX expuestos en este capítulo no estarán de acuerdo con este título. “¡Que diablos! Se me ha enseñado que el interfaz del Linux es muy estándar, y ahora tenemos un grupo de comandos, cada uno trabajando de una manera completamente diferente. Nunca recordaré todas esas opciones, ¿y dice que son *divertidos*?” Sí, Vd. ha visto un ejemplo del humor de los hackers¹. Además, mírelo desde el lado positivo: no hay comandos del MS-DOS equivalentes a éstos. Si los necesita, tiene que adquirirlos por separado, y nunca sabrá como será su interfaz. Aquí hay un útil —y económico— valor añadido, ¡¡así que disfrútelo!!.

El conjunto de comandos explicados en este capítulo cubre `find`, que permite al usuario buscar grupos de ficheros especificados dentro del árbol de directorios; `tar`, útil para empaquetar algún archivo para ser enviado o sólo salvado; `dd`, el multicopista de bajo nivel; y `sort`, el cual . . . sí, clasifica ficheros. Una última condición: estos comandos no están bajo ningún concepto estandarizados, y si bien podríamos encontrar un núcleo de opciones comunes en todos los sistemas *IX, las versiones (de GNU) que se explican más abajo, y Vd. puede encontrar en su sistema Linux, usualmente poseen muchas más capacidades. Por tanto, si Vd. planea usar otros sistemas operativos UNIX, por favor, no olvide comprobar la página del manual del sistema en cuestión para aprender las quizás no tan pequeñas diferencias.

11.1 `find`, el buscador de ficheros

11.1.1 Generalidades

Entre los diversos comandos vistos hasta aquí, había algunos que permitían al usuario bajar recursivamente el árbol del directorio para llevar a cabo alguna acción: los ejemplos canónicos son “`ls -R`” y “`rm -R`”. Bien. `find` es *el* comando recursivo. Cada vez que piense “Bueno, tengo que hacer tal cosa con todos esos ficheros en mi propia partición”, haría mejor en pensar en usar `find`. En

¹N.T.: La palabra Hacker en este contexto se refiere a una persona con profundos conocimientos de informática.

cierto sentido el hecho que **find** encuentre ficheros es solo un efecto secundario: su ocupación real es evaluar.

La estructura básica del comando es como sigue:

```
find ruta [...] expresión [...]
```

Esto es al menos en la versión de GNU; otras versiones no permiten especificar más que una ruta, y además es muy infrecuente la necesidad de hacer tal cosa. La explicación burda de la sintaxis del comando es bastante simple: *Vd.* indica desde dónde quiere empezar la búsqueda (la parte de la *ruta*; con el **find** de GNU puede omitir esto y será tomado por defecto el directorio en uso *.*), y que clase de búsqueda quiere realizar (la parte de la *expresión*).

El comportamiento estándar del comando es un poco engañoso, por lo que más vale tenerlo en cuenta. Supongamos que en su directorio principal de usuario existe un directorio llamado **basura**, el cual contiene un fichero **foobar**. *Vd.* felizmente escribe “**find . -name foobar**” (lo que como puede adivinar busca ficheros llamados **foobar**), y obtiene ... nada más que otra vez el prompt. El problema reside en el hecho de que **find** es por defecto un comando silencioso; sólo devuelve 0 si la búsqueda fue completada (con o sin haber encontrado algo) o un valor distinto de cero si hubiera habido algún problema. Esto no ocurre con la versión que *Vd.* puede encontrar en el Linux, pero de todas maneras es útil recordarlo.

11.1.2 Expresiones

La parte de la *expresión* puede ser dividida en cuatro grupos diferentes de palabras clave: *opciones*, *tests*, *acciones*, y *operadores*. Cada uno de ellos puede devolver un valor verdadero/falso, junto con un efecto secundario. La diferencia entre los grupos se muestra a continuación.

opciones afecta a la función general de **find**, más que al procesamiento de un solo fichero. Un ejemplo es “**-follow**”, el cual instruye a **find** para seguir enlaces simbólicos en vez de sólo presentar el nodo-i. Siempre devuelven verdadero.

tests son verdaderos tests (por ejemplo, “**-empty**” comprueba si el fichero está vacío), y puede devolver verdadero o falso.

acciones tienen también un efecto secundario sobre el nombre del fichero considerado. Pueden asimismo devolver verdadero o falso.

operadores no devuelven realmente un valor (convencionalmente pueden ser considerados como verdaderos), y se usan para construir expresiones sintéticas. Un ejemplo es “**-or**”, el cual hace la OR lógica de las dos subexpresiones a su lado. Note que cuando se juxtaponen expresiones, está involucrada una “**-and**”.

Debe advertir que **find** depende del intérprete de comandos para tener la línea de comandos analizada sintácticamente: esto significa que todas las palabras clave deben ser introducidas entre

espacios en blanco y además, que hay un montón de caracteres bonitos que deberán precederse de escape, si no serían destrozados por el propio intérprete de comandos. Cada vía de escape (barra invertida, simples y dobles comillas) está bien; en los ejemplos las palabras clave de un solo carácter se señalarán en general mediante con una barra invertida, porque es la manera más simple (al menos en mi opinión. ¡Y soy yo quien está escribiendo estas notas!).

11.1.3 Opciones

Aquí está la lista de todas las opciones conocidas de la versión GNU de **find**. Recuerde que siempre devuelven verdadero.

-daystart mide el tiempo transcurrido no desde hace 24 horas si no desde la última medianoche. Un auténtico hacker probablemente no entenderá la utilidad de tal opción, pero un trabajador que programa de ocho a cinco si lo apreciará.

-depth procesa el contenido de cada directorio antes que el directorio en si. A decir verdad, no conozco muchos usos de esto, aparte de una emulación del comando “**rm -F**” (por supuesto Vd. no puede borrar un directorio antes de que todos sus ficheros sean borrados también ...

-follow respeta (es decir, sigue) los enlaces simbólicos. Implica la opción “**-noleaf**”; ver abajo.

-noleaf desactiva una optimización que dice “Un directorio contiene dos subdirectorios menos que su cuenta de enlaces rígidos²”. Si el mundo fuera perfecto, todos los directorios serían referenciados por cada uno de sus subdirectorios (a causa de la opción **-follow**), como **.** dentro de sí mismo, y por su nombre “real” desde su directorio padre.

Esto significa que cada directorio debe ser referenciado al menos dos veces (una por sí mismo, otra por su directorio padre) y cualquier referencia adicional es por subdirectorios. En la práctica, sin embargo, los enlaces simbólicos y los sistemas de fichero distribuidos³ pueden alterar esto. Esta opción hace que **find** funcione ligeramente más despacio, pero debe dar los resultados esperados.

-maxdepth *niveles*, **-mindepth *niveles***, donde *niveles* es un entero no negativo. Indican respectivamente que a lo sumo o que al menos *niveles* de directorios deberían explorarse. Un par de ejemplos son obligados: “**-maxdepth 0**” indica que el comando debería realizarse sólo sobre los argumentos de la línea de comandos, es decir, sin bajar recursivamente el árbol de directorios; “**-mindepth 1**” inhibe el procesamiento del comando para los argumentos en la línea de comandos, pero se consideran todos los otros ficheros debajo.

-version imprime la versión en uso del programa.

-xdev, el cual es un nombre engañoso, instruye a **find** para **no** cruzarse de dispositivo⁴, es decir, no cambiarse de sistema de ficheros. Es muy útil cuando se tiene que buscar algo en el

²N.T. hard link count

³Los sistemas de fichero distribuidos permiten a los ficheros mostrarse como si fueran locales a una máquina cuando realmente están situados en algún otro lugar.

⁴N.T.: to cross device

sistema de ficheros raíz; en muchas máquinas es una partición algo pequeña, pero, sino, ¡un “find /” buscaría en la estructura entera! Podemos así evitar los sistemas de archivo que no son promisorios.

11.1.4 Test

Los dos primeros test son muy simples de comprender: “-false” siempre devuelve falso, mientras que “-true” siempre devuelve verdadero. Otros test que no necesitan la especificación de un valor son “-empty”, el cual devuelve verdadero si el fichero está vacío, y el par “-nouser”/“-nogroup”, los cuales devuelven verdadero en el caso que ninguna entrada en **/etc/passwd** o **/etc/group** cuadre con el identificador de usuario/grupo del propietario del fichero. Veamos un problema común en los sistemas multiusuario: se borra un usuario, pero los ficheros pertenecientes a él permanecen en la parte más extraña de los sistemas de ficheros, y debido a las leyes de Murphy ocupan mucho espacio. El par de test que hemos visto en último lugar nos permitirá encontrar esos archivos.

Por supuesto, es posible buscar un usuario o grupo específico. Los test son “-uid *nn*” y “-gid *nn*”. Desafortunadamente no es posible dar directamente el nombre del usuario, sino que es necesario usar el identificador numérico, *nn*.

Además está permitido usar las formas “+*nn*”, que significa “un valor estrictamente más grande que *nn*”, y “-*nn*”, que significa “un valor estrictamente más pequeño que *nn*”. Esto es bastante estúpido en el caso de identificadores de usuario⁵, pero se volverá práctico con otros test.

Otra opción útil es “-type *c*”, la cual devuelve verdadero si el fichero es del tipo especial de carácter. Los mnemónicos para las posibles selecciones son las mismas letras que se encuentran en **ls**; entonces, tenemos **b**loque cuando el fichero es uno especial de bloque; **d**irectorio; **p**ipes para tuberías designadas⁶; **l**ink para enlaces simbólicos, y **s**ocketsockets⁷. Los ficheros regulares se indican con **f**ile. Un test relacionado es “-xtype”, que es similar a “-type” excepto en el caso de enlaces simbólicos. Si se ha proporcionado la opción “-follow”, se examina el enlace, en lugar del fichero al que se apunta. El test “-fstype *type*” no tiene nada que ver; en este caso, lo que se examina es el tipo del sistema de ficheros. Creo que la información se consigue del fichero **/etc/mstab**, el único que presenta los sistemas de ficheros montados; estoy seguro que los tipos **nfs**, **tmp**, **msdos** y **ext2** están entre los reconocidos.

Los test “-inum *nn*” y “-links *nn*” examinan si el fichero tiene número de nodo-i *nn*, o *nn* enlaces, mientras que “-sizenn” es verdadero si el fichero tiene *nn* bloques de 512 bytes asignados. (Bueno, no precisamente; para ficheros disgregados los bloques no asignados también se cuentan). Como hoy en día el resultado de “**ls -s**” no siempre se mide en trozos de 512 bytes (Linux, por ejemplo, usa 1 kbyte como la unidad), es posible adicionar a *nn* el carácter “**b**”, el cual significa contar en bytes, o “**k**”, contar en kilobytes.

Los bits de permiso son examinados a través del test “-perm *mode*”. Si el *modo* no está antecedido por un signo, entonces los bits de permiso del fichero deben cuadrar exactamente con ellos. Un “-” al principio significa que todos los bits de permiso deben ser puestos a uno, pero no hace

⁵N.T.: UIDs

⁶N.T.: named pipes

⁷N.T.: enchufe

suposiciones sobre los otros; un “+” al principio se satisface sólo si cualquiera de los bits está a uno. ¡Ups! Olvidé decir que el modo se escribe en octal o simbólicamente, como se usa en `chmod`.

El próximo grupo de test está relacionado con el tiempo que hace que se ha utilizado un fichero por última vez. Esto viene bien cuando un usuario ha llenado su espacio, ya que usualmente hay muchos ficheros que no usaba desde hace años, y cuyo significado ha olvidado. El problema es localizarlos, y `find` es la única esperanza a la vista. “`-atime nn`” es verdadero si la última vez que se accedió al fichero fue hace *nn* días; “`-ctime nn`” si el estado del fichero fue cambiado por última vez hace *nn* días —por ejemplo, con un `chmod`— y “`-mtime nn`” si el fichero fue modificado por última vez hace *nn* días. Algunas veces Vd. necesita una marca de tiempo ⁸ más precisa; el test “`-newer archivo`” se satisface si el fichero considerado ha sido modificado más tarde que el *archivo*. Por lo tanto, Vd. sólo tiene que usar `touch` con la fecha deseada, y ya está. El `find` de GNU añade los test “`-anewer`” y “`-cnewer`” los cuales se comportan de una manera similar; y los test “`-amin`”, “`-cmin`” y “`-mmin`” que cuentan el tiempo en minutos en vez de períodos de 24 horas.

Por último pero no menos importante, el test que yo uso más a menudo. “`-name patrón`” es verdadero si el nombre del fichero concuerda exactamente con *patrón*, el cual es más o menos el único que se usaría en un `ls` estándar. (¿Por qué ‘más o menos’? Porque, por supuesto, Vd. tiene que recordar que todos los parámetros son procesados por el intérprete de comandos, y esos encantadores metacaracteres sufren una expansión. Así, un test como “`-name foo*`” no devolverá lo que Vd. desea, y Vd. debería escribir cualquiera de los dos “`-name foo`” o “`-name "foo*"`”. Este es probablemente uno de los más comunes errores cometidos por usuarios descuidados, así que escríbalo en letras GRANDES en su pantalla. Otro problema es que, como con `ls`, no se reconocen los puntos al principio del nombre del archivo. Para hacer frente a esto, puede usar el test “`-path patrón`” el cual no se preocupa acerca de los puntos y barras inclinadas cuando compara la ruta del fichero considerado con *patrón*.

11.1.5 Acciones

Hemos dicho que las acciones son aquellas que realmente hacen algo. Bueno, “`-prune`” más bien *no* hace nada, es decir, al descender el árbol de directorios (a menos que se le agregue “`-depth`”). Se la encuentra usualmente junto con “`-fstype`”, para elegir entre varios sistemas de ficheros a aquellos que deberían examinarse.

Las otras acciones pueden ser divididas en dos amplias categorías;

Acciones que *imprimen* algo. La más obvia de estas —y en realidad, la acción por defecto de `find`— es “`-print`” que imprime el nombre del(os) fichero(s) que concuerde(n) con las otras condiciones en la línea de comandos, y devuelve verdadero. Una variante sencilla de “`-print`” es “`-fprint archivo`”, en la cual se usa *archivo* en vez de la salida estándar; otra es “`-ls`” lista el fichero en uso en el mismo formato que “`ls -dils`”; “`-printf formato`” se comporta más o menos como la función de C `printf()`, para que se pueda especificar que formato debería tener la salida, y “`-fprintf archivo formato`” hace lo mismo, pero escribiéndolo en *archivo*. Estas acciones también devuelven verdadero.

⁸N.T.: timestamp

Acciones que *ejecutan* algo. Su sintaxis es un poco extraña y su uso está muy extendido, por lo que, por favor, considérelas.

“**-exec comando \;**” ejecuta el comando, y la acción devuelve verdadero si el estado retornado por el comando es 0, lo que significa que la ejecución fue correcta. La razón para el “\;” es bastante lógica: **find** no sabe donde termina el comando, y el truco de poner la acción ejecutable al final del comando no es aplicable. Bueno, la mejor manera para señalar el final del comando es usar el caracter que usa el intérprete de comandos para este fin, que es “;”, pero, por supuesto, un punto y coma solo en la línea de comandos sería ”comido” por el intérprete de comandos y nunca llegaría a **find**, así que tienen que ser precedidos de escape. La segunda cosa a recordar es como especificar el nombre del fichero en uso dentro del *comando*, ya que probablemente Vd. hizo todo el esfuerzo al construir la expresión para hacer algo, y no sólo para ver la hora. Esto se hace por medio de la cadena de caracteres “{ }”. Algunas versiones antiguas de **find** requieren que coloque las llaves entre espacios en blanco —no muy práctico si Vd. necesita, por ejemplo, la ruta entera y no sólo el nombre del fichero— pero con el **find** de GNU podría ser cualquier lugar de la cadena de caracteres que compone el *comando*. (¿Y no debería estar precedido de escape o entrecomillado?, seguramente se estará preguntando. Asombrosamente, yo nunca tuve que hacer esto ni bajo **tcsh** ni bajo **bash** (**sh** no considera { y } como caracteres especiales, así que no es ni mucho menos un problema). Mi idea es que los intérpretes de comandos “saben” que { } no es una opción que tenga sentido, por tanto no intentan expandirlas, afortunadamente para **find** que puede obtenerla intacta.

“**-ok comando \;**” se comporta como “**-exec**”, con la diferencia que para cada fichero seleccionado se pregunta al usuario que confirme el comando; si la respuesta empieza por y o Y, se ejecuta, de otra manera no, y la acción devuelve falso.

11.1.6 Operadores

Hay una cierta cantidad de operadores; aquí hay una lista, en orden de prioridad decreciente.

\(expr \)

Fuerza el orden de prioridad. Los paréntesis deben, por supuesto, estar entre comillas, ya que también son muy significativos para el intérprete de comandos.

! expr

-not expr

Cambia el verdadero valor de la expresión, esto es, si *expr* es verdadera, se vuelve falsa. El signo de exclamación no tiene que ser precedido de escape, porque está seguido por un espacio en blanco.

expr1 expr2

expr1 -a expr2

expr1 -and expr2

Todas corresponden con la operación lógica AND. El caso más común es el primero, en el cual está implícita. *expr2* no se evalúa, si *expr1* es falsa.

expr1 -o expr2

expr1 -or expr2

corresponden a la operación lógica OR. *expr2* no es evaluada, si *expr1* es verdadera.

expr1, expr2

es la declaración de lista; ambas *expr1* y *expr2* se evalúan (junto con todos los efectos secundarios, ¡por supuesto!), y el valor final de la expresión es el de *expr2*.

11.1.7 Ejemplos

Sí, `find` tiene demasiadas opciones, lo sé. Pero, hay un montón de casos preparados que valen la pena recordar, porque son usados muy a menudo. Veamos algunos de ellos.

```
$ find . -name foo\* -print
```

encuentra todos los nombres de fichero que empiezan con **foo**. Si la cadena de caracteres está incluida en el nombre, probablemente tiene más sentido escribir algo como `"*foo*"`, en vez de `"foo"`.

```
$ find /usr/include -xtype f -exec grep foobar \
    /dev/null {} \;
```

Es un `grep` ejecutado recursivamente que empieza del directorio `/usr/include`. En este caso, estamos interesados tanto en ficheros regulares como en enlaces simbólicos que apuntan a ficheros regulares, por tanto el test `"-xtype"`. Muchas veces es más simple evitar especificarlo, especialmente si estamos bastante seguros cuales ficheros binarios no contienen la cadena de caracteres deseada. (¿Y por qué el `/dev/null` en el comando? Es un truco para forzar al `grep` a escribir el fichero del nombre donde se ha encontrado un emparejamiento. El comando `grep` se aplica a cada fichero con una invocación diferente, y, por tanto no cree que sea necesario mostrar a la salida el nombre del fichero. Pero ahora hay *dos* ficheros, esto es: ¡el activo y `/dev/null`! Otra posibilidad podría ser redirigir la salida⁹ del comando a `xargs` y dejarle llevar a cabo el `grep`. Yo lo intenté, e hice pedazos completamente mi sistema de ficheros (junto con estas notas que estoy intentando recuperar a mano :-)).

```
$ find / -atime +1 -fstype ext2 -name core \
    -exec rm {} \;
```

Es un trabajo clásico para la tabla de tareas preplaneadas. Borra todos los ficheros llamados **core** en el sistema de ficheros del tipo `ext2` al cual no se ha accedido en las últimas 24 horas. Es posible que alguien quiera usar los ficheros de imagen de memoria¹⁰ para realizar un volcado post mortem, pero nadie podría recordar lo que estuvo haciendo después de 24 horas...

```
$ find /home -xdev -size +500k -ls > piggies
```

⁹N.T.: to pipe

¹⁰N.T.: core file

Es útil para ver quién tiene esos archivos que atascan el sistema de ficheros. Note el uso de “-xdev”; como sólo estamos interesados en un sistema de ficheros, no es necesario descender otro sistema de ficheros montado bajo **/home**.

11.1.8 Una última palabra

Tenga en mente que **find** es un comando que consume mucho tiempo, ya que tiene que acceder a cada uno de los nodos-i del sistema para realizar su operación. Por lo tanto, es sabio combinar cuantas operaciones sean posibles en una única invocación de **find**, especialmente en las tareas de ‘mantenimiento interno’ que usualmente se administran mediante un trabajo de la tabla de tareas planificadas. Un ejemplo informativo es el siguiente: supongamos que queremos borrar los ficheros que acaban en **.BAK** y cambiar la protección de todos los directorios a 771 y todos los ficheros que acaban en **.sh** a 755. Y quizás tengamos montado el sistema de ficheros NFS en un enlace telefónico, y no querramos examinar los ficheros ahí. (¿Por qué escribir tres comandos diferentes? La manera más efectiva para realizar la tarea es ésta:

```
$ find . \( -fstype nfs -prune \) -o \  
    \( -type d      -a -exec chmod 771 {} \; \) -o \  
    \( -name "*.BAK" -a -exec /bin/rm {}    \; \) -o \  
    \( -name "*.sh"  -a -exec chmod 755 {} \; \)
```

Parece feo (¡y con mucho abuso de barras invertidas!), pero mirándolo fijamente revela que la lógica subyacente es bastante sencilla. Recuerde: lo que se hace es una evaluación verdadero/falso; el comando introducido es sólo un efecto secundario. Pero esto significa que se ejecuta sólo si **find** puede evaluar la parte ejecutable de la expresión, esto es sólo si la parte izquierda de la subexpresión se evalúa como verdadera. Así, si por ejemplo el fichero considerado en el momento es un directorio entonces el primer ejecutable se evalúa y el permiso del nodo-i se cambia a 771; de otra manera olvida todo y pasa a la siguiente subexpresión. Probablemente, es más fácil verlo en la práctica que escribirlo; pero después de un momento, llegará a ser una cosa natural para Vd.

11.2 tar, el archivador en cinta

11.2.1 Introducción

11.2.2 Opciones principales

11.2.3 Modificadores

11.2.4 Ejemplos

11.3 dd, el duplicador de datos

La leyenda dice que allá lejos en las nieblas del tiempo, cuando el primer UNIX fue creado, sus desarrolladores necesitaron un comando de bajo nivel para copiar datos entre dispositivos. Como tenían prisa, decidieron tomar prestada la sintaxis usada en las máquinas IBM-360, y desarrollar más tarde un interfaz consistente con la de los otros comandos. El tiempo pasó, y todos se acostumbraron tanto con la manera extraña de usar `dd` que ya ha quedado así. No se si es verdad, pero es una bonita historia para contar.

11.3.1 Opciones

A decir verdad, `dd` no es totalmente distinto de los otros comandos de UNIX: es en realidad un **filtro**, esto es, lee por defecto de la entrada estándar y escribe en la salida estándar. Por tanto, si Vd. escribe sólo `dd`, la terminal se queda quieta, esperando una entrada, y un Ctrl-C es la única cosa con sentido a teclear (para cortarlo).

La sintaxis del comando es como sigue:

```
dd [if=archivo_entrada] [of=archivo_salida] [ibs=cant_bytes] [obs=cant_bytes]
[bs=cant_bytes] [cbs=cant_bytes] [skip=cant_bloques]
[seek=cant_bloques] [count=cant_bloques] [conv={ascii, ebcdic, ibm, block,
unblock,
lcase, ucase, swab, noerror, notrunc, sync}]
```

Todas las opciones son de la forma *opción=valor*. No se permite ningún espacio ni antes ni después del signo de igual; esto solía ser molesto, porque el intérprete de comandos no expandía un nombre de fichero en esta situación, pero la versión de `bash` presente en Linux es bastante inteligente, así que Vd. no tiene que preocuparse por eso. Es también importante recordar que todos los valores numerables (`cant_bytes` y `cant_bloques` citados arriba) pueden ser seguidos por un multiplicador. Las opciones posibles son **b** para bloque, el cual multiplica por 512, **k** para kilobytes (1024), **w** para palabra (2), y **xm** que multiplica por **m**.

El significado de las opciones se explica a continuación.

`if=archivo_entrada` y `of=archivo_salida` instruye a `dd` a respectivamente leer de `archivo_entrada` y escribir a `archivo_salida`. En el último caso, el fichero de salida es truncado al valor dado por “`seek`”, o si la palabra clave no está presente, a 0 (esto es, borrado), antes de realizar la operación. Mire también la opción “`notrunc`”.

`ibs=nn` y `obs=nn` especifican cuantos bytes deben ser leídos o escritos a la vez. Creo que por defecto es 1 bloque, esto es, 512 bytes, pero no estoy muy seguro de ello: desde luego funciona de esa manera con ficheros planos¹¹. Estos parámetros son muy importantes cuando se usa dispositivos especiales como entrada o salida; por ejemplo, leer de la red debería establecer “`ibs`” a 10 kbyte, mientras que una disquetera ¹² de 3.5” de alta densidad tiene como tamaño natural de bloque 18 kbyte. Fallar al establecer estos valores podría repercutir no sólo a largo plazo al llevar a cabo el comando, sino inclusive en errores de expiración de plazo, así que tenga cuidado.

`bs=nn` tanto lee y escribe `nn` bytes a la vez. Este no hace caso a las palabras clave “`ibs`” y “`obs`”.

`cbs=nn` establece el buffer de conversión a `nn` bytes. Este buffer se usa cuando se traduce de ASCII a EBCDIC, o desde un dispositivo no ablocado a uno ablocado¹³. Por ejemplo, los ficheros creados bajo VMS tienen a menudo un tamaño de bloque de 512, así que se tiene que poner “`cbs`” a 1b cuando se lee una cinta externa de VMS. ¡Espero que Vd. no tenga que entretenerse en estas cosas!

`skip=nbl` y `seek=nbl` le dicen al programa que omita `nbl` bloques respectivamente al final de la entrada y al principio de la salida. Por supuesto, el último caso tiene sentido si se da la conversión “`notrunc`” (ver abajo). Cada tamaño de bloque es el valor de “`ibs`” (“`obs`”). Cuidado: si no establece el “`ibs`” y escribe “`skip=1b`” Vd. está realmente omitiendo 512×512 bytes, esto es 256KB. No es precisamente lo que Vd. quería, ¿no?

`count=nbl` pretende copiar sólo `nbl` bloques de la entrada, cada uno de los tamaños dados por “`ibs`”. Esta opción, junto con la previa, se vuelve útil si, por ejemplo, Vd. tiene un fichero corrupto y quiere recuperar cuanto sea posible de él. Vd. debe omitir la parte ilegible y copiar lo que resta.

`conv=conversión,[conversión...]` transforma el fichero de la manera que se especifica en sus argumentos. Transformaciones posibles son “`ascii`”, que convierte de EBCDIC a ASCII; “`ebcdic`” y “`ibm`”, las cuales realizan una conversión inversa (¡sí, no existe una única conversión de EBCDIC a ASCII! La primera es la estándar, pero la segunda funciona mejor cuando se imprimen ficheros en una impresora IBM); “`block`”, la cual rellena registros terminados en nueva línea del tamaño de “`cbs`”, reemplazando los cambios de línea con espacios finales; “`unblock`”, que realiza lo contrario (elimina espacios finales, y los reemplaza con cambios de línea); “`lcase`” y “`ucase`”, para convertir a minúsculas y mayúsculas, respectivamente; “`swab`”, que intercambia cada par de bytes de entrada (por ejemplo, para usar un fichero que contiene enteros de

¹¹ N.T.: plain files

¹² N.T.: floppy

¹³ N. del T.: ablocado se refiere a la característica de agrupamiento en bloques de los datos en el dispositivo.

tipo short escritos en una máquina 680x0 en una máquina basada en Intel se necesita dicha conversión); “**noerror**”, para continuar procesando aún después de encontrar errores; “**sync**”, que rellena los bloques de entrada del tamaño de “**ibs**” con caracteres NUL finales.

11.3.2 Ejemplos

El ejemplo canónico es con el único que Vd. probablemente ha chocado hasta ahora, cuando intentó crear el primer disco de Linux: como escribir a un disco sin un sistema de ficheros MS-DOS. La solución es simple:

```
$ dd if=disk.img of=/dev/fd0 obs=18k count=80
```

Decidí no usar “**ibs**” porque no se cual es el mejor tamaño de bloque para un disco duro, pero en este caso no habría hecho ningún perjuicio si en vez de “**obs**” usara sencillamente el “**bs**” —inclusive podría haber sido un poquito más rápido. Observe la explicación del número de sectores a escribir (18 kbyte es lo que ocupa un sector, así que “**count**” se pone a 80) y el uso del nombre de bajo nivel del dispositivo de la disquetera.

Otra aplicación útil de **dd** está relacionada con la copia de seguridad de la red. Supongamos que estamos en una máquina *alfa* y que en la máquina *beta* está la unidad de cinta **/dev/rst0** con un fichero **tar** que estamos interesados en obtener. Tenemos los mismos privilegios en ambas máquinas, pero no hay espacio en *beta* para volcar ¹⁴ el fichero **tar**. En este caso, podríamos escribir

```
$ rsh beta 'dd if=/dev/rst0 ibs=8k obs=20k' | tar xvBf -
```

para hacer en un solo paso la operación completa. En este caso, hemos usado las facilidades de **rsh** para realizar la lectura de la cinta. Los tamaños de entrada y salida están establecidos a los valores por defecto por estas operaciones, esto es 8 kbyte para la lectura de la cinta y 20 kbyte para la escritura en ethernet; desde el punto de vista de la otra cara del **tar**, hay el mismo flujo de bytes el cual podría ser obtenido de la cinta, excepto el hecho de que llega de una manera bastante errática, y la opción **B** es necesaria.

¡Ah! Casi lo olvidaba: No creo para nada que **dd** sea un acrónimo de “data duplicator”, pero al menos es una manera divertida de recordar su significado...

11.4 sort, el clasificador de datos

11.4.1 Introducción

11.4.2 Opciones

11.4.3 Ejemplos

¹⁴N.T.: dump

Capítulo 12

Errores, equivocaciones, bugs, y otras molestias

Unix nunca fue diseñado para evitar que la gente hiciera cosas estúpidas, porque esa política les habría evitado también hacer cosas inteligentes.

Doug Gwyn

12.1 Evitando errores

Muchos usuarios hablan de su frustración con el sistema operativo Unix en alguna ocasión, a menudo a causa de lo que ellos mismos han hecho. Una característica del sistema operativo Unix que muchos usuarios adoran cuando están trabajando bien, y odian después de una sesión hasta bien avanzada la noche, es que muy pocas órdenes piden confirmación. Cuando un usuario está despierto y funcionando, raramente piensa sobre ello, y esto es una ventaja, ya que le permite trabajar más eficientemente.

De todos modos, hay algunas desventajas. `rm` y `mv` no piden nunca confirmación, y esto conduce frecuentemente a problemas. Por eso, veamos una pequeña lista que puede ayudarle a evitar el desastre total:

- ¡Tenga copias de seguridad! Esto va dirigido especialmente a sistemas de un solo usuario— ¡todos los administradores de sistema deben realizar copias de seguridad de su sistema regularmente! Una vez a la semana es suficiente para salvar muchos ficheros. Para más información consulte *The Linux System Administrator's Guide*.
- Los usuarios individuales deben tener sus propias copias de seguridad, si es posible. Si usa más de una máquina regularmente, intente mantener copias actualizadas de todos sus ficheros en cada una de las máquinas. Si tiene acceso a una unidad de disquetes, puede hacer copias de seguridad en disquetes de su material crítico. En el peor de los casos, mantenga copias adicionales del material más importante que haya en su cuenta *en un directorio separado*.

- Piense sobre las órdenes, especialmente las “destructivas” como `mv`, `rm`, y `cp` antes de actuar. Debe ser también cuidadoso con las redirecciones (`>`) —sobrecribirán sus ficheros cuando no esté prestando atención. Incluso el más inofensivo de los comandos puede convertirse en siniestro:

```
/home/larry/report$ cp report-1992 report-1993 backups
```

puede convertirse fácilmente en desastre:

```
/home/larry/report$ cp report-1992 report-1993
```

- El autor también recomienda, a partir de su propia experiencia personal, no hacer limpieza de ficheros a altas horas de la madrugada. ¿Que su estructura de directorios parece un poco desordenada a la 1:32am? Déjelo estar —un poco de desorden nunca ha dañado un ordenador.
- Sígale la pista a su directorio actual. A veces, el prompt que está usando no muestra en que directorio está usted trabajando, y el peligro acecha. ¡Es triste leer un mensaje en `comp.unix.admin`¹ sobre un usuario `root` que estaba en `/` en vez de en `/tmp`! Por ejemplo:

```
mousehouse> pwd
/etc
mousehouse> ls /tmp
passwd
mousehouse> rm passwd
```

La anterior serie de comandos podría hacer muy infeliz al usuario, al ver como eliminaron el fichero de contraseñas de su sistema. ¡Sin él la gente no puede acceder!

12.2 Qué hacer cuando algo va mal

12.3 No es fallo tuyo

Por desgracia para los programadores del mundo, no todos los problemas son a causa de errores del usuario. Unix y Linux son sistemas complicados, y todas las versiones conocidas tienen errores de programación (bugs²). Algunas veces esos errores son difíciles de encontrar y sólo aparecen bajo ciertas circunstancias.

Ante todo, ¿qué es un error? Un ejemplo de error es si le pide al ordenador que calcule “5+3” y contesta “7”. Aunque este es un ejemplo trivial de que puede ir mal, la mayoría de los errores en programas de computadoras se relacionan con la aritmética en alguna forma extremadamente extraña.

¹Un foro de discusión internacional en Usenet, que trata sobre la administración de computadoras Unix.

²Bug: (polilla, *fam.* bicho). Se denomina así a los errores que aparecen en un programa. Cuenta la leyenda que cuando los ordenadores eran unos monstruos llenos de válvulas, que ocupaban habitaciones enteras, un técnico que trataba de resolver un error en el Mark II de Harvard detectó que la causa era una polilla de verdad que se había colado entre las válvulas y provocaba pequeños cortocircuitos al revolotear de un lado a otro.

12.3.1 Cuando hay un error

Si la computadora da una respuesta errónea (¡compruebe que la respuesta es errónea!) o se bloquea, eso es un error. Si cualquier programa se bloquea o da un mensaje de error del sistema operativo, eso es un error.

Si un comando no finaliza nunca su ejecución, puede ser un error, pero debe asegurarse de que no le ha pedido que esté durante mucho tiempo haciendo lo que usted quería que hiciera. Pida ayuda si no sabe lo que hacía el comando.

Algunos mensajes le alertarán de la existencia de errores. Algunos mensajes no son errores. Revise la sección 3.4 y el resto de la documentación para estar seguro de que no son mensajes informativos normales. Por ejemplo, mensajes como “disk full” (disco lleno) o “lp0 on fire” (lp0 ardiendo) no son problemas de software, sino algo incorrecto en su hardware—no hay suficiente espacio libre en el disco, o la impresora está mal.

Si no puede encontrar información sobre un programa, es un error en la documentación, y debería ponerse en contacto con el autor de dicho programa y ofrecerse para escribirla usted mismo. Si algo está incorrecto en la documentación existente³, es un error en ese manual. Si algo aparece incompleto o poco claro en el manual, eso es un error.

Si no puede vencer al **gnuchess** al ajedrez, es un fallo de diseño en el algoritmo de ajedrez que usted usa, pero no necesariamente un error en su cerebro.

12.3.2 Notificando un error

Cuando esté seguro de haber encontrado un error, es importante asegurarse de que su información llega al lugar adecuado. Intente encontrar que programa causa el error—si no puede encontrarlo, tal vez pueda pedir ayuda en `comp.os.linux.help` o `comp.unix.misc`. Una vez encuentre el programa, intente leer la página del manual para ver quien lo escribió.

El método preferido para enviar notificaciones de errores en el mundo Linux es vía correo electrónico. Si no tiene acceso al correo electrónico puede ponerse en contacto con la persona que le suministró Linux —eventualmente, encontrará alguien que o bien tiene correo electrónico, o vende Linux comercialmente y por tanto quiere eliminar el mayor número de errores posibles. Recuerde, en todo caso que nadie está obligado a corregir ningún error a menos que tenga un contrato.

Cuando envíe una notificación de error, incluya toda la información que se le ocurra. Esto incluye:

- Una descripción de lo que usted piensa que es incorrecto. Por ejemplo, “Obtengo 5 cuando calculo 2+2” o “Dice `segmentation violation -- core dumped`”. Es importante decir exactamente que esté sucediendo para que el responsable del mantenimiento pueda corregir *su* error.
- Incluya cualquier variable de entorno relevante.
- La versión de su núcleo (mire en el fichero `/proc/version`) y sus bibliotecas de sistema (mire en el directorio `/lib`—si no puede descifrarlo, envíe un listado de `/lib`).

³¡Especialmente en ésta!

- ¿Cómo ejecutó el programa en cuestión?, o, si era un error del núcleo, ¿qué estaba haciendo en ese momento?.
- **Toda** la información complementaria. Por ejemplo, el comando `w` puede no mostrar el proceso actual para ciertos usuarios. No diga simplemente, “`w` no funciona para cierto usuario”. El error podría ocurrir debido a que el nombre del usuario tiene ocho caracteres de longitud, o cuando está accediendo a través de la red. En su lugar diga, “`w` no muestra el proceso actual para el usuario `greenfie` cuando accede a través de la red”.
- Y recuerde, sea amable. La mayoría de la gente trabaja en el software libre por el gusto de hacerlo, y porque tienen un gran corazón. No los amargue—la comunidad Linux ha desilusionado ya a demasiados desarrolladores, y aún es pronto en la vida del Linux.

Apéndice A

Introducción a vi

`vi` (pronunciado “vi ai” en inglés, o “uve i”) es en realidad el único editor que se puede encontrar en prácticamente cualquier instalación de Unix. Este editor fue escrito originalmente en la Universidad de California en Berkeley y se puede encontrar versiones en casi cualquier edición de Unix, incluido Linux. Al principio cuesta un poco acostumbrarse a él, pero tiene muchas características muy potentes. En general, recomendamos que un nuevo usuario aprenda Emacs, que generalmente es más fácil de usar. Sin embargo, la gente que usa más de una plataforma o que encuentra que no le gusta Emacs, puede estar interesada en aprender `vi`.

Es necesaria una breve reseña histórica de `vi` para comprender cómo la tecla `[k]` puede significar mover el cursor arriba una línea y por qué hay tres diferentes modos de uso. Si no le apetece aprender a usar el editor, los dos tutoriales le llevarán de ser un crudo principiante hasta tener el suficiente conocimiento del grupo de comandos que usted puede llegar a usar. El capítulo también incluye una guía de comandos, que sirve de útil referencia para tenerla cerca del terminal.

Incluso si `vi` no se convierte en su editor de texto normal, el conocimiento de su uso no será desperdiciado. Es casi seguro que el sistema de Unix que use tendrá alguna variante del editor `vi`. Puede ser necesario usar `vi` mientras instala otro editor, como Emacs. Muchas herramientas de Unix, aplicaciones y juegos usan un subconjunto del grupo de comandos de `vi`.

A.1 Una rápida historia de Vi

Los editores de texto en la antigüedad funcionaban en modo de una línea y se usaban típicamente desde terminales no inteligentes de impresión. Un editor típico que operaba en este modo es `Ed`. El editor es potente y eficiente, y usa una cantidad muy pequeña de recursos del sistema, y funcionaba con las pantallas de aquel entonces. `vi` ofrece al usuario una alternativa visual con un grupo de comandos notablemente amplio comparado con `ed`.

`vi` como lo conocemos hoy comenzó como el editor de línea `ex`. De hecho, `ex` se ve como un modo especial de edición en `vi`, aunque ahora lo contrario es verdad. El componente visual de `ex` puede ser iniciado desde la línea de comando usando el comando `vi`, o desde dentro de `ex`.

El editor `ex/vi` fue desarrollado en la universidad de California en Berkeley por William Joy. Originalmente se suministraba como una utilidad no soportada hasta su inclusión oficial en la distribución del Unix AT&T System V. Se ha ido haciendo cada vez más popular, incluso con la competencia de los modernos editores de pantalla completa.

Debido a la popularidad de `vi` existen muchas variantes clónicas y se pueden encontrar varias versiones para la mayoría de los sistemas operativos. No es el propósito de este capítulo el incluir todos los comandos disponibles bajo `vi` o sus variantes. Muchos clones han aumentado y modificado el comportamiento original de `vi`. La mayoría de los clones no soportan todos los comandos originales de `vi`.

Si tiene un buen conocimiento de trabajo con `ed` entonces `vi` presenta una curva de aprendizaje más suave. Aunque usted no tenga ninguna intención de usar `vi` como su editor regular, el conocimiento básico de `vi` solamente puede serle una ventaja.

A.2 Rápido tutorial de Ed

El propósito de esta guía es que usted empiece a usar `ed`. `ed` está diseñado para ser fácil de usar, y requiere poco entrenamiento para comenzar. La mejor manera de aprender es practicando, así que siga las instrucciones y pruebe el editor antes de descontar sus ventajas prácticas.

A.2.1 Crear un fichero

`ed` sólo puede editar un fichero a la vez. Siga el siguiente ejemplo para crear su primer fichero de texto usando `ed`.

```
/home/larry$ ed
a
Este es mi primer fichero de texto usando Ed.
Esto es divertido de verdad.
.
w primero.txt
q
/home/larry$
```

Ahora puede verificar el contenido del fichero usando la utilidad de concatenación de Unix.

```
/home/larry$ cat primero.txt
```

El ejemplo anterior ha mostrado un número de puntos importantes. Cuando invoca `ed` como en el ejemplo tendrá un fichero vacío. La tecla `a` se usa para añadir texto al fichero. Para finalizar la sesión de entrada de texto, se usa un punto `.` en la primera columna del texto. Para salvar el texto a un fichero, la tecla `w` se usa en combinación con el nombre del fichero y finalmente la tecla `q` se usa para salir del editor.

La observación más importante es que hay dos modos de operación. Al principio el editor está en modo de comandos. Un comando se define por caracteres, para asegurarse de cuales son las intenciones del usuario, **ed** usa un **modo de texto**, y un **modo de comando**.

A.2.2 Editar un fichero existente

Para añadir una línea de texto a un fichero existente siga el siguiente ejemplo:

```
/home/larry$ ed primero.txt
a
Esta es una nueva linea de texto.
.
w
q
```

Si comprueba el fichero con **cat** verá que hay una nueva línea insertada entre las dos líneas originales. ¿Cómo supo **ed** dónde poner la nueva línea de texto?.

Cuando **ed** lee el fichero, se acuerda de cual es la línea actual. El comando **a** añadirá el texto después de la línea actual. **ed** también puede poner el texto antes de la línea actual con la tecla **i**. El efecto será la inserción del texto antes de la línea actual.

Ahora es fácil ver que **ed** opera en el texto línea por línea. Todos los comandos se pueden aplicar a una línea elegida.

Para añadir una línea de texto al final de un fichero.

```
/home/larry$ ed primero.txt
$a
La ultima linea de texto.
.
w
q
```

El modificador de comandos **\$** le dice a **ed** que añada la línea después de la última línea. Para añadir la línea después de la primera línea el modificador sería **1**. Ahora tenemos la posibilidad de seleccionar la línea para añadir una línea de texto después del número de línea o insertar una línea antes del número de línea.

¿Cómo sabemos lo que hay en la línea actual? El comando **p** muestra el contenido de la línea actual. Si quiere hacer que la línea actual sea la línea 2, y a la vez quiere ver el contenido de esa línea entonces haga lo siguiente.

```
/home/larry$ ed primero.txt
2p
q
```

A.2.3 Números de línea en detalle

Ha visto cómo mostrar el contenido de la línea actual, usando el comando `[p]`. También sabemos que hay modificadores del número de línea para los comandos. Para ver el texto de la segunda línea,

`2p`

Hay algunos modificadores especiales que se refieren a posiciones que pueden cambiar durante el tiempo que edita un fichero. El `[$]` es la última línea del texto. Para ver la última línea,

`$p`

El número de línea actual usa un simbolo de modificador especial `[.]`. Para mostrar la línea actual usando un modificador,

`.p`

Esto puede parecer innecesario, aunque es muy útil en el contexto de los grupos de números de línea.

Para mostrar el contenido del texto desde la línea 1 a la línea 2, se le debe pasar el alcance.

`1,2p`

El primer número se refiere a la línea de comienzo y el segundo se refiere a la última línea. La línea actual será por consiguiente el segundo número del grupo de líneas.

Si quiere mostrar el contenido del fichero desde el comienzo hasta la línea actual,

`1,.p`

Para mostrar el contenido desde la línea actual hasta el final del fichero,

`.,$p`

Todo lo que queda es mostrar el contenido de todo el fichero, y esto queda como un ejercicio para usted.

¿Cómo puede borrar las dos primeras líneas del fichero?

`1,2d`

El comando `[d]` borra el texto línea por línea. Si usted quisiera borrar el contenido completo, lo haría con:

`1,$d`

Si usted ha hecho muchos cambios y no quiere salvar los contenidos del fichero, lo mejor es salir del editor sin escribir el fichero antes.

La mayoría de los usuarios no usan `ed` como el principal editor. Los editores más modernos ofrecen una pantalla completa de edición y grupos de comandos más flexibles. `ed` ofrece una buena introducción a `vi` y ayuda a explicar cómo se originaron los comandos de `vi`.

A.3 Rápido tutorial de Vi

El propósito de este tutorial es que comience a usar el editor `vi`. Este tutorial asume que no tiene ninguna experiencia con `vi`, así que le mostraremos los diez comandos más básicos de `vi`. Estos comandos fundamentales son suficientes para realizar la mayoría de sus necesidades de edición, y puede expandir su vocabulario de `vi` cuanto necesite. Se recomienda que tenga un ordenador para practicar, según avanza con el tutorial.

A.3.1 Ejecutar `vi`

Para ejecutar `vi`, simplemente tiene que teclear las letras `vi` seguidas del nombre de fichero que desea crear. Verá una pantalla con una columna de tildes (~) en el lado izquierdo. `vi` está ahora en modo de comando. Cualquier cosa que teclee será interpretado como un comando, no como texto que usted desea escribir. Para introducir texto, tiene que teclear un comando. Los dos comandos de entrada básicos son los siguientes:

- i insertar texto a la izquierda del cursor.
- a añadir texto a la derecha del cursor.

Dado que está al comienzo de un fichero vacío, no importa cual de estos usar. Escriba uno de ellos, y después teclee el siguiente texto (un poema de Augustus DeMorgan encontrado en *The Unix Programming Environment* por B.W. Kernighan y R. Pike):

```
Las pulgas grandes tienen peque~nas pulgas<Intro>
    sobre sus espaldas para que les muerdan.<Intro>
Y las pulgas peque~nas tienen pulgas mas peque~nas<Intro>
    y as'i hasta el infinito.<Intro>
Y las pulgas grandes, a su vez,<Intro>
    tienen pulgas mas grandes sobre las que estar;<Intro>
Mientras que estas de nuevo tienen otras mas grandes aun,<Intro>
    y mas grandes aun, y asi.<Intro>
<Esc>
```

Fíjese que tiene que pulsar la tecla Esc para finalizar la inserción y volver al modo de comando.

A.3.2 Comandos de movimiento del cursor

- h** mueve el cursor un espacio a la izquierda.
- j** mueve el cursor un espacio abajo.
- k** mueve el cursor un espacio arriba.
- l** mueve el cursor un espacio a la derecha.

Estos comandos se pueden repetir manteniendo la tecla pulsada. Intente mover el cursor por el texto ahora. Si intenta un movimiento imposible, por ejemplo, pulsar la tecla **k** cuando el cursor está en la línea superior, la pantalla parpadeará momentaneamente o el terminal sonará. No se preocupe, no muerde, y su fichero no saldrá dañado.

A.3.3 Borrar texto

- x** borra el carácter que hay en el cursor.
- dd** borra la línea donde está el cursor.

Mueva el cursor a la primera línea y póngalo de modo que esté bajo la **n**. Pulse la letra **x**, y la **n** desaparecerá. Ahora pulse la letra **i** para cambiarse al modo de inserción y vuelva a teclear la **n**. Pulse **Esc** cuando haya terminado.

A.3.4 Salvar un fichero

- :w** salvar (escribir al disco).
- :q** salir.

Asegúrese de que está en modo de comando pulsando la tecla **Esc**. Ahora pulse **:w**. Esto salvará su trabajo escribiéndolo a un fichero de disco.

El comando para salir de **vi** es **q**. Si quiere combinar el salvar y salir, escriba **:wq**. También hay una abreviación para **:wq** — **ZZ**. Dado que gran parte del trabajo de programación consiste en ejecutar un programa, encontrar algún problema y llamar el programa en el editor para hacer algún pequeño cambio, y luego volver a salir del editor para ejecutar el programa otra vez, **ZZ** será un comando que usará a menudo. (En realidad, **ZZ** no es un sinónimo exacto de **:wq** — si no ha hecho ningún cambio al fichero que está editando desde la última vez que lo salvo, **ZZ** simplemente saldrá del editor mientras que **:wq** salvará (redundantemente) el fichero antes de salir.)

Si usted se lo ha cargado todo sin esperanzas y simplemente quiere volver a empezar todo, puede teclear **:q!** (recuerde pulsar la tecla **Esc** primero). Si omite el **!**, **vi** no le permitirá salir sin salvar.

A.3.5 ¿Qué viene a continuación?

Los diez comandos que acaba de aprender deberían ser suficientes para su trabajo. Sin embargo, solamente ha rozado la superficie del editor **vi**. Hay comandos para copiar material de un lugar del fichero a otro, para mover material de un lugar a otro, para mover material de un fichero a otro, para ajustar el editor a sus gustos personales, etc. Con todo, hay unos 150 comandos.

A.4 Tutorial avanzado de Vi

La ventaja y la potencia de `vi` está en la habilidad de usarlo con éxito con sólo saber unos pocos comandos. La mayoría de los usuarios de `vi` se sienten un poco incómodos al principio, sin embargo después de algún tiempo se encuentran ávidos de mayor conocimiento de los comandos disponibles.

El siguiente tutorial asume que el usuario ha completado el tutorial rápido (arriba) y por lo tanto se siente a gusto con `vi`. Este tutorial expone algunas de las características más poderosas de `ex/vi` desde copiar texto hasta la definición de macros. Hay una sección sobre `ex` y los parámetros para ajustar el editor a su gusto. Este tutorial describe los comandos, en vez de llevarle grupo por grupo. Se recomienda que usted disponga de algún tiempo para probar los comandos en algún texto de ejemplo, que se pueda permitir destruir.

Este tutorial no expone todos los comandos de `vi`, aunque se cubren los más comúnmente usados y aún otros adicionales. Aunque usted elija usar otro editor de texto, se espera que apreciará `vi` y lo que ofrece a quienes deciden usarlo.

A.4.1 Movimiento

La funcionalidad más básica de un editor es el mover el cursor por el texto. Aquí están los comandos de movimiento.

- `h` mueve el cursor un espacio a la izquierda.
- `j` mueve el cursor una línea hacia abajo.
- `k` mueve el cursor una línea hacia arriba.
- `l` mueve el cursor un espacio a la derecha.

Algunas implementaciones también permiten el uso de las teclas de cursor para mover el cursor.

- `w` mueve al principio de la siguiente palabra.
- `e` mueve al final de la siguiente palabra.
- `E` mueve al final de la siguiente palabra antes de un espacio.
- `b` mueve al principio de la palabra anterior.
- `O` mueve al principio de la línea actual.
- `^` mueve a la primera palabra de la línea actual.
- `$` mueve al final de la línea.
- `<CR>` mueve al principio de la siguiente línea.
- `-` mueve al principio de la línea anterior.
- `G` mueve al final del fichero.
- `1G` mueve al principio del fichero.
- `nG` mueve a la línea `n`.
- `<Cntl> G` muestra el número de línea actual.
- `%` va al paréntesis correspondiente.
- `H` mueve a la línea superior en pantalla.
- `M` mueve a la línea de en medio de la pantalla.
- `L` mueve al final de la pantalla.
- `n|` mueve el cursor a la columna `n`.

El texto se desplaza automáticamente cuando el cursor alcanza la parte superior o inferior de la pantalla. También hay comandos que permiten controlar el desplazamiento del texto.

<Cntl> f desplaza una pantalla hacia delante.
<Cntl> b desplaza una pantalla hacia atrás.
<Cntl> d desplaza media pantalla hacia abajo.
<Cntl> u desplaza media pantalla hacia arriba

Los comandos anteriores controlan el movimiento del cursor. Algunos de los comandos usan un modificador de comandos en la forma de un número que precede al comando. Esta característica normalmente repite el comando ese número de veces.

Para mover el cursor ocho posiciones a la izquierda.

8l mueve el cursor 8 posiciones a la izquierda.

Si desea introducir un número de espacios delante de un texto, podría usar el modificador de comandos con el comando insertar. Introduzca el número de repeticiones (n) y después i seguido por el espacio y luego pulse la tecla Esc.

ni inserta algún texto y lo repite n veces.

Los comandos que tratan con líneas usan el modificador para referirse al número de línea. El G es un buen ejemplo.

1G mueve el cursor a la primera línea.

vi tiene un voluminoso grupo de comandos que se puede usar para mover el cursor por el texto. vi también puede posicionar el cursor en una línea desde la línea de comandos.

```
vi +10 mi_fichero.tex
```

Este comando abre el fichero llamado *mi_fichero.tex* y pone el cursor 10 líneas más abajo del comienzo del fichero.

Pruebe los comandos de esta sección. Muy poca gente los recuerda todos en una sola sesión. La mayor parte de los usuarios solamente utiliza una parte de estos comandos.

Ahora se puede mover, pero ¿cómo modificar el texto?

A.4.2 Modificación del texto

El propósito es cambiar el contenido del fichero y vi ofrece un amplio conjunto de comandos para ayudarnos en este proceso.

Esta sección se concentra en añadir texto, cambiar el texto existente y borrar texto. Al final de esta sección usted tendrá el conocimiento para crear cualquier fichero de texto que desee. Las secciones restantes se concentran en comandos más especializados y convenientes.

Cuando usted teclea nuevo texto puede introducir varias líneas usando la tecla Intro. Si hay que corregir un error tipográfico mientras introduce texto, puede usar la tecla Retroceso para mover el cursor sobre el texto. Las varias implementaciones de vi se comportan de manera diferente. Algunas

simplemente mueven el cursor hacia atrás y el texto sigue viéndose y es aceptado. Otras borran el texto cuando pulsa la tecla de borrado. Otras versiones incluso permiten usar las teclas de cursores para mover el cursor mientras se está en modo de entrada de texto. Este modo de funcionamiento no es el normal de vi. Si el texto es visible y usa la tecla `[Esc]` mientras está en la línea en la que pulsó `[Retraceso]`, entonces el texto que hay después del cursor se borrará. Use el editor para acostumbrarse a este comportamiento.

- a Añadir texto a partir de la posición actual del cursor.
- A Añadir al final de la línea.
- i Insertar texto a la izquierda del cursor.
- I Insertar texto a la izquierda del primer carácter que no sea espacio en la línea actual.
- o Abrir una nueva línea y añade texto debajo de la línea actual.
- O Abrir una nueva línea y añade texto encima de la línea actual.

vi tiene un pequeño grupo de comandos para borrar texto que se pueden mejorar con el uso de modificadores.

- x Borrar el carácter que está debajo del cursor.
- dw Borrar desde la posición actual al final de la palabra.
- dd Borrar la línea actual.
- D Borrar desde la posición actual al final de la línea.

Los modificadores se pueden usar para aumentar la potencia de los comandos. Los siguientes ejemplos son un subgrupo de las posibilidades.

- nx Borrar n caracteres desde el que está bajo el cursor.
- ndd Borrar n líneas.
- dnw Borrar n palabras (igual que ndw).
- dG Borrar desde la posición actual hasta el final del fichero.
- d1G Borrar desde la posición actual hasta el principio del fichero.
- d\$ Borrar desde la posición actual al final de la línea. (Esto es igual que D).
- dn\$ Borrar desde la línea actual al final de la enésima línea.

La lista de comandos anterior muestra que la operación de borrado puede ser muy útil. Esto se hace patente cuando se aplica en combinación con los comandos de movimiento de cursor. Hay que hacer notar que el comando `[D]` ignora los modificadores que se le apliquen.

En ocasiones usted puede necesitar deshacer los cambios al texto. Los siguientes comandos recuperan el texto que había antes de hacer cambios.

- u Deshacer el último comando.
- U Deshacer todos los cambios ocurridos en la línea actual.
- :e! Editar otra vez. Recupera el estado del fichero desde la última vez que se salvó.

vi no sólo le permite deshacer cambios, también puede dar la vuelta al comando “deshacer”. Por ejemplo, si usamos el comando `[5dd]` para borrar 5 líneas, se pueden recuperar usando el comando `[u]`. Si usted usa `[u]` una vez más, las 5 líneas desaparecerán de nuevo.

vi ofrece comandos que permiten que se hagan cambios al texto sin tener que borrar y volver a escribir:

rc	Reemplaza el caracter bajo el cursor con la letra “c”. Mueve el cursor a la derecha si se usa el modificador de repetición, por ejemplo: 2rc .
R	Sobreescribe el texto con el nuevo texto..
cw	Cambia el texto de la palabra actual.
c\$	Cambia el texto desde la posición actual al final de la línea.
cnw	Cambia las siguientes n palabras. (Igual que ncw).
cn\$	Hacer cambios hasta el final de la enésima línea.
C	Hacer cambios hasta el final de la línea actual.
cc	Hacer cambios en la línea actual.
s	Sustituye el texto que escriba por el caracter actual.
ns	Sustituye el texto que escriba por los siguientes n caracteres.

Las series de comandos de cambio que permiten que se introduzca un grupo de caracteres se terminan con la tecla **Esc**.

El comando **cw** empieza en la posición actual en la palabra y termina al final de la palabra. Cuando use un comando de cambio que especifica la distancia sobre la que se aplicará el cambio, vi pondrá un **\$** en la última posición de carácter. El nuevo texto puede sobrepasar o no llegar a la longitud original de texto.

A.4.3 Copiar y mover bloques de texto

El movimiento de texto supone la combinación de un número de comandos para conseguir el propósito final. Esta sección introduce los buffers¹ con y sin nombre junto a los comandos que cortan y pegan el texto.

La copia de texto se realiza en tres pasos principales:

1. **Copia** del texto a un buffer.
2. **Movimiento** del cursor al lugar de destino.
3. **Pegar** (poner) el texto en el buffer de edición.

El buffer de edición no es sino el propio fichero en el que usted está trabajando.

Para **Copiar** texto al buffer sin nombre use el comando **y**.

yy	Mueve una copia de la línea actual al buffer sin nombre.
Y	Mueve una copia de la línea actual al buffer sin nombre.
nyy	Mueve las siguientes n líneas al buffer sin nombre.
nY	Mueve las siguientes n líneas al buffer sin nombre.
yw	Mueve una palabra al buffer sin nombre.
ynw	Mueve n palabras al buffer sin nombre.
nyw	Mueve n palabras al buffer sin nombre.
y\$	Mueve el texto desde la posición actual al final de la línea al buffer sin nombre.

¹N. del T.: “buffer” es una palabra inglesa que se refiere a un espacio reservado en memoria para guardar alguna información temporalmente. En este caso se usan buffers para almacenar una copia temporal del texto que usted desea copiar o mover.

El buffer sin nombre es un buffer temporal que puede ser fácilmente afectado por otros comandos comunes. Hay ocasiones en las cuales necesitamos mantener el contenido de un buffer durante un período más extenso de tiempo. En este caso se usaría un buffer con nombre. `vi` tiene 26 buffers con nombre. Los buffers usan las letras del alfabeto como el nombre de identificación. Para distinguir un comando de un buffer con nombre, `vi` usa el carácter `[]`. Cuando use un buffer con nombre de letra minúscula los contenidos se sobre escriben, mientras que el uso del nombre en letra mayúscula añade el nuevo texto al contenido actual.

"`ayy` Mueve la línea actual al buffer con nombre `a`.
"`aY` Mueve la línea actual al buffer con nombre `a`.
"`byw` Mueve la palabra actual al buffer con nombre `b`.
"`Byw` Añade la palabra actual al contenido del buffer `b`.
"`by3w` Mueve las siguientes 3 palabras al buffer `b`.

Use el comando `[]p` para pegar el contenido del buffer al buffer de edición.

`p` Pegar del buffer sin nombre a la DERECHA del cursor.
`P` Pegar del buffer sin nombre a la IZQUIERDA del cursor.
`nP` Pegar `n` copias del buffer sin nombre a la IZQUIERDA del cursor.
"`aP` Pegar del buffer con nombre "`a`" a la DERECHA del cursor.
"`b3P` Pegar 3 copias del buffer con nombre "`b`" a la IZQUIERDA del cursor.

Cuando use `vi` dentro de un `xterm` tiene otra opción para copiar texto. Marque el bloque de texto que desea copiar arrastrando el cursor del ratón sobre el texto. Si mantiene pulsado el botón izquierdo del ratón y arrastra el ratón desde el comienzo al final, invertirá el vídeo del texto. Esta operación automáticamente coloca el texto en un buffer reservado para el servidor de X Window. Para pegar el texto, pulse el botón de enmedio. Acuérdesse de poner `vi` en modo de inserción dado que la entrada de texto se podría interpretar como comandos y el resultado sería cuando menos desagradable. Usando la misma técnica se puede copiar una sola palabra haciendo doble click con el botón izquierdo del ratón sobre la palabra. Solamente se copiará esa única palabra. Para pegarla se usa el mismo procedimiento que anteriormente. Los contenidos del buffer sólo cambian cuando se marca otro bloque de texto.

Para mover un bloque de texto son necesarios tres pasos.

1. **Borrar** el texto para ponerlo en un buffer con o sin nombre.
2. **Mover** el cursor a la posición de destino.
3. **Pegar** el buffer.

El proceso es el mismo que el de copia con un solo cambio en el primer paso para borrar. Cuando se usa el comando `[]dd`, la línea se borra y se coloca en el buffer sin nombre. Luego puede poner el contenido del mismo modo que lo hizo cuando copió el texto en la posición deseada.

"`add` Borrar la línea y ponerla en el buffer con nombre `a`.
"`a4dd` Borrar cuatro líneas y ponerlas en el buffer con nombre `a`.
`dw` Borrar una palabra y ponerla en el buffer sin nombre.

Vea la sección sobre el modificado de texto para ver más ejemplos de borrado de texto.

En el caso de que el sistema falle, el contenido de todos los buffers se pierde excepto el contenido de los buffers de edición, que se puede recuperar (ver comandos Útiles).

A.4.4 Búsqueda y cambio de texto

vi dispone de varios comandos para búsqueda. Usted puede buscar desde un solo carácter hasta una expresión completa.

Los dos comandos principales de búsqueda basados en caracteres son **f** y **t**.

- fc** Encuentra el siguiente caracter c. Se mueve a la derecha del siguiente.
- Fc** Encuentra el siguiente caracter c. Se mueve a la izquierda del anterior.
- tc** Se mueve a la DERECHA al caracter anterior al siguiente c.
- Tc** Se mueve a la IZQUIERDA al caracter que sigue al c anterior. (En algunas versiones de vi esto es lo mismo que Fc).
- ;** Repite el último comando f,F,t,T.
- ,** Igual que (;) pero cambiando la dirección del comando original.

Si el caracter que usted buscaba no se encuentra, vi emitirá un pitido o le dará otro tipo de señal (tal vez un parpadeo de toda la pantalla).

vi le permite buscar una cadena de texto en el buffer de edición.

- /tira** Busca hacia la derecha y abajo la siguiente instancia de “tira”.
- ?tira** Busca hacia la izquierda y arriba la siguiente instancia de “tira”.
- n** Repite el último comando / o ?.
- N** Repite el último comando / o ? en la dirección opuesta.

Cuando use los comandos **/** or **?**, la última línea en la parte baja de la pantalla se vaciará. Introduzca en esa línea la tira de caracteres que desea buscar y pulse **Intro**.

La tira en el comando **/** or **?** puede ser una expresión regular. Una expresión regular puede ser una descripción de un conjunto de tiras. La descripción se construye usando texto entremezclado con caracteres especiales. Los caracteres especiales de una expresión regular son **.** ***** **[** **]** **^** **\$**.

- .** Vale por cualquier carácter excepto el carácter de nueva línea.
- ** Tecla de “escape” para cualquier caracter especial.
- *** Vale por 0 o más instancias del caracter anterior.
- []** Busca exactamente uno de los caracteres incluidos entre los corchetes.
- ^** El caracter que sigue a ^ debe estar al principio de la línea.
- \$** El caracter que precede a \$ se busca al final de la línea.
- [^]** Busca cualquier cosa que no se encuentre despues de ^ dentro de los corchetes.
- [-]** Busca un rango de caracteres.

El único modo de acostumbrarse a usar las expresiones regulares es usándolas. A continuación mostramos varios ejemplos de uso:

c.ch	Coincide con coche, cacho, cochera, etc.
c\ .pe	Coincide con c.pe, c.per, etc.
sto*p	Coincide con stp, stop, stoop, etc.
car.*n	Coincide con cartón, carrillón, carmen, etc.
xyz.*	Coincide con xyz al final de la línea.
^Los	Encuentra cualquier línea que empiece con Los.
abcdef\$	Encuentra cualquier línea que termine con abcdef.
^Solo\$	Coincide con cualquier línea que sólo tenga la palabra “Solo”.
p[aiue]so	Coincide con paso, piso, puso, peso.
Ver[D-F]	Coincide con VerD, VerE, VerF.
Ver[^1-9]	Coincide con Ver siempre que a continuación NO haya un número.
the[ir][re]	Coincide con their, therr, there, theie.
[A-Za-z][A-Za-z]*	Coincide con cualquier palabra.

vi usa el modo de comandos de **ex** para llevar a cabo búsquedas y sustituciones. Todos los comandos que empiezan con dos puntos son comandos en modo **ex**.

Los comandos de búsqueda y sustitución permiten que se usen expresiones regulares sobre un rango de líneas y sustituir la tira de caracteres que se busca. El usuario puede pedir confirmación antes de que se realice la sustitución. Merece la pena hacer una revisión de la representación de los números de línea en el tutorial de **ed**.

:<prim>,<ult>s/<busca>/<sust>/g	Comando general.
:1,\$s/los/Los/g	Buscar el fichero entero y sustituir los por Los.
:%s/the/The/g	% significa el fichero entero (igual que arriba).
:. ,5s/^.*//g	Borra el contenido de la línea actual hasta la quinta.
:%s/los/Los/gc	Sustituye “los” con “Los” preguntando antes de hacer la operación.
:%s/^....//g	Borra los primeros cuatro caracteres de cada línea.

El comando de búsqueda es muy potente cuando se combina con las expresiones regulares. Si no se incluye la directiva **[g]** entonces el cambio sólo se aplica a la primera instancia que se encuentre en cada línea.

En ocasiones usted quiere usar los caracteres de búsqueda en los caracteres de sustitución. Se puede reescribir todo el comando en la línea pero vi permite que los caracteres de sustitución incluyan algunos caracteres especiales.

:1,5s/ayuda/&ndo/g	Sustituye ayuda por ayudando en las cinco primeras líneas.
:%s/ */&&/g	Duplica el número de espacios entre palabras.

El uso de la tira de caracteres completa para búsqueda tiene sus límites y por lo tanto vi usa los paréntesis en secuencia de escape **[(]** y **[)]** para seleccionar el rango de la sustitución. El uso de un dígito en secuencia de escape **[1]** identifica el rango en el orden de la definición y la sustitución se puede hacer.

:s/\^(\.*\):.*\/1/g	Borra todo despues de e incluyendo los dos puntos.
:s/\(\.*\):\(\.*\)/2:1/g	Da la vuelta a las palabras a los dos lados de los dos puntos.

vi ofrece comandos muy potentes que muchos editores de texto modernos no tienen o no pueden

ofrecer. El coste de esta potencia suele ser también el principal argumento en contra de `vi`. Los comandos pueden ser difíciles de aprender y leer. Sin embargo, la mayoría de las cosas buenas suelen ser un poco extrañas al principio. Con un poco de práctica y tiempo, los comandos de `vi` se convertirán en una segunda naturaleza para Ud..

Apéndice B

The GNU General Public License

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation’s software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license

which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 - a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
 - b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

- c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:
 - a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
 - c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary

form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.
6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES

OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does.
Copyright © 19yy *name of author*

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139,

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright © 19yy *name of author*
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it under certain conditions;
type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’
(which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Apéndice C

The GNU Library General Public License

GNU LIBRARY GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright © 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the library GPL. It is numbered 2 because it goes with version 2 of the ordinary GPL.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users.

This license, the Library General Public License, applies to some specially designated Free Software Foundation software, and to any other libraries whose authors decide to use it. You can use it for your libraries, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library, or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link a program with the library, you must provide complete object files to the recipients so that they can relink them with the library, after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

Our method of protecting your rights has two steps: (1) copyright the library, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the library.

Also, for each distributor's protection, we want to make certain that everyone understands that there is no warranty for this free library. If the library is modified by someone else and passed on, we want its recipients to know that what they have is not the original version, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that companies distributing free software will individually obtain patent licenses, thus in effect transforming the program into proprietary software. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License, which was designed for utility programs. This license, the GNU Library General Public License, applies to certain designated libraries. This license is quite different from the ordinary one; be sure to read it in full, and don't assume that anything in it is the same as in the ordinary license.

The reason we have a separate public license for some libraries is that they blur the distinction we usually make between modifying or adding to a program and simply using it. Linking a program with a library, without changing the library, is in some sense simply using the library, and is analogous to running a utility program or application program. However, in a textual and legal sense, the linked executable is a combined work, a derivative of the original library, and the ordinary General Public License treats it as such.

Because of this blurred distinction, using the ordinary General Public License for libraries did not effectively promote software sharing, because most developers did not use the libraries. We concluded that weaker conditions might promote sharing better.

However, unrestricted linking of non-free programs would deprive the users of those programs of all benefit from the free status of the libraries themselves. This Library General Public License is intended to permit developers of non-free programs to use free libraries, while preserving your freedom as a user of such programs to change the free libraries that are incorporated in them. (We have not seen how to achieve this as regards changes in header files, but we have achieved it as regards changes in the actual functions of the Library.) The hope is that this will lead to faster development of free libraries.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, while the latter only works together with the library.

Note that it is possible for a library to be covered by the ordinary General Public License rather than by this special one.

Terms and Conditions for Copying, Distribution and Modification

0. This License Agreement applies to any software library which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Library General Public License (also called “this License”). Each licensee is addressed as “you”.

A “library” means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The “Library”, below, refers to any such software library or work which has been distributed under these terms. A “work based on the Library” means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term “modification”.)

“Source code” for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library’s complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 - a. The modified work must itself be a software library.
 - b. You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
 - c. You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

- d. If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement

to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a “work that uses the Library”. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a “work that uses the Library” with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a “work that uses the library”. The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a “work that uses the Library” uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also compile or link a “work that uses the Library” with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer’s own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- a. Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable “work that uses the Library”, as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
- b. Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost

of performing this distribution.

- c. If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
- d. Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

- 7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:
 - a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
 - b. Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.
- 8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
- 9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.
- 10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further

restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Library General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of

preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the library's name and a brief idea of what it does.

Copyright (C) *year* name of author

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details.

You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library ‘Frob’ (a library for tweaking knobs) written by James Random Hacker.

signature of Ty Coon, 1 April 1990

Ty Coon, President of Vice

That’s all there is to it!

Apéndice D

¿Qué es LuCAS?

El proyecto LuCAS (Linux en CASTellano) comenzó, como tantas otras cosas ligadas a Linux, en un bar tomando cervezas (reales, no virtuales). Era allá por marzo de 1995 y ya se había comentado en las áreas de FidoNet la necesidad de disponer de textos en castellano sobre Linux. Fue en aquel bar, el bar Ekin de Algorta (Bizkaia), donde nos reuníamos los usuarios de FidoNet todos los domingos, donde surgió la idea de pasar a la acción. Tras discutirlo un poco, decidimos comenzar por traducir algo que fuese útil para aquellos que comenzaban a dar sus primeros pasos en linux, y creímos que el “Installation & Getting Started” de Matt Welsh era el documento más adecuado. Como era mucho trabajo para una sola persona, y había mucha gente que se había mostrado dispuesta a colaborar, había que hacerlo bien, y el trabajo de traducir un libro como éste requería una labor de coordinación que inmediatamente asumimos Ramón Gutiérrez y yo, Alfonso Belloso. Yo me hice cargo de coordinar a los traductores para que no hiciesen trabajo repetido, y Ramón se encargaba de juntar los trabajos y generar el documento final. Pedimos permiso al propio Matt Welsh, y así echamos a andar.

Hubo que esperar un largo tiempo a tener una primera versión *presentable* del libro, pero se consiguió. Fundamentalmente, el hecho de que era una experiencia nueva, y la diversidad de gente que colaboró, hizo que este primer paso durase casi un año, pero mereció la pena esperar. Hay que decir que a pesar de que todo este movimiento surgió de FidoNet, y que desde allí se sigue muy de cerca, al poco de comenzar a trabajar en ello fue cuando conseguí mi primera cuenta de Internet, y comprobé que la forma más práctica de coordinarnos era a través de La Red, ya que por Fidonet las comunicaciones se hacían demasiado largas en el tiempo y no siempre eran seguras (para contactar con algún colaborador a veces se podía tardar 2 semanas, o no saber nada de él jamás). Y con ello comenzamos a disponer de más herramientas: primero un servidor de `ftp`, luego una lista de correo, un servidor de Web, y así hasta hoy. Bueno, pues una vez terminado este primer libro nos quedó el gusanillo de hacer más, y visto que yo acabé un poco cansado, vino Juan José Amor, me tomó el relevo y comenzamos a traducir la “Network Administrator Guide” de Olaf Kirch. Esta vez, y con la infraestructura ya montada, tardamos unos pocos meses en tener la primera versión.

Y después de éste ya se están haciendo más. Ya esta terminada la traducción de la “Linux User Guide” (en su fase alfa), y hay algún proyecto más en marcha. No quiero dejar de mencionar otros proyectos que trabajan en paralelo a éste. LuCAS en sí se ha ido dedicando más a la labor de traduc-

ción de textos tipo libro, más concretamente a los del LDP, y creaciones propias, si bien en su origen se ideó para traducir cualquier tipo de documento e incluso para crear documentos directamente en castellano. El SLUG (Spanish Linux User Group) ya existía cuando surgió LuCAS, o acababa de aparecer. Al amparo de LuCAS surgió el INSFLUG, que se centra en la traducción/creación de FAQs, HOWTOs y documentos más pequeños. Y probablemente existan más grupos haciendo tareas similares, pero que ahora no conozco, ni soy capaz de recordar.

Para contactar con los coordinadores del grupo LuCAS, puede hacerlo con Alfonso Belloso Martínez <<mailto:alfon@bipv02.bi.ehu.es>> o en <<mailto:alfon@iies.es>>; y con Juan Jose Amor Iglesias <jjamor@ls.fi.upm.es>.

Apéndice E

¿Qué es INSFLUG?

El INSFLUG nació allá por marzo del '96, siendo su seno materno el área `echomail` de FidoNet R34.LINUX. Surge ante el entusiasmo de los que empezábamos por aquel entonces a adentrarnos en el mundo del Linux, maravillados por como un SO tan potente, de distribución libre, podía haberse hecho realidad gracias al buen hacer, generosidad e increíble colaboración entre tantas personas "dispersas" por el globo...

Nos sentimos aún más maravillados al observar la increíble participación de la gente que frecuentaba el área, modélica en FidoNet, y por la inestimable ayuda que allí se prestaba; eso fue, —haciendo un poco de psicólogo— lo que produjo la formación del INSFLUG, un inmenso y sincero agradecimiento, además de enormes ganas de corresponder por nuestra parte, aportando nuestro granito de arena a la comunidad Linux en forma de traducciones de Howtos y FAQs, documentos que generalmente no superan la decena de páginas, y con cuya traducción además de aportar, el nóvel o no tan nóvel, profundiza en el tema del que trate el Howto a medida que lo traduce, redundando en beneficio para él mismo...;-)

El INSFLUG se centra en la traducción de Howtos y FAQs (Frequently Asked Questions, algo así como Preguntas de Uso Frecuente, PUFs).

Para la traducción no necesitas más que un editor de texto (En Linux hay muchos) normal, y el paquete `linuxdoc-sgml`, con el que empleando un lenguaje muy sencillo, similar al HTML, una vez escrito un sólo fuente, se podrán obtener a partir de él el documento en formatos `*.dvi`, `*.ps`, `*.txt`, `*.lyx`, `*.rtf`, `*.info`, `*.html`, etc..

Cuentas además con una guía traducida del uso del `linuxdoc-sgml` en la que se añaden detalles de configuración para el castellano, además de la facilidad que supone el traducir editando el fuente original en inglés, ya que los codigos, etc., ya están; no obstante, es un lenguaje sencillísimo, además de ser el formato oficial de todos los Howtos.

El INSFLUG colabora estrechamente con LuCAS, es más, hay varias personas que colaboran con ambos grupos. El grupo LuCAS se dedica mayormente a las guías ("guides"), los libros más extensos del LDP o (Linux Documentation Project).

El coordinador de INSFLUG es Francisco J. Montilla (email: [<mailto:pacopepe@iname.com>](mailto:pacopepe@iname.com);

irc: pukka; FiDO: 2:345/402.22). Si desea más información puede consultar su página en WWW en [<http://www.insflug.nova.es>](http://www.insflug.nova.es) y su espacio de FTP en [<ftp://ftp.insflug.nova.es>](ftp://ftp.insflug.nova.es).

Bibliografía

- [1] Almesberger, Werner. *LILO: Generic Boot Loader for Linux*. Disponible electrónicamente: `tsx-11.mit.edu`. 3 de Julio de 1993.
- [2] Bach, Maurice J. *The Design of the UNIX Operating System*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc. 1986.
- [3] Lamport, Leslie. *LaTeX: A Document Preparation System*. Reading, Massachusetts: Addison-Wesley Publishing Company. 1986.
- [4] Stallman, Richard M. *GNU Emacs Manual*, octava edición. Cambridge, Massachusetts: Free Software Foundation. 1993.