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.

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 (Frecuently 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*>;

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 asi 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 traduccion 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-

> 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!

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

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

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.

Apéndice C

The GNU Library General Public License

GNU LIBRARY GENERAL PUBLIC LICENSE
Version 2, June 1991

[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.

> 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.

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

   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

Apéndice B

The GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

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

`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`	% siginifica 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 carácteres 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

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 caracter `"`. 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. Acuérdese 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 ateriormente. 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.

simplemenente 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 Retroceso, 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 caracter 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 caracter 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:

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.
`0`	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`.

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 basicos 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.

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 despues 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
```

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, incluído 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 porqué 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`.

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 existente[3], 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!

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*.

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

---

[14] N.T.: dump

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**.

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 evaluan (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 se. 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 salida[9] 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 memoria[10] 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
```

---

[9] N.T.: to pipe

[10] N.T.: core file

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.

---

[8] N.T.: timestamp

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 señalizarán 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 rígidos[2]". Si el mundo fuera perfecto, todos los directorios serían referenciados por cada uno de sus subdirectorios (a causa de la opción `..`), 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 distribuidos[3] 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 dispositivo[4], es decir, no cambiarse de sistema de ficheros. Es muy útil cuando se tiene que buscar algo en el

---

[2] N.T. hard link count

[3] Los sistemas de fichero distribuidos permiten a los ficheros mostrarse como si fueran locales a una máquina cuando realmente están situados en algún otro lugar.

[4] N.T.: to cross device

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 hackers[1]. 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

---

[1] N.T.: La palabra Hacker en este contexto se refiere a una persona con profundos conocimientos de informática.

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, despues 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 del 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 **f**ile **t**ransfer **p**rotocol (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

`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 provinientes 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 origenes 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 caracteristica 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

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]
```

```
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 explicitamente 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 incluída 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.

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.

[Companñeros: 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	Leido 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@
```

```
"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:
```

```
  "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
```

```
# 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
{
```

---

[5] Éste 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.

```
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 espero[3]), 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	Leido por el gestor de X Window, `twm`.
.fvwmrc	Leido 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 servidor[4] de como interpretar las señales del teclado.

---

[3] N. del T.: Yo también ;-)

[4] El “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

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 apóstrofos agudos. Estos apóstrofos 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 apóstrofos 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`'>'
```

Diseccionemos 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 apóstrofos graves, y el resultado se guarda en lugar del nombre del comando entrecomillado.

Figura 9.2 Algunas variables de entorno importantes.

Nombre	Contenido	Ejemplo
`HOME`	Directorio principal del usuario	**/home/larry**
`TERM`	Tipo de terminal del usuario	`xterm`, `vt100`, o `console`
`SHELL`	Path del shell del usuario	**/bin/bash**
`USER`	Nombre de ingreso	`larry`
`PATH`	Lista de búsqueda de programas	**/bin:/usr/bin:/usr/local/bin:/usr/bin/X11**

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 carácteres 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 ¿como 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 tambien 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$
```

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---quiza quiera echarles un vistazo
alias ds=dirs             # en la pagina 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"         # correcion ortografica!
alias moer="more"
alias email="emacs -f rmail" # mi lector de correo
alias ed2="emacs -d floss:0 -fg \"grey95\" -bg \"grey50\"" # asi 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

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 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!.

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 libreria 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-

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. Prueber 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:

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 delante)? 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, arréglelo, 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 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 .

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

---

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

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 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 Capitulo 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.

---

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

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:

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

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 Apendice 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`”.)

“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`”

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 nteresantes. 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 nteresantes. 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 comparando 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*

`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`.

---

[13] N. del T.: Abreviatura de **g**eneralized **r**egular **e**xpression **p**arser, analizador general de expresiones regulares.

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 carácteres de control como carácteres 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`:

---

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

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 **u**suario, **g**rupo, **o**tro o todos (**a**ll) 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 (**r**ead), escritura (**w**rite) , y ejecución (e**x**ecute).

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.

---

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

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

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 Unix[1] 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 conjuntamente[2].

---

[1] N. 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)

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

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

`fg` *%nºtrabajo*	Este es un comando del shell que devuelve un trabajo al primer plano. Para saber cuál es éste por defecto, se teclea "`jobs`" y se busca el que tiene el `+`. Parámetros: número de trabajo opcional. El trabajo por defecto se identifica con el `+`.
`&`	Cuando se añade un `&` 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.
`bg` *%nºtrabajo*	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 "`jobs`" y se busca el que tiene el `+`. Parámetros: número de trabajo opcional. El trabajo por defecto se identifica con el `+`.
`kill` *%nºtrabajo PID*	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 `%`. Parámetros: El número de trabajo (a continuación del `%`) o el PID (no es necesario el `%`). Se puede especificar más de un proceso o trabajo en una línea.
`jobs`	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.
Ctrl-c	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.
Ctrl-z	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.

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 `PID`s. El comando `ps` lista todos los procesos en ejecución, incluyendo el shell. Pruébelo. 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 ellos[7]. 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 `execl`. 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

---

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

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

[9] N.T.: No he creido 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.

```
/home/larry$ yes | uniq > /dev/null
[y 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, despues 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:

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$
```

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 con**cat**enar 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

---

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

[4] Se llama `more` porque ese es el indicador que originalmente mostraba: "`--more--`". En varias versiones de Linux el comando `more` es identico 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`".

6.2 Ganar tiempo con `bash`

6.2.1 Editando la línea de comandos

A veces, escribe 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 carácteres adyacentes[1]. 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 pude 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 carácteres 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`"

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

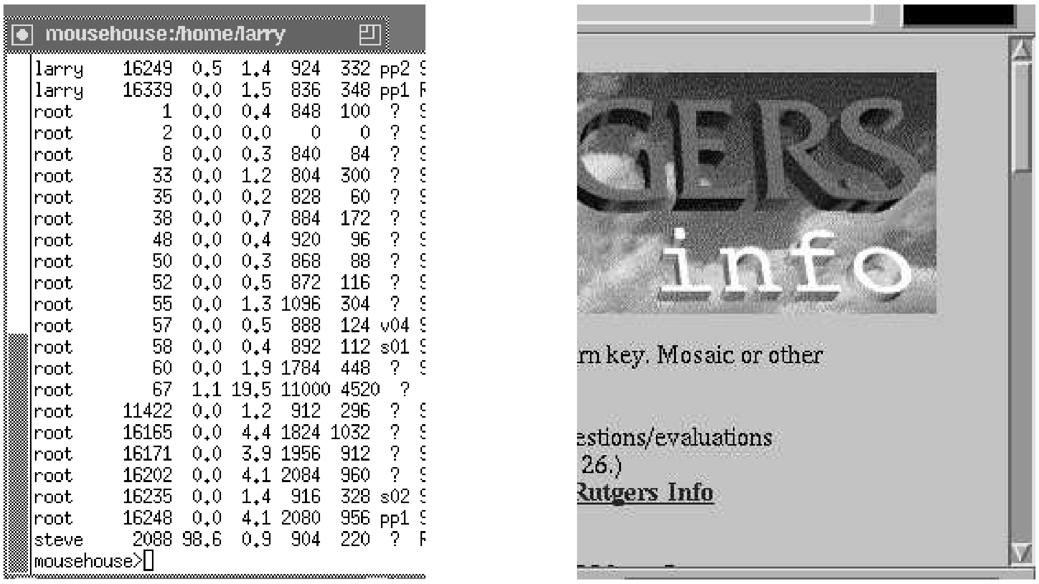
En el capítulo anterior, se enseñaron los comandos para mantener ficheros `cp`, `mv`, y `rm`. A veces, se querrá tratar con más de un fichero a la vez—en realidad, con muchos a la vez. Por ejemplo, se quiere copiar todos los ficheros que empiecen por **data** en un directorio llamado /**backup**. Se 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?

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`.



```
larry    16249  0.5  1.4  924  332 pp2
larry    16339  0.0  1.5  836  348 pp1
root         1  0.0  0.4  848  100  ?
root         2  0.0  0.0    0    0  ?
root         8  0.0  0.3  840   84  ?
root        33  0.0  1.2  804  300  ?
root        35  0.0  0.2  828   60  ?
root        38  0.0  0.7  884  172  ?
root        48  0.0  0.4  920   96  ?
root        50  0.0  0.3  868   88  ?
root        52  0.0  0.5  872  116  ?
root        55  0.0  1.3 1096  304  ?
root        57  0.0  0.5  888  124 v04
root        58  0.0  0.4  892  112 s01
root        60  0.0  1.9 1784  448  ?
root        67  1.1 19.5 11000 4520  ?
root     11422  0.0  1.2  912  296  ?
root     16165  0.0  4.4 1824 1032  ?
root     16171  0.0  3.9 1956  912  ?
root     16202  0.0  4.1 2084  960  ?
root     16235  0.0  1.4  916  328 s02
root     16248  0.0  4.1 2080  956 pp1
steve     2088 98.6  0.9  904  220  ?
mousehouse>
```

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

Barras de desplazamiento Motif

Una barra Motif actua 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 cual 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.

Figura 5.2 Opciones Estandar para Programas X

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

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 chismes[13]". 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 inmediantamente) el botón izquierdo. Los botones Athena y Motif son funcionalmente equivalentes pero tienen diferencias de apariencia.

---

[12] N.T. DISPLAY significa presentación en inglés.

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

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:

---

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

colocarla en la posición que prefiera en la pantalla. Este modo se llama **colcación 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 situe. 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 carácteres 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 botones[10] (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 $\boxed{\text{Alt}}$ y muevo el cursor sobre una ventana, puedo arrastrarla usando el boton izquierdo de mi ratón.

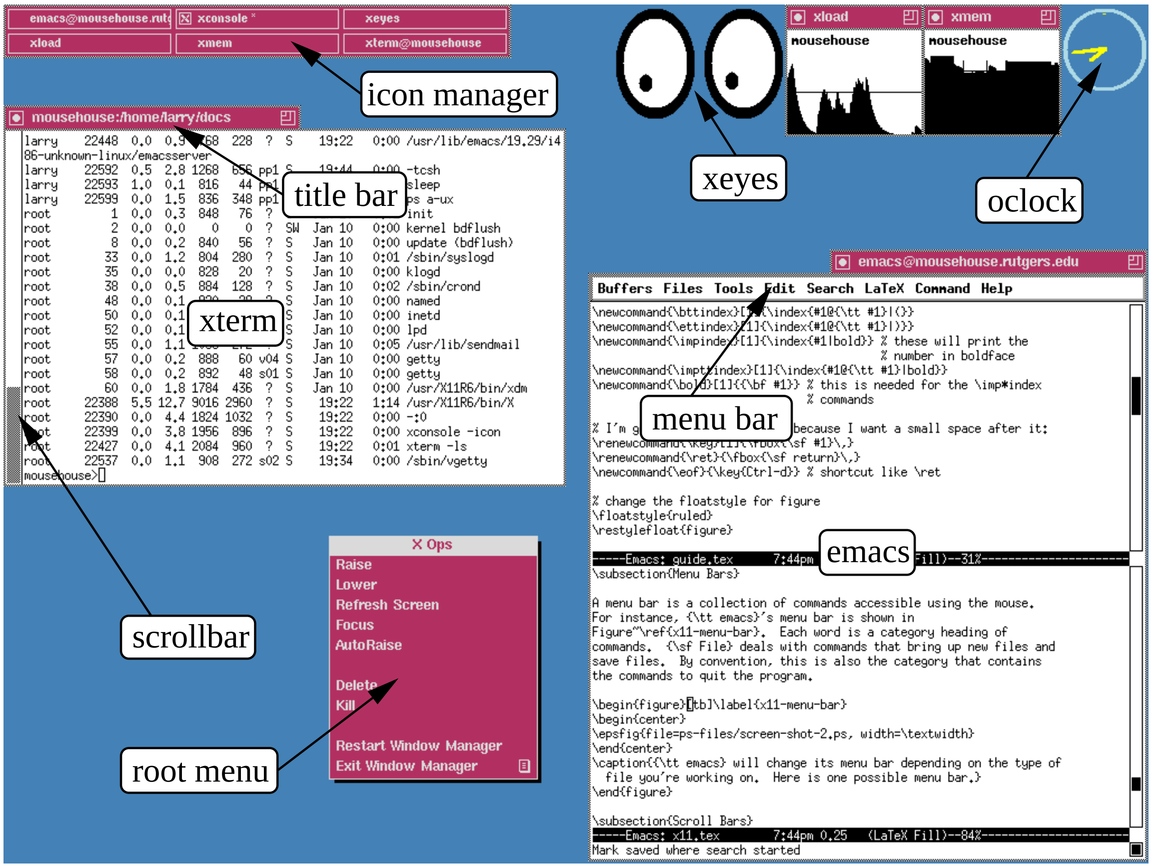
Por supuesto, para ver como funciona en su caso puede probar igual que antes o referirse a la documentación local. Alternativamente, si quiere tratar de interpretar los ficheros de configuración de los gestores de ventanas puede referirse a la Sección 9.2.1 para `twm` o 9.2.2 para `fvwm`.

---

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

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

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



```
larry    22448  0.0  0.9  68  228  ?  S    19:22  0:00 /usr/lib/emacs/19.29/i4
86-unknown-linux/emacsserver
larry    22592  0.5  2.8 1268  656 pp1 S   19:44  0:00 -tcsh
larry    22593  1.0  0.1  816   44 pp1              sleep
larry    22599  0.0  1.5  836  348 pp1              ps a-ux
root         1  0.0  0.3  848   76  ?               init
root         2  0.0  0.0    0    0  ?  SW  Jan 10  0:00 kernel bdflush
root         8  0.0  0.2  840   56  ?  S   Jan 10  0:00 update (bdflush)
root        33  0.0  1.2  804  280  ?  S   Jan 10  0:01 /sbin/syslogd
root        35  0.0  0.0  828   20  ?  S   Jan 10  0:00 klogd
root        38  0.0  0.5  884  128  ?  S   Jan 10  0:02 /sbin/crond
root        48  0.0  0.1               S   Jan 10  0:00 named
root        50  0.0  0.1               S   Jan 10  0:00 inetd
root        52  0.0  0.1               S   Jan 10  0:00 lpd
root        55  0.0  1.1               S   Jan 10  0:05 /usr/lib/sendmail
root        57  0.0  0.2  888   60 v04 S   Jan 10  0:00 getty
root        58  0.0  0.2  892   48 s01 S   Jan 10  0:00 getty
root        60  0.0  1.8 1784  436  ?  S   Jan 10  0:00 /usr/X11R6/bin/xdm
root     22388  5.5 12.7 9016 2960  ?  S    19:22  1:14 /usr/X11R6/bin/X
root     22390  0.0  4.4 1824 1032  ?  S    19:22  0:00 -:0
root     22399  0.0  3.8 1956  896  ?  S    19:22  0:00 xconsole -icon
root     22427  0.0  4.1 2084  960  ?  S    19:22  0:01 xterm -ls
root     22537  0.0  1.1  908  272 s02 S    19:34  0:00 /sbin/vgetty
mousehouse>
```

```
\newcommand{\bttindex}[1]{\index{#1@{\tt #1}|(}}
\newcommand{\ettindex}[1]{\index{#1@{\tt #1}|)}}
\newcommand{\impindex}[1]{\index{#1|bold}} % these will print the
                                           % number in boldface
\newcommand{\impttindex}[1]{\index{#1@{\tt #1}|bold}}
\newcommand{\bold}[1]{{\bf #1}} % this is needed for the \imp*index
                                % commands

% I'm g                       because I want a small space after it:
\renewcommand{\key}[1]{\fbox{\sf #1}\,}
\renewcommand{\ret}{\fbox{\sf return}\,}
\newcommand{\eof}{\key{Ctrl-d}} % shortcut like \ret

% change the floatstyle for figure
\floatstyle{ruled}
\restylefloat{figure}

-----Emacs: guide.tex      7:44pm              Fill)--31%--------------------
\subsection{Menu Bars}

A menu bar is a collection of commands accessible using the mouse.
For instance, {\tt emacs}'s menu bar is shown in
Figure~\ref{x11-menu-bar}.  Each word is a category heading of
commands.  {\sf File} deals with commands that bring up new files and
save files.  By convention, this is also the category that contains
the commands to quit the program.

\begin{figure}[tb]\label{x11-menu-bar}
\begin{center}
\epsfig{file=ps-files/screen-shot-2.ps, width=\textwidth}
\end{center}
\caption{{\tt emacs} will change its menu bar depending on the type of
  file you're working on.  Here is one possible menu bar.}
\end{figure}

\subsection{Scroll Bars}
-----Emacs: x11.tex      7:44pm 0.25   (LaTeX Fill)--84%--------------------
Mark saved where search started
```

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 exáctamente 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`”

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.

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:

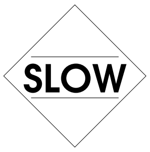
```
/home/larry$ ls -F
frog         passwd        passwd_version/
/home/larry$ rm frog toad passwd
rm: toad: No such file or directory
/home/larry$ ls -F
passwd_version/
/home/larry$
```

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
```

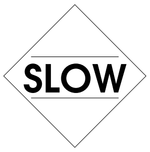
Aprenderemos más acerca de la creación y edición de archivos en los próximos capítulos.

Los comandos básicos para manipular archivos en Unix son `cp`, `mv`, y `rm`. Sus nombres provienen de **c**o**p**y[22], **m**o**v**e[23], y **r**e**m**ove[24], respectivamente.

4.4.1 `cp` como un monje

`cp` [-i] *fuente 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 sobreescribir 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**.

---

[22] N. del T.: Copiar.

[23] N. del T.: Mover.

[24] N. 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.

[26] N. del T.: Disco lleno.

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 **mak**e **dir**ectory[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.

---

[19] N. del T.: Crear directorio.

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 más directorios, y dentro de ellos hay archivos, y aún más directorios[12].

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 **list**ar 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 /
```

---

[12] Puede 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.)

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

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

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

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-file[7], o EOF, para abreviar. Puede que en ciertos libros de texto aparezca como end-of-text[8]. 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 dado[9]. Por ejemplo:

```
/home/larry$ man cat

cat(1)                                                    cat(1)

NAME
```

---

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

[7] N. del T.: fin de archivo.

[8] N. 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.

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".

---

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

[2] La 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.

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

```
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 4[22]. 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**rinter[23], 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.)

---

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

[23] N. del T.: Impresora de líneas.

```
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 interior[16].

   ```
   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 inglés[17].

---

[15] N. 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 incluída en el programa.

[17] N. del T.: "colour" es la variante ortográfica del inglés insular y "color" la norteamericana.

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

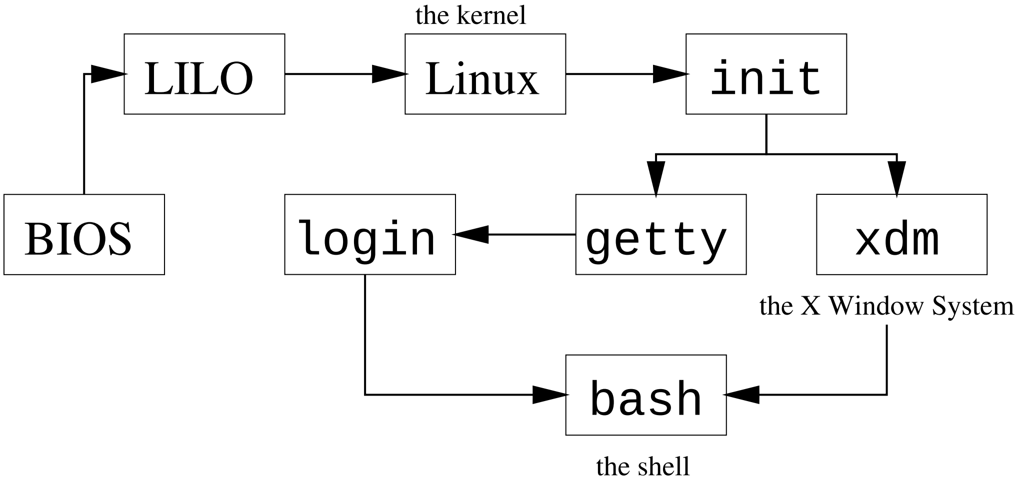
---

[9] N. 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. .

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

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

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.

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 identificarla[2]. 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 arranque[3].

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 saltearse 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

---

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

[2] Desde 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.

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

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 errores[6] que afectan a todos los usuarios son escasos y no se dan seguido.

Por supuesto, ésos 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

---

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

[6] N. del T.: *error* del inglés *bug*.

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 supliera 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 dió 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 original[1]. 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.

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 programas[9]. 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.

---

[9] Este 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 despúes sigue usándose.

[10] N. del T.: *núcleo* del inglés "*kernel*".

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 X[2]. 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 Mundial[3].

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 experimentados[4].

---

[2] N. del T.: X Window System.

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

[4] N. 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.

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

Índice General

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.

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.