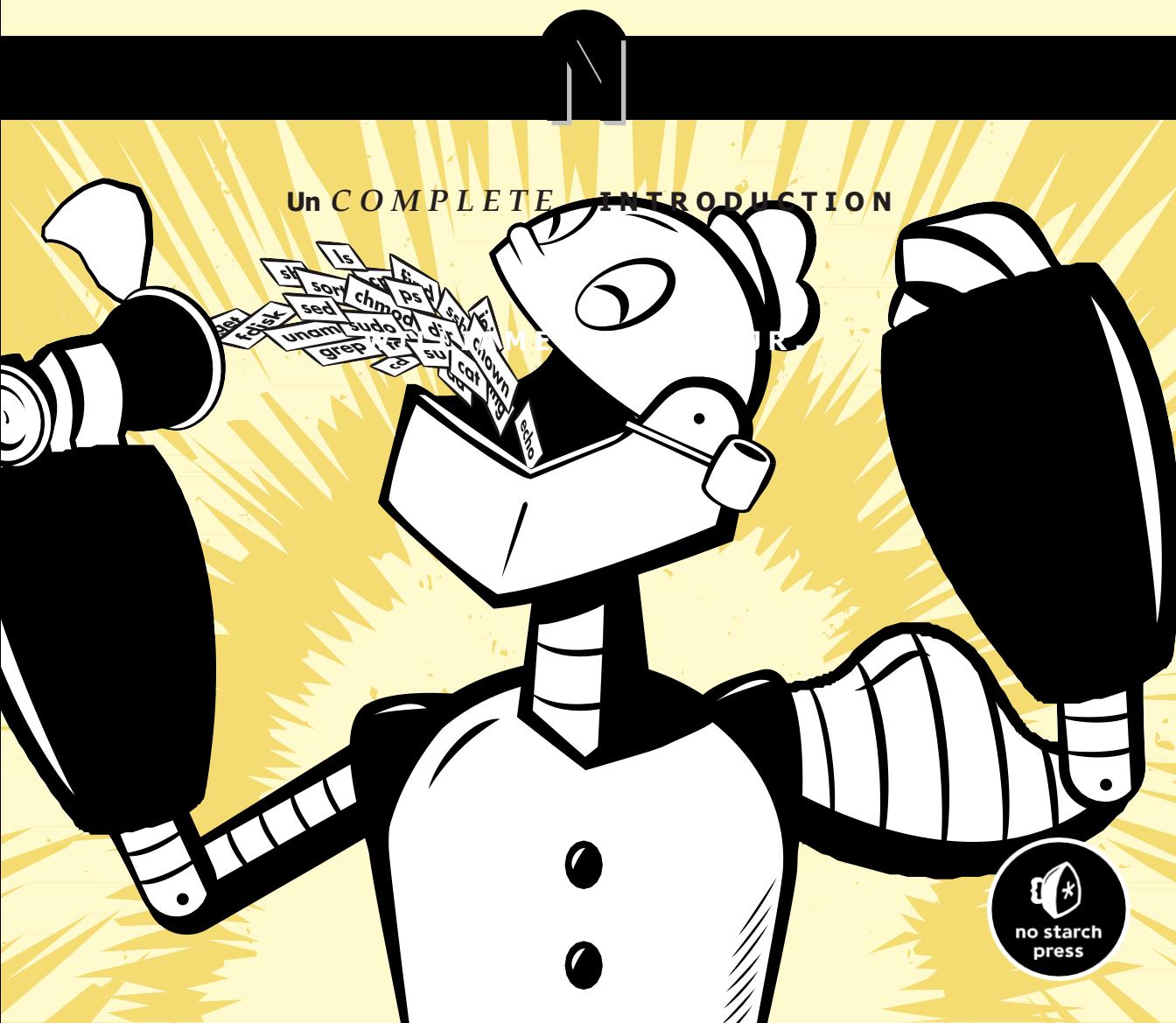


# EL LINUX LÍNEA DE COMUNICACIÓ

Un COMPLETE INTRODUCTION





## **ELOGIOS PARA LA LÍNEA DE COMANDOS DE LINUX**

"Puedo decir honestamente que he encontrado LA guía para principiantes de Linux."

—REVISTA DE LINUX

"El tomo más accesible sobre el tema".

—REVISTA LINUX

"Cualquiera que lea este libro y haga uso de los ejemplos proporcionados no podrá evitar convertirse en un profesional de la línea de comandos de UNIX para cuando haya llegado al final del libro".

—ITWORLD

"La guía ideal para el mundo de la línea de comandos de Linux (y UNIX y BSD)."

—DISTROWATCH.COM

"Si desea comenzar a usar la línea de comandos, mejorar sus habilidades existentes o simplemente desea descubrir herramientas que ni siquiera sabía que existían, este libro tiene todo lo que necesita y lo recomiendo totalmente".

—PHIL BULL, AUTOR DE LA DOCUMENTACIÓN OFICIAL DE UBUNTU

"Esta es la mejor introducción a la línea de comandos que he leído".

—BEGINLINUX.COM



# **EL LINUX LÍNEA DE COMANDOS**

**Una introducción  
completa**

**por William E. Shotts, Jr.**



San Francisco

**LA LÍNEA DE COMANDOS DE LINUX.** Derechos de autor © 2012 por William E. Shotts, Jr.

Todos los derechos reservados. Ninguna parte de este trabajo puede ser reproducida o transmitida en ninguna forma o por ningún medio, electrónico o mecánico, incluyendo fotocopias, grabaciones, o por cualquier sistema de almacenamiento o recuperación de información, sin el permiso previo por escrito del propietario de los derechos de autor y el editor.

Octava impresión

17 16 15      8 9 10

ISBN-10: 1-59327-389-4

ISBN-13: 978-1-59327-389-7

Editor: William Pollock  
Editor de producción: Serena Yang  
Diseño de portada: Octopod Studios  
Editor de desarrollo: Keith Fancher  
Revisor técnico: Therese Bao  
Corrector de estilo: Ward Webber

Compositores: Serena Yang y Alison Law  
Correctora: Paula L. Fleming

Para obtener información sobre distribuidores de libros o traducciones, comuníquese

directamente con No Starch Press, Inc.: No Starch Press, Inc.  
38 Ringold Street, San Francisco, CA 94103  
teléfono: 415.863.9900; info@nostarch.com; www.nostarch.com

*Datos de catalogación en publicación de la Biblioteca del Congreso*

Shotts, William E.

La línea de comandos de Linux: una introducción completa / William E. Shotts, Jr.  
p. cm.

Incluye índice.

ISBN-13: 978-1-59327-389-7 (pbk.)

ISBN-10: 1-59327-389-4 (pbk.)

1. Linux. 2. Lenguajes de scripting (informática) 3. Sistemas operativos (computadoras) I. Título.

QA76.76.0635556 2011

005.4'32--dc23

2011029198

No Starch Press y el logotipo de No Starch Press son marcas comerciales registradas de No Starch Press, Inc. Otros nombres de productos y empresas mencionados en este documento pueden ser marcas comerciales de sus respectivos propietarios. En lugar de utilizar un símbolo de marca comercial con cada aparición de un nombre de marca comercial, estamos utilizando los nombres solo de manera editorial y en beneficio del propietario de la marca comercial, sin intención de infringir la marca comercial.

La información de este libro se distribuye "tal cual", sin garantía. Si bien se han tomado todas las precauciones en la preparación de este trabajo, ni el autor ni No Starch Press, Inc. tendrán ninguna responsabilidad ante ninguna persona o entidad con respecto a cualquier pérdida o daño causado o presuntamente causado directa o indirectamente por la información contenida en él.

A Karen



# BRIEF CONTENTS

Reconocimientos.....Xxiii

Introducción.....Xv

## PARTE 1: APRENDIENDO EL CAPARAZÓN

Capítulo 1: ¿Qué es la cáscara? ..... 3

Capítulo 2: Navegación. ..... 7

Capítulo 3: Explorando el sistema..... 13

Capítulo 4: Manipulación de archivos y directorios ..... 25

Capítulo 5: Trabajar con comandos ..... 39

Capítulo 6: Redirección. ..... 49

Capítulo 7: Ver el mundo como lo ve el caparazón..... 59

Capítulo 8: Trucos avanzados de teclado. ..... 69

Capítulo 9: Permisos. ..... 77

Capítulo 10: Procesos ..... 95

## PARTE 2: LA CONFIGURACIÓN Y EL ENTORNO

Capítulo 11: El Medio Ambiente ..... 109

Capítulo 12: Una Suave Introducción a vi ..... 121

Capítulo 13: Personalización del mensaje..... 139

## PARTE 3: TAREAS COMUNES Y HERRAMIENTAS ESENCIALES

Capítulo 14: Gestión de paquetes. ..... 149

Capítulo 15: Medios de almacenamiento ..... 159

Capítulo 16: Redes..... 175

Capítulo 17: Búsqueda de archivos .....	187
Capítulo 18: Archivado y copia de seguridad .....	201
Capítulo 19: Expresiones regulares. ....	215
Capítulo 20: Procesamiento de textos.....	233
Capítulo 21: Formateo de la salida.....	267
Capítulo 22: Imprenta. ....	285
Capítulo 23: Compilación de programas .....	297

#### **PARTE 4: ESCRIBIR SCRIPTS DE SHELL**

Capítulo 24: Escribir tu primer guión. ....	309
Capítulo 25: Iniciar un proyecto.....	315
Capítulo 26: Diseño de arriba hacia abajo.....	325
Capítulo 27: Control de flujo: Ramificación con if. ....	333
Capítulo 28: Lectura de la entrada del teclado. ....	347
Capítulo 29: Control de flujo: Bucle con while y until.....	357
Capítulo 30: Solución de problemas.....	363
Capítulo 31: Control de flujo: Ramificación con caja. ....	375
Capítulo 32: Parámetros posicionales. ....	381
Capítulo 33: Control de flujo: Bucle con for .....	393
Capítulo 34: Cadenas y números. ....	399
Capítulo 35: Matrices.....	415
Capítulo 36: Exótico.....	423
Índice.....	433

# CONTENIDOS EN DETALLE

## RECONOCIMIENTOS

Xxiii

## INTRODUCCIÓN

Xxv

¿Por qué usar la línea de comandos?.....	XXVI
De qué trata este libro.....	XXVI
¿Quién debería leer este libro? .....	XXVII
¿Qué hay en este libro? .....	XXVII
Cómo leer este libro .....	XXVIII
Prerrequisitos .....	XXVIII

## PART 1 APRENDIENDO EL CAPARAZÓN

1

### ¿QUÉ ES EL CAPARAZÓN?

3

Emuladores de terminal.....	3
Tus primeras pulsaciones de teclas .....	4
Historial de comandos .....	4
Movimiento del cursor.....	4
Pruebe algunos comandos simples. ....	5
Finalización de una sesión de terminal. ....	6

2

### NAVEGACIÓN

7

Entendiendo el árbol del sistema de archivos.....	7
El directorio de trabajo actual. ....	8
Listar el contenido de un directorio .....	8
Cambiar el directorio de trabajo actual.....	9
Nombres de ruta absolutos. ....	9
Pariente Nombres de ruta .....	9
Algunos atajos útiles .....	10

**3****EXPLORANDO EL SISTEMA****13**

Más diversión con ls.....	13
Opciones y argumentos .....	14
Una mirada más larga al formato largo.....	15
Determinar el tipo de archivo con file.....	16
Ver el contenido de los archivos con menos.....	17
Una visita guiada .....	19
Vínculos simbólicos.....	22

**4****MANIPULACIÓN DE ARCHIVOS Y DIRECTORIOS****25**

Comodín.....	26
mkdir: crear directorios .....	28
cp: copiar archivos y directorios .....	28
mv: mover y cambiar el nombre de los archivos.....	30
rm: eliminar archivos y directorios .....	31
In: crear vínculos .....	32
Enlaces duros .....	32
Vínculos simbólicos.....	32
Construyamos un patio de recreo.....	33
Creación de directorios .....	33
Copia de archivos. ....	33
Mover y cambiar el nombre de los archivos.....	34
Creación de enlaces duros.....	35
Creación de enlaces simbólicos .....	36
Eliminación de archivos y directorios.....	37
Nota final.....	38

**5****TRABAJAR CON COMANDOS****39**

¿Qué son exactamente los comandos? .....	40
Identificación de comandos .....	40
type: muestra el tipo de un comando .....	40
which: muestra la ubicación de un ejecutable. ....	41
Obtener la documentación de un comando.....	41
help—Obtener ayuda para las órdenes internas de shell.....	41
- - help: mostrar información de uso.....	42
man: muestra la página de manual de un programa. ....	42
apropos: muestra los comandos adecuados. ....	43
whatis: muestra una descripción muy breve de un comando.....	44
info: muestra la entrada de información de un programa.....	44
README y otros archivos de documentación del programa .....	45
Creación de sus propios comandos con alias.....	46
Volver a visitar a viejos amigos.....	47

**6****REDIRECCIONAMIENTO****49**

Entrada, salida y error estándar.....	50
Redireccionamiento de la salida estándar.....	50
Redireccionamiento de error estándar .....	51
Redireccionamiento de la salida estándar y el error estándar a un archivo .....	52
Eliminación de salidas no deseadas.....	52
Redireccionamiento de entrada estándar.....	53
Tuberías.....	54
Filtros.....	55
uniq: informar u omitir líneas repetidas.....	55
wc: recuentos de líneas, palabras y bytes de impresión.....	55
grep: líneas de impresión que coinciden con un patrón.....	56
head/tail: imprime la primera y la última parte de los archivos.....	56
tee: lectura de Stdin y salida a stdout y archivos .....	57
Nota final.....	58

**7****VER EL MUNDO COMO LO VE EL CAPARAZÓN****59**

Expansión.....	59
Ruta Expansión .....	60
Expansión de tilde.....	61
Expansión aritmética.....	62
Expansión de la abrazadera.....	63
Expansión de parámetros.....	64
Sustitución de comandos .....	64
Citando.....	65
Comillas dobles.....	65
Comillas simples.....	67
Caracteres de escape.....	67
Nota final.....	68

**8****TRUCOS AVANZADOS DE TECLADO****69**

Edición de línea de comandos.....	70
Movimiento del cursor.....	70
Modificación de texto.....	70
Cortar y pegar (matar y tirar) texto .....	70
Terminación.....	72
Uso de la historia.....	73
Historial de búsqueda .....	74
Expansión de la historia.....	75
Nota final.....	76

**PERMISOS**

Propietarios, miembros del grupo y todos los demás .....	78
Leer, escribir y ejecutar .....	79
chmod: cambiar el modo de archivo.....	81
Configuración del modo de archivo con la interfaz gráfica de usuario.....	84
umask: establezca los permisos predeterminados.....	84
Cambio de identidades .....	87
su: ejecute un shell con ID de usuario y grupo sustitutos.....	87
sudo: ejecute un comando como otro usuario.....	88
chown: cambiar el propietario del archivo y el grupo .....	90
chgrp: cambiar la propiedad del grupo.....	91
Ejercer sus privilegios .....	91
Cambiar la contraseña.....	93

**PROCESOS**

Cómo funciona un proceso.....	96
Visualización de procesos con ps.....	96
Visualización dinámica de procesos con top .....	98
Control de procesos .....	100
Interrupción de un proceso.....	101
Poner un proceso en segundo plano.....	101
Devolver un proceso al primer plano.....	102
Detener (pausar) un proceso .....	102
Señales .....	103
Envío de señales a procesos con kill.....	103
Envío de señales a múltiples procesos con killall.....	106
Más comandos relacionados con el proceso .....	106

**PART 2****LA CONFIGURACIÓN Y EL MEDIO AMBIENTE****EL MEDIO AMBIENTE**

¿Qué se almacena en el entorno?.....	110
Examinando el medio ambiente.....	110
Algunas variables interesantes.....	111
¿Cómo se establece el entorno?.....	112
Shells de inicio de sesión y sin inicio de sesión .....	112
¿Qué hay en un archivo de inicio?.....	113

Modificación del entorno.....	115
¿Qué archivos debemos modificar? .....	115
Mensaje de texto Editores.....	115
Usar un editor de texto.....	116
Activando nuestros cambios .....	118
Nota final.....	119

## **12**

### **UNA AMABLE INTRODUCCIÓN A VI**

**121**

Por qué debemos aprender vi .....	122
Un poco de historia .....	122
Arranque y parada vi. ....	122
Modos de edición .....	123
Entrar en el modo de inserción .....	124
Salvando nuestro trabajo. ....	124
Mover el cursor.....	125
Edición básica.....	126
Anexar texto .....	127
Abrir una línea.....	127
Eliminación de texto.....	128
Cortar, copiar y pegar texto.....	129
Líneas de unión .....	131
Buscar y reemplazar. ....	131
Búsqueda dentro de una línea.....	131
Búsqueda en todo el archivo .....	131
Búsqueda y reemplazo global.....	132
Edición de varios archivos. ....	133
Cambiar entre archivos.....	134
Abrir archivos adicionales para editarlos. ....	134
Copiar contenido de un archivo a otro. ....	135
Insertar un archivo completo en otro .....	136
Salvando nuestro trabajo.....	137

## **13**

### **PERSONALIZACIÓN DE LA SOLICITUD**

**139**

Anatomía de un prompt .....	139
Probar algunos diseños de indicaciones alternativos .....	141
Adición de color .....	142
Mover el cursor.....	144
Guardar el mensaje.....	146
Nota final.....	146



## **PART 3**

# **TAREAS COMUNES Y HERRAMIENTAS ESENCIALES**

### **14**

#### **GESTIÓN DE PAQUETES**

**149**

Sistemas de embalaje .....	150
Cómo funciona un sistema de paquetes .....	150
Archivos de paquetes.....	150
Repositorios.....	151
Dependencias.....	151
Herramientas de paquete de alto y bajo nivel.....	152
Tareas comunes de administración de paquetes.....	152
Encontrar un paquete en un repositorio.....	152
Instalación de un paquete desde un repositorio.....	153
Instalación de un paquete desde un archivo de paquete.....	153
Eliminación de un paquete.....	154
Actualización de paquetes desde un repositorio.....	154
Actualización de un paquete a partir de un archivo de paquete.....	154
Listado de paquetes instalados.....	155
Determinar si un paquete está instalado.....	155
Visualización de información sobre un paquete instalado.....	155
Encontrar qué paquete instaló un archivo .....	156
Nota final.....	156

### **15**

#### **ALMACENAMIENTO MEDIO**

**159**

Montaje y desmontaje de dispositivos de almacenamiento.....	160
Visualización de una lista de sistemas de archivos montados.....	161
Determinación de los nombres de los dispositivos .....	164
Creación de nuevos sistemas de archivos .....	167
Manipulación de particiones con fdisk.....	167
Creación de un nuevo sistema de archivos con mkfs.....	169
Pruebas y reparación de sistemas de archivos.....	170
Formateo de discos.....	171
Traslado de datos directamente hacia y desde dispositivos .....	171
Creación de imágenes de CD-ROM.....	172
Creación de una copia de imagen de un CD-ROM.....	172
Creación de una imagen a partir de una colección de archivos.....	172
Escritura de imágenes de CD-ROM .....	173
Montaje de una imagen ISO directamente.....	173
Borrar un CD-ROM regrabable.....	173
Escribir una imagen.....	173
Crédito extra.....	174

**16****GESTIÓN DE REDES****175**

Examen y monitoreo de una red.....	176
ping: envíe un paquete especial a un host de red.....	176
traceroute: rastrea la ruta de un paquete de red.....	177
netstat: examinar la configuración de red y las estadísticas.....	178
Transporte de archivos a través de una red.....	179
ftp: transfiera archivos con el protocolo de transferencia de archivos.....	179
lftp: un ftp mejor.....	181
wget: descargador de red no interactivo.....	181
Comunicación segura con hosts remotos.....	182
ssh: inicie sesión de forma segura en equipos remotos.....	182
scp y sftp: transfiere archivos de forma segura .....	185

**17****BÚSQUEDA DE ARCHIVOS****187**

localizar: encuentre archivos de forma sencilla .....	188
find: encuentre archivos de la manera difícil.....	189
Pruebas .....	189
Acciones .....	194
Un regreso al patio de recreo.....	198
Opciones.....	200

**18****ARCHIVADO Y COPIA DE SEGURIDAD****201**

Compresión de archivos .....	202
gzip: comprimir o expandir archivos.....	202
bzip2: mayor compresión a costa de la velocidad .....	204
Archivado de archivos .....	205
tar: utilidad de archivado de cintas.....	205
zip: empaquetar y comprimir archivos.....	209
Sincronización de archivos y directorios .....	211
rsync: sincronización remota de archivos y directorios .....	212
Uso de rsync a través de una red.....	213

**19****REGULAR EXPRESIONES****215**

¿Qué son las expresiones regulares?.....	216
grep: búsqueda a través de texto.....	216
Metacaracteres y literales.....	217
El cualquier carácter.....	218
Anclajes.....	219

Expresiones de corchetes y clases de caracteres .....	220
Negación.....	220
Rangos de caracteres tradicionales .....	220
Clases de caracteres POSIX.....	221
POSIX Expresiones regulares básicas frente a expresiones regulares extendidas .....	224
Alternancia .....	225
Cuantificadores .....	226
?: coincide con un elemento cero veces o una vez .....	226
*: coincide con un elemento cero o más veces.....	227
+: coincide con un elemento una o más veces.....	227
{ }: coincide con un elemento un número específico de veces .....	228
Poner a trabajar las expresiones regulares.....	229
Validación de una lista de teléfonos con grep .....	229
Encontrar nombres de archivo feos con find .....	230
Búsqueda de archivos con localizar .....	230
Búsqueda de texto con less y vim .....	231
Nota final.....	232

## 20

### Mensaje de texto TRATAMIENTO

**233**

Aplicaciones del texto .....	234
Documentos .....	234
Páginas Web. ....	234
Correo electrónico.....	234
Salida de la impresora.....	234
Código fuente del programa.....	235
Volviendo a visitar a algunos viejos amigos.....	235
cat: concatenar archivos e imprimirlos en una salida estándar.....	235
sort: ordena las líneas de los archivos de texto.....	236
uniq: informar u omitir líneas repetidas.....	242
Cortar en rodajas y cubitos.....	243
cortar: elimina secciones de cada línea de archivos.....	243
pegar: fusionar líneas de archivos .....	246
join: unir líneas de dos archivos en un campo común .....	247
Comparación de texto.....	249
comm: comparar dos archivos ordenados línea por línea .....	249
diff: comparar archivos línea por línea.....	250
patch: aplicar una diferencia a un original.....	253
Edición sobre la marcha. .....	254
tr: transliterar o eliminar caracteres.....	254
sed: editor de secuencias para filtrar y transformar texto .....	256
aspell: corrector ortográfico interactivo.....	263
Nota final.....	266
Crédito extra.....	266

**21****FORMATO DE LA SALIDA****267**

Herramientas de formateo sencillas .....	268
nl: rectas numéricas .....	268
fold: ajuste cada línea a una longitud especificada .....	271
fmt: un formateador de texto simple .....	271
pr: Formato de texto para impresión .....	274
printf: formatear e imprimir datos .....	275
Sistemas de formateo de documentos .....	278
La familia roff y TEX .....	279
groff: un sistema de formateo de documentos .....	279
Nota final .....	283

**22****IMPRENTA****285**

Breve historia de la imprenta .....	286
Impresión en los tiempos oscuros .....	286
Impresoras basadas en caracteres .....	286
Impresoras gráficas .....	287
Impresión con Linux .....	288
Preparación de archivos para impresión .....	288
pr: convertir archivos de texto para imprimir .....	288
Envío de un trabajo de impresión a una impresora .....	290
lpr: imprimir archivos (estilo Berkeley) .....	290
lp: Imprimir archivos (estilo System V) .....	291
Otra opción: a2ps .....	292
Supervisión y control de trabajos de impresión .....	294
lpstat: muestra el estado del sistema de impresión .....	294
lpq: mostrar el estado de la cola de la impresora .....	295
lprm y cancel: cancelar trabajos de impresión .....	296

**23****COMPILACIÓN DE PROGRAMAS****297**

¿Qué es compilar? .....	298
¿Están compilados todos los programas? .....	299
Compilar un programa en C .....	299
Obtención del código fuente .....	300
Examinando el árbol de fuentes .....	301
Construcción del programa .....	302
Instalación del programa .....	305
Nota final .....	306



## PART 4

# ESCRIBIR GUIONES SH ELL

### 24

#### ESCRIBIR TU PRIMER GUIÓN

309

¿Qué son los scripts de shell? .....	309
Cómo escribir un script de shell.....	310
Formato de archivo de script.....	310
Permisos ejecutables.....	311
Ubicación del archivo de script.....	311
Buenas localizaciones para los guiones.....	312
Más trucos de formato .....	312
Nombres de opciones largos .....	313
Sangría y continuación de línea.....	313
Nota final.....	314

### 25

#### INICIAR UN PROYECTO

315

Primera etapa: documento mínimo.....	315
Segunda etapa: agregar un poco de datos.....	317
Variables y constantes. ....	318
Creación de variables y constantes .....	318
Asignación de valores a variables y constantes .....	320
Aquí documentos. ....	321
Nota final.....	323

### 26

#### DISEÑO TOP-DOWN

325

Funciones de shell .....	326
Variables locales.....	328
Mantenga los scripts en ejecución.....	330
Nota final.....	332

### 27

#### CONTROL DE FLUJO: RAMIFICACIÓN CON IF

333

El uso de if.....	334
Estado de salida .....	334
Uso de la prueba.....	336
Expresiones de archivo.....	336
Expresiones de cadena .....	338
Expresiones enteras .....	340

U	Una versión más moderna de la prueba .....	341
( )	: diseñado para números enteros .....	342
C	Combinación de expresiones .....	343
O	Operadores de control: otra forma de ramificarse.....	345
N	Nota final.....	346

## **28**

### **LECTURA DE LA ENTRADA DEL TECLADO**

**347**

read: lea valores de la entrada estándar .....	348
Opciones .....	351
Separación de campos de entrada con IFS.....	351
Validación de la entrada .....	353
Menús .....	355
Nota final.....	356
Crédito extra.....	356

## **29**

### **CONTROL DE FLUJO: BUCLE CON WHILE Y UNTIL**

**357**

Bucle.....	358
mientras .....	358
Salir de un bucle.....	360
hasta.....	361
Lectura de archivos con bucles .....	362
Nota final.....	362

## **30**

### **SOLUCIÓN DE PROBLEMAS**

**363**

Errores sintácticos .....	363
Faltan comillas .....	364
Tokens faltantes o inesperados .....	365
Expansiones imprevistas .....	365
Errores lógicos .....	366
Programación defensiva.....	367
Verificación de la entrada.....	368
Ensayo.....	369
Talones.....	369
Prueba Casos.....	369
Depuración.....	370
Encontrar el área problemática .....	370
Trazado .....	371
Examen de valores durante la ejecución.....	373
Nota final.....	373

<b>31</b>	<b>CONTROL DE FLUJO: RAMIFICACIÓN CON CAJA</b>	<b>375</b>
caso .....	376	
Patrones .....	377	
Combinación de múltiples patrones .....	378	
Nota final.....	379	
<b>32</b>	<b>PARÁMETROS POSICIONALES</b>	<b>381</b>
Acceso a la línea de comandos.....	381	
Determinación del número de argumentos .....	382	
shift: obtener acceso a muchos argumentos.....	383	
Aplicaciones sencillas.....	384	
Uso de parámetros posicionales con funciones de shell.....	385	
Manejo de parámetros posicionales en masa.....	385	
Una aplicación más completa.....	387	
Nota final.....	390	
<b>33</b>	<b>CONTROL DE FLUJO: BUCLE CON FOR</b>	<b>393</b>
para: Forma de concha tradicional.....	393	
para: Formulario de Lenguaje C.....	396	
Nota final.....	397	
<b>34</b>	<b>CADERAS Y NÚMEROS</b>	<b>399</b>
Expansión de parámetros.....	399	
Parámetros básicos.....	400	
Expansiones para gestionar variables vacías.....	400	
Expansiones que devuelven nombres de variables .....	401	
Operaciones de cadena.....	402	
Evaluación y expansión aritmética .....	404	
Bases numéricas .....	405	
Operadores unarios .....	405	
Aritmética simple.....	405	
Asignación.....	406	
Operaciones de bits.....	408	
Lógica.....	409	
bc: un lenguaje de calculadora de precisión arbitraria.....	411	
Usando bc .....	412	
Un script de ejemplo.....	413	
Nota final.....	414	
Crédito extra.....	414	

¿Qué son las matrices?.....	415
Creación de una matriz.....	416
Asignación de valores a una matriz.....	416
Acceso a los elementos de la matriz .....	417
Operaciones de matriz .....	418
Salida de todo el contenido de una matriz.....	419
Determinación del número de elementos de la matriz .....	419
Encontrar los subíndices utilizados por una matriz.....	420
Agregar elementos al final de una matriz .....	420
Ordenar una matriz.....	420
Eliminación de una matriz.....	421
Nota final.....	422

Comandos de grupo y subshells. ....	423
Realización de redirecciónamientos .....	424
Proceso Sustitución.....	424
Trampas.....	426
Ejecución asincrónica. ....	429
esperar .....	429
Canalizaciones .....	430
Configuración de una canalización con nombre.....	431
Uso de canalizaciones con nombre .....	431
Nota final.....	432



# **RECONOCIMIENTOS**

**Quiero agradecer a las siguientes personas que ayudaron a hacer posible este libro.**

Primero, las personas que me inspiraron: Jenny Watson, editora de adquisiciones de Wiley Publishing, originalmente me sugirió que escribiera un libro de shell-scripting. Aunque Wiley no aceptó mi propuesta, se convirtió en la base de este libro. John C. Dvorak, destacado columnista y experto, dio excelentes consejos. En una epis-

En su oda a su podcast de video, "Cranky Geeks", Dvorak describió el proceso de escritura: "Demonios. Escribe 200 palabras al día y en un año tendrás una novela". Este consejo me llevó a escribir una página al día hasta que tuve un libro. Dmitri Popov escribió un artículo en la revista *Free Software titulado* "Creación de una plantilla de libro con Writer", que me inspiró a usar OpenOffice.org Writer para componer el texto. Resultó que funcionó de maravilla.

A continuación, los voluntarios que me ayudaron a producir la versión original de este libro (disponible en LinuxCommand.org): Mark Polesky realizó una revisión y una prueba extraordinarias del texto. Jesse Becker, Tomasz Chrzczonewicz, Michael Levin y Spence Miner también probaron y revisaron partes del texto. Karen M. Shotts contribuyó con muchas horas editando mi manuscrito original.

A continuación, la buena gente de No Starch Press que trabajó mucho y duro en la elaboración de la versión comercial de mi libro: Serena Yang, directora de producción; Keith Fancher, mi editor; y el resto del personal de No Starch Press.

Y por último, a los lectores de LinuxCommand.org, que me han enviado tantos correos electrónicos amables. ¡Su aliento me dio la idea de que realmente estaba en algo!

# **INTRODUCCIÓN**

Quiero contarles una historia. No, no es la historia de cómo, en 1991, Linus Torvalds escribió la primera versión de la Kernel de Linux. Puedes leer esa historia en muchos libros de Linux. Tampoco te voy a contar la historia de cómo,

algunos años antes, Richard Stallman comenzó el Proyecto GNU para crear un sistema operativo libre similar a Unix. Esa también es una historia importante, pero la mayoría de los otros libros de Linux también tienen esa. No, quiero contarte la historia de cómo puedes recuperar el control de tu computadora.

Cuando comencé a trabajar con computadoras como estudiante universitario a fines de la década de 1970, se estaba produciendo una revolución. La invención del microprocesador había hecho posible que la gente común como tú y yo tuviéramos una computadora. Es difícil para muchas personas hoy en día imaginar cómo era el mundo cuando solo las grandes empresas y los grandes gobiernos manejaban todas las computadoras. Digamos que no se pudo hacer mucho.

Hoy, el mundo es muy diferente. Las computadoras están en todas partes, desde pequeños relojes de pulsera hasta centros de datos gigantes y todo lo demás. Además de

Computadoras ubicuas, también tenemos una red ubicua que las conecta entre sí. Esto ha creado una maravillosa nueva era de empoderamiento personal y libertad creativa, pero en las últimas dos décadas ha estado sucediendo algo más. Una sola corporación gigante ha estado imponiendo su control sobre la mayoría de las computadoras del mundo y decidiendo lo que se puede y no se puede hacer con ellas. Afortunadamente, personas de todo el mundo están haciendo algo al respecto. Están luchando para mantener el control de sus computadoras escribiendo su propio software. Están construyendo Linux.

Mucha gente habla de "libertad" con respecto a Linux, pero no creo que la mayoría de la gente sepa lo que realmente significa esta libertad. La libertad es el poder de decidir lo que hace tu computadora, y la única manera de tener esta libertad es saber lo que tu computadora está haciendo. La libertad es una computadora que no tiene secretos, una en la que todo se puede saber si te preocupas lo suficiente como para averiguarlo.

## ¿Por qué usar la línea de comandos?

¿Alguna vez has notado en las películas cuando el "súper hacker" -ya sabes, el tipo que puede entrar en la computadora militar ultra segura en menos de 30 segundos- se sienta frente a la computadora, nunca toca un mouse? Es porque los cineastas se dan cuenta de que nosotros, como seres humanos, sabemos instintivamente que la única forma de hacer algo en una computadora es escribiendo en un teclado.

La mayoría de los usuarios de computadoras de hoy en día están familiarizados solo con la *interfaz gráfica de usuario (GUI)* y los proveedores y expertos les han enseñado que la *interfaz de línea de comandos (CLI)* es algo aterrador del pasado. Esto es lamentable, porque una buena interfaz de línea de comandos es una forma maravillosamente expresiva de comunicarse con una computadora de la misma manera que la palabra escrita lo es para los seres humanos. Se ha dicho que "las interfaces gráficas de usuario facilitan las tareas, mientras que las interfaces de línea de comandos hacen posibles las tareas difíciles", y esto sigue siendo muy cierto hoy en día.

Dado que Linux se basa en la familia de sistemas operativos Unix, comparte la misma rica herencia de herramientas de línea de comandos que Unix. Unix cobró importancia a principios de la década de 1980 (aunque se desarrolló por primera vez una década antes), antes de la adopción generalizada de la interfaz gráfica de usuario y, como resultado, desarrolló una extensa interfaz de línea de comandos en su lugar. De hecho, una de las razones más fuertes por las que los primeros usuarios de Linux lo eligieron en lugar de, por ejemplo, Windows NT fue la poderosa interfaz de línea de comandos, que hizo posible las "tareas difíciles".

## De qué trata este libro

Este libro es una visión general de "vivir" en la línea de comandos de Linux. A diferencia de algunos libros que se concentran en un solo programa, como el programa shell, bash, este libro intentará transmitir cómo llevarse bien con la interfaz de línea de comandos en un sentido más amplio. ¿Cómo funciona todo? ¿Qué puede hacer? ¿Cuál es la mejor manera de usarlo?





**Este no es un libro sobre administración de sistemas Linux.** Si bien cualquier discusión seria sobre la línea de comandos conducirá invariablemente a temas de administración de sistemas, este libro toca solo algunos temas de administración. Sin embargo, preparará al lector para un estudio adicional al proporcionar una base sólida en el uso de la línea de comandos, una herramienta esencial para cualquier tarea seria de administración de sistemas.

**Este libro está muy centrado en Linux.** Muchos otros libros intentan ampliar su atractivo incluyendo otras plataformas, como Unix genérico y Mac OS X. Al hacerlo, "diluyen" su contenido para presentar solo temas generales. Este libro, por otro lado, cubre solo las distribuciones contemporáneas de Linux. El noventa y cinco por ciento del contenido es útil para los usuarios de otros sistemas tipo Unix, pero este libro está muy dirigido al usuario moderno de la línea de comandos de Linux.

## ¿Quién debería leer este libro?

Este libro es para nuevos usuarios de Linux que han migrado desde otras plataformas. Lo más probable es que seas un "usuario avanzado" de alguna versión de Microsoft Windows. Tal vez tu jefe te haya dicho que administre un servidor Linux, o tal vez solo sea un usuario de escritorio que está cansado de todos los problemas de seguridad y quiere probar Linux. Está bien. Todos son bienvenidos aquí.

Dicho esto, no hay un atajo para la iluminación de Linux. Aprender la línea de comandos es un desafío y requiere un verdadero esfuerzo. No es que sea tan difícil, sino que es tan *vasto*. El sistema Linux promedio tiene literalmente *miles* de programas que puede emplear en la línea de comandos. Considérese advertido: aprender la línea de comandos no es un esfuerzo casual.

Por otro lado, aprender la línea de comandos de Linux es extremadamente gratificante. Si crees que eres un "usuario avanzado" ahora, solo espera. Todavía no sabes lo que es el poder real. Y, a diferencia de muchas otras habilidades informáticas, el conocimiento de la línea de comandos es duradero. Las habilidades aprendidas hoy seguirán siendo útiles dentro de 10 años. La línea de comandos ha sobrevivido a la prueba del tiempo.

También se supone que no tienes experiencia en programación, no te preocupes. Te iniciaremos en ese camino también.

## ¿Qué hay en este libro?

Este material se presenta en una secuencia cuidadosamente elegida, como si un tutor estuviera sentado a tu lado, guiándote. Muchos autores tratan este material de una manera "sistémica", lo que tiene sentido desde la perspectiva de un escritor, pero puede ser muy confuso para los nuevos usuarios.

Otro objetivo es familiarizarlo con la forma de pensar de Unix, que es diferente de la forma de pensar de Windows. A lo largo del camino, haremos algunos viajes secundarios para ayudarlo a comprender por qué ciertas cosas funcionan de la manera en que lo hacen y cómo llegaron a ser así. Linux no

es solo una pieza de software; también es una pequeña parte de la cultura Unix más grande, que tiene su propio lenguaje e historia. También podría lanzar una perorata o dos.

Este libro está dividido en cuatro partes, cada una de las cuales cubre algún aspecto de la experiencia de la línea de comandos:

- **Parte 1: Aprendiendo el Shell** comienza nuestra exploración del lenguaje básico de la línea de comandos, incluyendo cosas como la estructura de los comandos, la navegación del sistema de archivos, la edición de la línea de comandos y la búsqueda de ayuda y documentación para los comandos.
- **Parte 2: Configuración y entorno** cubre la edición de archivos de configuración que controlan el funcionamiento de la computadora desde la línea de comandos.
- **Parte 3: Tareas Comunes y Herramientas Esenciales** explora muchas de las tareas ordinarias que comúnmente se realizan desde la línea de comandos. Los sistemas operativos tipo Unix, como Linux, contienen muchos programas de línea de comandos "clásicos" que se utilizan para realizar operaciones potentes con los datos.
- **Parte 4: Escritura de scripts de shell** presenta la programación de shell, una técnica ciertamente rudimentaria, pero fácil de aprender, para automatizar muchas tareas informáticas comunes. Al aprender a programar shell, se familiarizará con conceptos que se pueden aplicar a muchos otros lenguajes de programación.

## Cómo leer este libro

Empieza por el principio del libro y síguelo hasta el final. No está escrito como una obra de referencia; En realidad, es más como una historia con un principio, un medio y un final.

## Prerrequisitos

Para usar este libro, todo lo que necesitará es una instalación de Linux que funcione. Puede obtenerlo de una de estas dos maneras:

- **Instala Linux en un ordenador (no tan nuevo).** No importa qué distribución elijas, aunque la mayoría de la gente hoy en día comienza con Ubuntu, Fedora u OpenSUSE. En caso de duda, pruebe primero con Ubuntu. Instalar una distribución moderna de Linux puede ser ridículamente fácil o ridículamente difícil, dependiendo de su hardware. Sugiero una computadora de escritorio que tenga un par de años y tenga al menos 256 MB de RAM y 6 GB de espacio libre en el disco duro. Evite las computadoras portátiles y las redes inalámbricas si es posible, ya que a menudo son más difíciles de hacer funcionar.
- **Usa un CD en vivo.** Una de las cosas interesantes que se pueden hacer con muchas distribuciones de Linux es ejecutarlas directamente desde un CD-ROM sin necesidad de instalarlas. En absoluto. Sólo tienes que ir a la configuración de tu BIOS, configurar tu ordenador en "Arrancar desde CDROM", insertar el CD en vivo y reiniciar. Usar un CD en vivo es una excelente manera



para probar la compatibilidad de un equipo con Linux antes de la instalación. La desventaja de usar un Live CD es que puede ser muy lento en comparación con tener Linux instalado en su disco duro. Tanto Ubuntu como Fedora (entre otros) tienen versiones en CD en vivo.

**Nota:** *Independientemente de cómo instale Linux, necesitará tener privilegios de superusuario ocasionales (es decir, administrativos) para llevar a cabo las lecciones de este libro.*

Una vez que tenga una instalación que funcione, comience a leer y siga con su propia computadora. La mayor parte del material de este libro es "manos a la obra", ¡así que siéntate y ponte a escribir!

### **POR QUÉ NO LO LLAMO "GNU/ LINUX"**

En algunos sectores, es políticamente correcto llamar al sistema operativo Linux el "sistema operativo GNU/Linux". El problema con "Linux" es que no hay una forma completamente correcta de nombrarlo porque fue escrito por muchas personas diferentes en un vasto esfuerzo de desarrollo distribuido. Técnicamente hablando, Linux es el nombre del kernel del sistema operativo, nada más. El kernel es muy importante, por supuesto, ya que hace que el sistema operativo funcione, pero no es suficiente para formar un sistema operativo completo.

Entra en escena Richard Stallman, el genio filósofo que fundó el movimiento del Software Libre, creó la Fundación para el Software Libre, formó el Proyecto GNU, escribió la primera versión del Compilador C de GNU (GCC), creó la Licencia Pública General de GNU (la GPL), etc., etc. Insiste en que lo llames "GNU/Linux" para reflejar adecuadamente las contribuciones del Proyecto GNU. Si bien el Proyecto GNU es anterior al kernel de Linux y las contribuciones del proyecto son extremadamente dignas de reconocimiento, colocarlas en el nombre es injusto para todos los demás que hicieron contribuciones significativas. Además, creo que "Linux/GNU" sería técnicamente más preciso ya que el kernel arranca primero y todo lo demás se ejecuta sobre él.

En el uso popular, "Linux" se refiere al kernel y a todo el resto del software libre y de código abierto que se encuentra en la distribución típica de Linux, es decir, todo el ecosistema de Linux, no sólo los componentes de GNU. El mercado de sistemas operativos parece preferir nombres de una sola palabra como DOS, Windows, Solaris, Irix, AIX. He optado por utilizar el formato popular. Sin embargo, si prefiere usar "GNU/Linux" en su lugar, por favor realice una búsqueda mental y reemplace mientras lee este libro. No me importará.



# **PARTES**

**Aprendiendo el Shell**



# 1

## ¿QUÉ ES EL CAPARAZÓN?

Cuando hablamos de la línea de comandos, en realidad nos estamos refiriendo al shell. El *shell* es un programa que toma comandos del teclado y los pasa al sistema operativo para que los lleve a cabo. Casi todas las distribuciones de Linux suministran un programa shell del Proyecto GNU llamado bash. El nombre *bash* es un acrónimo de *Bourne Again Shell*, una referencia al hecho de que bash es un reemplazo mejorado de sh, el programa original de shell de Unix escrito por Steve Bourne.

### Emuladores de terminal

Cuando usamos una interfaz gráfica de usuario, necesitamos otro programa llamado *emulador de terminal* para interactuar con el shell. Si miramos a través de nuestro escritorio-menús superiores, probablemente encontraremos uno. KDE usa konsole y GNOME usa gnome-terminal, aunque es probable que se llame simplemente "terminal" en nuestro menú. Un

Hay muchos otros emuladores de terminal disponibles para Linux, pero todos hacen básicamente lo mismo: darnos acceso al shell. Es probable que desarrolles una preferencia por uno u otro en función de la cantidad de campanas y silbatos que tenga.

## Tus primeras pulsaciones de teclas

Así que comencemos. ¡Inicie el emulador de terminal! Una vez que aparezca, deberías ver algo como esto:

---

```
[me@linuxbox ~]$
```

---

Esto se denomina símbolo del *sistema de shell* y aparece siempre que el shell está listo para aceptar entradas. Si bien puede variar un poco en apariencia, dependiendo de

La distribución, por lo general, incluirá su *username@machinename*, seguido del directorio de trabajo actual (más sobre eso en un momento) y un signo de dólar.

Si el último carácter de la solicitud es una almohadilla (#) en lugar de un signo dólar, la sesión de terminal tiene *privilegios de superusuario*. Esto significa que, o bien hemos iniciado sesión como usuario root, o bien hemos seleccionado un emulador de terminal que proporciona privilegios de superusuario (administrativos).

Suponiendo que las cosas van bien hasta ahora, intentemos escribir un poco. Ingrese un galimatías en el símbolo del sistema como este:

---

```
[me@linuxbox ~]$ kaekfjaeifj
```

---

Como este comando no tiene sentido, el shell nos lo dice y nos da otra oportunidad:

---

```
bash: kaekfjaeifj: comando no encontrado  
[me@linuxbox ~]$
```

---

### *Histórial de comandos*

Si presionamos la tecla de flecha hacia arriba, vemos que el comando anterior kaekfjaeifj vuelve a aparecer después del mensaje. A esto se le llama *histórial de comandos*. La mayoría de las distribuciones de Linux recuerdan los últimos 500 comandos de forma predeterminada. Presione la tecla de flecha hacia abajo y el comando anterior desaparecerá.

### *Movimiento del cursor*

Recupere el comando anterior con la tecla de flecha hacia arriba nuevamente. Ahora pruebe las teclas de flecha izquierda y derecha. ¿Ves cómo podemos colocar el cursor en cualquier lugar de la línea de comandos? Esto facilita la edición de comandos.

## ALGUNAS PALABRAS SOBRE LOS RATONES Y FOCU S

Si bien el shell tiene que ver con el teclado, también puede usar un mouse con su emulador de terminal. Un mecanismo integrado en el sistema X Window (el motor subyacente que hace que la interfaz gráfica de usuario funcione) admite una técnica rápida de copiar y pegar. Si se resalta un texto manteniendo pulsado el botón izquierdo del ratón y arrastrándolo sobre él (o haciendo doble clic en una palabra), se copia en un botón mantenido por X. Al pulsar el botón central del ratón, el texto se pegará en la ubicación del cursor. Pruébalo.

No caiga en la tentación de usar CTRL-C y CTRL-V para copiar y pegar dentro de una ventana de terminal. No funcionan. Para el shell, estos códigos de control tienen diferentes significados que se asignaron muchos años antes de que Microsoft Windows entrara en escena.

Su entorno gráfico de escritorio (probablemente KDE o GNOME), en un esfuerzo por comportarse como Windows, probablemente tenga su política de *enfoque* establecida en "clic para enfocar". Esto significa que para que una ventana se enfoque (se active), debe hacer clic en ella. Esto es contrario al comportamiento tradicional de X de "el enfoque sigue al mouse", lo que significa que una ventana obtiene el enfoque cuando el mouse simplemente pasa sobre ella. La ventana no aparecerá en primer plano hasta que haga clic en ella, pero podrá recibir entradas. Establecer la política de enfoque en "el foco sigue al mouse" facilitará el uso de las ventanas de terminal. Pruébalo. Creo que si le das una oportunidad lo preferirás. Encontrará este ajuste en el programa de

## Pruebe algunos comandos simples

Ahora que hemos aprendido a escribir, probemos algunos comandos simples. El primero es la fecha. Este comando muestra la fecha y la hora actuales:

---

```
[me@linuxbox ~]$ fecha  
Thu Oct 25 13:51:54 EDT 2012
```

---

Un comando relacionado es cal, que, de forma predeterminada, muestra un calendario del mes actual:

---

```
[me@linuxbox ~]$ cal  
Octubre 2012  
Su Mo Tu We Th Fr Sa  
     1  2  3  4  5  6  
     7  8  9 10 11 12 13  
14 15 16 17 18 19 20  
21 22 23 24 25 26 27  
28 29 30 31
```

---

Para ver la cantidad actual de espacio libre en las unidades de disco, escriba **df**:

---

```
[me@linuxbox ~]$ df
Sistema de archivos      1K-bloques   Usado   Uso Disponible% Montado en
/dev/sda2                15115452    5012392  9949716 34% /
/dev/sda5                59631908  26545424 30008432 47% /hogar
/dev/sda1                147764     17370   122765 13% /arranque
TMPFS                   256856        0   256856  0% /dev/shm
```

---

Del mismo modo, para mostrar la cantidad de memoria libre, introduzca el **archivo** mandar:

---

```
[me@linuxbox ~]$ gratis
          total       usado     Gratis  compartidoBúferes      Mem
en caché:513712      503976      9736        0      5312  122916
-/+ búferes/caché: 375748      137964
Permuta: 1052248    104712      947536
```

---

## Finalización de una sesión de terminal

Podemos finalizar una sesión de terminal cerrando el emulador de terminal win-dow o ingresando el comando **exit** en el símbolo del shell:

---

```
[me@linuxbox ~]$ salir
```

---

### LA CONSOLA DETRÁS DE LA CURTA EN

Incluso si no tenemos ningún emulador de terminal en ejecución, varias sesiones de terminal continúan ejecutándose detrás del escritorio gráfico. Llamadas *terminales virtuales* o *consolas virtuales*, se puede acceder a estas sesiones en la mayoría de las distribuciones de Linux presionando CTRL-ALT-F1 a CTRL-ALT-F6 en la mayoría de los sistemas. Cuando se accede a una sesión, presenta un mensaje de inicio de sesión en el que podemos introducir nuestro nombre de usuario y contraseña. Para cambiar de una consola virtual a otra, presione ALT y F1-F6. Para volver al escritorio gráfico, presione ALT-F7.

# 2

## NAVEGACIÓN

Lo primero que tenemos que aprender (además de escribir) es cómo navegar por el sistema de archivos de nuestro sistema Linux. En este capítulo presentaremos los siguientes comandos:

- `pwd`: imprime el nombre del directorio de trabajo actual.
- `cd`: cambiar directorio.
- `ls`: enumera el contenido del directorio.

### Entendiendo el árbol del sistema de archivos

Al igual que Windows, un sistema operativo similar a Unix como Linux organiza sus archivos en lo que se denomina una *estructura jerárquica de directorios*. Esto significa que están organizados en un patrón de directorios en forma de árbol (a veces llamados carpetas en otros sistemas), que pueden contener archivos y otros directorios. El primer directorio del sistema de archivos se denomina *directorio raíz*. El directorio raíz contiene archivos y subdirectorios, que a su vez contienen más archivos y subdirectorios, y así sucesivamente.

Tenga en cuenta que, a diferencia de Windows, que tiene un árbol de sistema de archivos separado para cada dispositivo de almacenamiento, los sistemas similares a Unix, como Linux, siempre tienen un solo árbol de sistema de archivos, independientemente de cuántas unidades o dispositivos de almacenamiento estén conectados a la computadora. Los dispositivos de almacenamiento se fijan (o más correctamente, se *montan*) en varios puntos del árbol según los caprichos del administrador del *sistema*, la persona (o personas) responsables del mantenimiento del sistema.

## El directorio de trabajo actual

La mayoría de nosotros probablemente estamos familiarizados con un administrador de archivos gráfico, que representa el árbol del sistema de archivos, como en la Figura 2-1. Nótese que el árbol suele mostrarse al revés, es decir, con la raíz en la parte superior y las distintas ramas descendiendo por debajo.

Sin embargo, la línea de comandos no tiene imágenes, por lo que para navegar por el árbol del sistema de archivos, debemos pensar en él de una manera diferente.

Imaginemos que el sistema de archivos es un laberinto con la forma de un árbol al revés y somos capaces de pararnos en medio de ella. En un momento dado, estamos dentro de un solo directorio y podemos ver los archivos contenidos en el directorio y la ruta al directorio que está por encima de nosotros (llamado *directorio principal*) y cualquier subdirectorio directos por debajo de nosotros. El directorio en el que nos encontramos se llama directorio de *trabajo actual*. Para mostrar el directorio de trabajo actual, usamos el comando `pwd` (imprimir directorio de trabajo):

```
[me@linuxbox -]$ pwd  
/inicio/yo
```

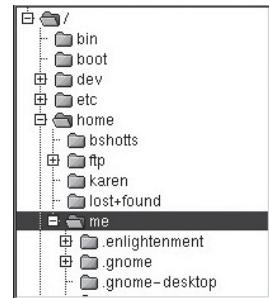


Figura 2-1: Árbol del sistema de archivos como se muestra en un administrador de archivos gráfico

Cuando iniciamos sesión por primera vez en nuestro sistema (o iniciamos una sesión de emulador de terminal), nuestro directorio de trabajo actual se establece en nuestro *directorio de inicio*. A cada cuenta de usuario se le asigna su propio directorio de inicio, que es el único lugar donde el usuario puede escribir archivos cuando opera como un usuario normal.

## Listar el contenido de un directorio

Para listar los archivos y directorios en el directorio de trabajo actual, usamos el comando

Comando `ls` :

[me@linuxbox ~]\$ ls

Documentos de escritorio Música Imágenes Plantillas públicas Videos

---

En realidad, podemos usar el comando ls para enumerar el contenido de cualquier dirección, no solo el directorio de trabajo actual, y también puede hacer muchas otras cosas divertidas. Dedicaremos más tiempo a ls en el Capítulo 3.

# Cambiar el directorio de trabajo actual

Para cambiar su directorio de trabajo (donde estamos parados en nuestro laberinto en forma de árbol) usamos el comando `cd`: Escriba `cd` seguido del nombre de la ruta del directorio de trabajo deseado. Un *pathname* es la ruta que tomamos a lo largo de las ramas del árbol para llegar al directorio que queremos. Los nombres de ruta se pueden especificar de dos maneras, como nombres de ruta absolutos o como nombres de ruta relativos. Vamos a tratar primero con los nombres de ruta absolutos.

## Nombres de ruta absolutos

Un *nombre de ruta absoluto* comienza con el directorio raíz y sigue el árbol rama por rama hasta que se completa la ruta al directorio o archivo deseado. Por ejemplo, hay un directorio en su sistema en el que están instalados la mayoría de los programas de su sistema. El nombre de la ruta de ese directorio es

`/usr/bin`. Esto significa que desde el directorio raíz (representado por la barra diagonal inicial en el nombre de la ruta) hay un directorio llamado `usr` que contiene una dirección llamada `bin`.

---

```
[me@linuxbox ~]$ cd /usr/bin  
[me@linuxbox bin]$ pwd  
/usr/bin  
[me@linuxbox bin]$ ls
```

... Listado de muchos, muchos archivos ...

---

Ahora podemos ver que hemos cambiado el directorio de trabajo actual a `/usr/bin` y que está lleno de archivos. ¿Te das cuenta de cómo ha cambiado el símbolo del shell? Para mayor comodidad, generalmente se configura para mostrar automáticamente el nombre del directorio de trabajo.

## Nombres de ruta relativos

Donde un nombre de ruta absoluto comienza en el directorio raíz y conduce a su destino, un *nombre de ruta relativo* comienza en el directorio de trabajo. Para ello, utiliza un par de símbolos especiales para representar posiciones relativas en el árbol del sistema de archivos. Estos símbolos especiales son `.` (punto) y `..` (punto punto).

El archivo `.` se refiere al directorio de trabajo y el símbolo `..` hace referencia al directorio principal del directorio de trabajo. Así es como funciona. Cambiemos de nuevo el directorio de trabajo a `/usr/bin`:

---

```
[me@linuxbox ~]$ cd /usr/bin  
[me@linuxbox bin]$ pwd  
/usr/bin
```

---



Bien, ahora digamos que queríamos cambiar el directorio de trabajo al padre de `/usr/bin`, que es `/usr`. Podríamos hacerlo de dos maneras diferentes, ya sea con un nombre de ruta absoluto:

---

```
[me@linuxbox bin]$ cd /usr  
[me@linuxbox usr]$ pwd  
/Usr
```

---

o con un nombre de ruta relativo:

---

```
[me@linuxbox bin]$ cd ..  
[me@linuxbox usr]$ pwd  
/Usr
```

---

Dos métodos diferentes producen resultados idénticos. ¿Cuál deberíamos usar? ¡El que menos requiere escribir!

Del mismo modo, podemos cambiar el directorio de trabajo de `/usr` a `/usr/bin` de dos maneras diferentes, ya sea usando un nombre de ruta absoluto:

---

```
[me@linuxbox usr]$ cd /usr/bin  
[me@linuxbox bin]$ pwd  
/usr/bin
```

---

o con un nombre de ruta relativo:

---

```
[me@linuxbox usr]$ cd ./bin  
[me@linuxbox bin]$ pwd  
/usr/bin
```

---

Ahora bien, hay algo importante que debo señalar aquí. En casi todos los casos, puede omitir el `./` porque está implícito. Mecanografía

---

```
[me@linuxbox usr]$ Bandeja de CD
```

---

hace lo mismo. En general, si no especifica un nombre de ruta para algo, se asumirá el directorio de trabajo.

### Algunos atajos útiles

En la Tabla 2-1 vemos algunas formas útiles en que el directorio de trabajo actual se puede cambiar rápidamente.

Tabla 2-1: Atajos de `cd`

Atajo	Resultado
CD	Cambia el directorio de trabajo a su directorio de inicio.
CD -	Cambia el directorio de trabajo al directorio de trabajo anterior directorio.
CD~usernameChanges	el directorio de trabajo al directorio de inicio de <i>nombre de usuario</i> . Por ejemplo CD ~bob Cambia el directorio al directorio de inicio del usuario <i>bob</i> .

## FACTS IMPORTANTES SOBRE LOS NOMBRES DE ARCHIVO

- Los nombres de archivo que comienzan con un carácter de punto están ocultos. Esto solo significa que ls no los enumerará a menos que diga ls -a. Cuando se creó su cuenta, se colocaron varios archivos ocultos en su directorio de inicio para configurar cosas para su cuenta. Más adelante echaremos un vistazo más de cerca a algunos de esos archivos para ver cómo puede personalizar su entorno. Además, algunas aplicaciones colocan sus archivos de configuración y configuración en su directorio de inicio como archivos ocultos.
- Los nombres de archivo y los comandos en Linux, al igual que en Unix, distinguen entre mayúsculas y minúsculas. Los nombres de archivo *File1* y *file1* se refieren a archivos diferentes.
- Linux no tiene el concepto de una "extensión de archivo" como otros sistemas operativos. Puede nombrar los archivos de la forma que deseé. El contenido y/o la finalidad de un archivo se determina por otros medios. Aunque los sistemas operativos tipo Unix no utilizan extensiones de archivo para determinar el contenido o el propósito de los archivos, algunos programas de aplicación sí lo hacen.
- Aunque Linux admite nombres de archivo largos que pueden contener espacios incrustados y caracteres de puntuación, limite los caracteres de puntuación en los nombres de los archivos que cree a punto, guión (guion) y guión bajo. *Lo más importante es que no incruste espacios en los nombres de archivo.* Incrustar espacios en los nombres de archivo hará que muchas tareas de línea de comandos sean más difíciles, como descubriremos en el Capítulo 7. Si desea representar espacios entre palabras en un nombre de archivo, utilice caracteres de subrayado. Se lo agradecerá más tarde.



# 3

## EXPLORANDO EL SISTEMA

Ahora que sabemos cómo movernos por el sistema de archivos, es hora de un recorrido guiado por nuestro sistema Linux. Sin embargo, antes de comenzar, vamos a aprender algunos comandos más que serán útiles en el camino:

- **ls**: enumera el contenido del directorio.
- **file**: determine el tipo de archivo.
- **less**: permite ver el contenido del archivo.

### Más diversión con ls

ls es probablemente el comando más utilizado y por una buena razón. Con él, podemos ver el contenido del directorio y determinar una variedad de atributos importantes de archivo y directorios. Como hemos visto, simplemente podemos introducir ls para ver una lista de archivos y subdirectorios contenidos en el directorio de trabajo actual:

---

```
[me@linuxbox ~]$ ls
Documentos de escritorio Música Imágenes Plantillas públicas Videos
```

---

Además del directorio de trabajo actual, podemos especificar el directorio a listar, así:

---

```
[me@linuxbox ~]$ ls /usr
JUEGOS DE BASURA      kerberos libexec
sbin                  src, etc. incluyen Lib
                      local   Compartir Tmp
```

---

o incluso especificar varios directorios. En este ejemplo, enumeraremos tanto el directorio de inicio del usuario (simbolizado por el carácter ~) como el directorio */usr*:

---

```
[me@linuxbox ~]$ ls ~ /usr
/casa/yo:
Documentos de escritorio Música Imágenes Plantillas públicas Videos
/Usr:
JUEGOS DE BASURA      kerberos libexec
sbin                  src, etc. incluyen Lib
                      local   Compartir Tmp
```

---

También podemos cambiar el formato de la salida para revelar más detalles:

---

```
[me@linuxbox ~]$ ls -l
Total 56
Drxervser-S 2M Mi 4096 2012-10-26 17:20 Escritorio
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Documentos
Drxervser-S 2m Me 4096 2012-10-26 17:20 Música
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Imágenes
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Público
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Plantillas
drwxrwxr-x 2 me me 4096 2012-10-26 17:20 Vídeos
```

---

Al agregar *-l* al comando, cambiamos la salida al formato largo .

### Opciones y argumentos

Esto nos lleva a un punto muy importante sobre cómo funcionan la mayoría de los comandos. Los comandos suelen ir seguidos de una o varias *opciones* que modifican su comportamiento y, además, de uno o varios *argumentos*, los elementos sobre los que actúa el comando. Por lo tanto, la mayoría de los comandos tienen un aspecto similar al siguiente:

#### Argumentos de command -options

La mayoría de los comandos utilizan opciones que constan de un solo carácter precedido por un guión, como *-l*. Pero muchos comandos, incluidos los del Proyecto GNU, también admiten *opciones largas*, que consisten en una palabra precedida por dos guiones. Además, muchos comandos permiten encadenar varias opciones cortas. En este ejemplo, el comando *ls* tiene dos opciones, la opción *l* para producir una salida de formato largo y la opción *t* para ordenar el resultado por el tiempo de

---

modificación del archivo:

---

```
[me@linuxbox ~]$ ls -lt
```

Agregaremos la opción long --reverse para invertir el orden de la orden:

```
[me@linuxbox ~]$ ls -lt --reverse
```

El comando ls tiene un gran número de opciones posibles. Los más comunes se enumeran en la Tabla 3-1.

Tabla 3-1: Opciones comunes de ls

Opción	Opción larga	Descripción
-un	--todo	Enumere todos los archivos, incluso aquellos con nombres que comienzan con un punto, que normalmente no se enumeran (es decir, oculto).
-d	--directorio	Normalmente, si se especifica un directorio, ls enumerará el contenido del directorio, no el directorio directorio en sí. Utilice esta opción junto con la opción -l para ver detalles sobre el directorio en lugar de su contenido.
-F	--clasificar	Esta opción agregará un carácter de indicador al final de cada nombre enumerado (por ejemplo, un barra diagonal si el nombre es un directorio).
-h	--legible por humanos	En los listados de formato largo, muestra los tamaños de archivo en Formato legible por humanos en lugar de en bytes.
-l		Muestra los resultados en formato largo.
-r	--Marcha atrás	Muestre los resultados en orden inverso. Normalmente ls muestra sus resultados en alfabético.
-S		Ordene los resultados por tamaño de archivo.
-t		Ordenar por hora de modificación.

### *Una mirada más larga al formato largo*

Como vimos antes, la opción -l hace que ls muestre sus resultados en formato largo. Este formato contiene una gran cantidad de información útil. Aquí está el directorio de ejemplos de un sistema Ubuntu:

```
-rw-r--r-- 1 raíz raíz 3576296 2012-04-03 11:05 Experiencia ubuntu.ogg
-rw-r--r-- 1 raíz raíz 1186219 2012-04-03 11:05 kubuntu-leaflet.png
-rw-r--r-- 1 raíz raíz      47584 03/04/2012 11:05 a. m. logo-Edubuntu.png
-rw-r--r-- 1 raíz raíz     44355 03/04/2012 11:05 a. m. logo-Kubuntu.png
-rw-r--r-- 1 raíz raíz    34391 03/04/2012 11:05 logo-Ubuntu.png
```

-rw-r--r-- 1 raíz raíz 32059 2012-04-03 11:05 oo-cd-cover.odf  
-rw-r--r-- 1 raíz raíz 159744 2012-04-03 11:05 oo-derivatives.doc  
-rw-r--r-- 1 raíz raíz 27837 03/04/2012 11:05 oo-maxwell.odt  
-rw-r--r-- 1 raíz raíz 98816 03/04/2012 11:05 oo-trig.xls

```
-rw-r--r-- 1 raíz raíz 453764 2012-04-03 11:05 oo-welcome.odt
-rw-r--r-- 1 root root 358374 2012-04-03 11:05 ubuntu Sax.ogg
```

---

Echemos un vistazo a los diferentes campos de uno de los archivos y examinemos sus significados en la Tabla 3-2.

**Tabla 3-2: Campos de lista larga ls**

Campo	Significado
-rw-r--r--	Derechos de acceso al fichero. El primer carácter indica el tipo de archivo. Entre los diferentes tipos, un guión inicial significa un archivo normal, mientras que un d Indica un directorio. Los tres caracteres siguientes son los derechos de acceso para el propietario del archivo, los tres siguientes son para los miembros del grupo del archivo y los tres últimos son para todos los demás. El significado completo de esto se discute en el Capítulo 9.
1	Número de enlaces duros del archivo. Véase la discusión de los enlaces al final de este capítulo.
raíz	El nombre de usuario del propietario del archivo.
raíz	Nombre del grupo propietario del archivo.
32059	Tamaño del archivo en bytes.
2012-04-03 11:05	Fecha y hora de la última modificación del archivo.
oo-cd-portada.odf	Nombre del archivo.

## Determinar el tipo de archivo con file

A medida que exploramos el sistema, será útil saber qué contienen los archivos. Para hacer esto, usaremos el comando `file` para determinar el tipo de archivo. Como discutimos anteriormente, los nombres de archivo en Linux no son necesarios para reflejar el contenido de un archivo. Por ejemplo, mientras que normalmente se esperaría que un nombre de archivo como *picture.jpg* contenga una imagen comprimida JPEG, no es necesario en Linux. Podemos invocar el comando `file` de esta manera:

`nombre de archivo`

Cuando se invoca, el comando `file` imprimirá una breve descripción del contenido del archivo. Por ejemplo:

---

```
[me@linuxbox ~]$ archivo picture.jpg
picture.jpg: Datos de imagen JPEG, estÁndar JFIF 1.01
```

---

Hay muchos tipos de archivos. De hecho, una de las ideas comunes en los sistemas operativos tipo Unix, como Linux, es que "todo es un archivo". A medida que avancemos en nuestras lecciones, veremos cuán cierta es esa afirmación.

Si bien muchos de los archivos de su sistema son familiares, por ejemplo, los archivos MP3 y JPEG, muchos tipos son un poco menos obvios y algunos son bastante extraños.

## Visualización del contenido de los archivos con menos

El comando `less` es un programa para ver archivos de texto. En todo nuestro sistema Linux, hay muchos archivos que contienen texto legible por humanos. El programa menor proporciona una forma conveniente de examinarlos.

¿Por qué queríamos examinar archivos de texto? Debido a que muchos de los archivos que contienen la configuración del sistema (llamados *archivos de configuración*) se almacenan en este formato, poder leerlos nos da una idea de cómo funciona el sistema. Además, muchos de los programas reales que utiliza el sistema (llamados *scripts*) se almacenan en este formato. En capítulos posteriores, aprenderemos cómo editar archivos de texto para modificar la configuración del sistema y escribir nuestros propios scripts, pero por ahora solo veremos su contenido.

### ¿QUÉ ES "TEXTO"?

Hay muchas maneras de representar información en una computadora. Todos los métodos implican definir una relación entre la información y algunos números que se utilizarán para representarla. Al fin y al cabo, los ordenadores sólo entienden números, y todos los datos se convierten en representaciones numéricas.

Algunos de estos sistemas de representación son muy complejos (como los archivos de vídeo comprimidos), mientras que otros son bastante sencillos. Uno de los primeros y más sencillos se llama *texto ASCII*. ASCII (pronunciado "As-Key") es la abreviatura de Código Estándar Americano para el Intercambio de Información. Este sencillo esquema de codificación se utilizó por primera vez en las máquinas de teletipo.

El texto es una simple asignación uno a uno de caracteres a números. Es muy compacto. Cincuenta caracteres de texto se traducen en cincuenta bytes de datos. No es lo mismo que el texto de un documento de procesador de textos, como uno creado por Microsoft Word o OpenOffice.org Writer. Estos archivos, a diferencia del texto ASCII simple, contienen muchos elementos no textuales que se utilizan para describir su estructura y formato. Los archivos de texto ASCII sin formato contienen solo los caracteres y algunos códigos de control rudimentarios como tabulaciones, retornos de carro y saltos de línea.

En un sistema Linux, muchos archivos se almacenan en formato de texto, y muchas herramientas de Linux trabajan con archivos de texto. Incluso Windows reconoce la importancia de este formato. El conocido programa Bloc de notas es

El comando less se usa de la siguiente manera:

menos *nombre de archivo*

Una vez iniciado, el programa less le permite desplazarse hacia adelante y hacia atrás a través de un archivo de texto. Por ejemplo, para examinar el archivo que define todas las cuentas de usuario del sistema, escriba el siguiente comando:

---

```
[me@linuxbox ~]$ menos /etc/passwd
```

---

Una vez que se inicia el programa less, podemos ver el contenido del archivo. Si el archivo tiene más de una página, podemos desplazarnos hacia arriba y hacia abajo. Para salir menos, presione la tecla Q.

En la tabla 3-3 se enumeran los comandos de teclado más comunes utilizados por menos.

**Tabla 3-3: menos comandos**

Mandar	Acción
SUBIR PÁGINA o b	Desplázate hacia atrás una página.
AVANCE DE PÁGINA O Barra espaciadora	Desplácese hacia adelante una página.
Flecha hacia arriba	Desplázate una línea hacia arriba.
Flecha hacia abajo	Desplázate hacia abajo una línea.
G	Desplácese hasta el final del archivo de texto.
1G o g	Vaya al principio del archivo de texto.
/Caracteres	Busque hacia adelante hasta la próxima ocurrencia de Caracteres.
n	Busque la siguiente aparición de la búsqueda anterior.
h	Mostrar pantalla de ayuda.
q	Renunciar menos.

## MENOS ES MÁS

El programa less fue diseñado como un reemplazo mejorado de un programa Unix anterior llamado more. Su nombre es un juego de palabras con la frase "menos es más", un lema de los arquitectos y diseñadores modernistas.

Less pertenece a la clase de programas llamados *pagadores*, programas que permiten la visualización fácil de documentos de texto largos página por página. Mientras que el programa más solo podía avanzar la página, el programa menos permite la paginación tanto hacia adelante como hacia atrás y también tiene muchas otras características.

## Una visita guiada

El diseño del sistema de archivos en su sistema Linux es muy similar al que se encuentra en otros sistemas similares a Unix. De hecho, el diseño se especifica en un estándar publicado llamado Estándar de jerarquía del sistema de archivos de Linux. No todas las distribuciones de Linux se ajustan exactamente al estándar, pero la mayoría se acercan bastante.

A continuación, vamos a deambular por el sistema de archivos nosotros mismos para ver qué hace que nuestro sistema Linux funcione. Esto te dará la oportunidad de practicar tus habilidades de navegación. Una de las cosas que descubriremos es que muchos de los archivos interesantes están en texto plano y legible por humanos. A medida que avanzamos en nuestro recorrido, intente lo siguiente:

1. cd en un directorio dado.
2. Enumere el contenido del directorio con ls -l.
3. Si ve un archivo interesante, determine su contenido con file.
4. Si parece que podría ser texto, intente verlo con menos.

**Nota:** ¡Recuerda el truco de copiar y pegar! Si está utilizando un mouse, puede hacer doble clic en un nombre de archivo para copiarlo y hacer clic con el botón central para pegarlo en los comandos.

Mientras paseamos, no tengas miedo de mirar las cosas. Los usuarios habituales tienen prohibido en gran medida estropear las cosas. ¡Ese es el trabajo del administrador del sistema! Si un comando se queja de algo, simplemente pasa a otra cosa. Dedica un tiempo a mirar a tu alrededor. El sistema es nuestro para explorarlo.

Recuerda, ¡en Linux no hay secretos!

La Tabla 3-4 enumera solo algunos de los directorios que podemos explorar. ¡Siéntete libre de probar más!

Tabla 3-4: Directorios encontrados en sistemas Linux

Directorio	Comentarios
/	El directorio raíz, donde todo comienza.
/boot	Contiene binarios (programas) que deben estar presentes para que el sistema arranque y se ejecute.
/boot	Contiene el kernel de Linux, la imagen de disco RAM inicial (para los controladores necesarios en el momento del arranque) y el cargador de arranque.
	Archivos interesantes: <ul style="list-style-type: none"><li>• /boot/grub/grub.conf o menu.lst, que se utilizan para configurar el cargador de arranque</li><li>• /boot/vmlinuz, el kernel de Linux</li></ul>

*(continuación)*

Cuadro 3-4 (*continuación* )

Directorio	Comentarios
/Dev	Este es un directorio especial que contiene <i>Nodos de dispositivo</i> . "Todo es un archivo" también se aplica a los dispositivos. Aquí es donde el kernel mantiene una lista de todos los dispositivos que entiende.
/etcetera	El /etcetera contiene todos los archivos de configuración de todo el sistema. También contiene una colección de scripts de shell que inician cada uno de los servicios del sistema en el momento del arranque. Todo en este directorio debe ser texto legible.
	Archivos interesantes: Si bien todo en /etc es interesante, estos son algunos de mis favoritos de todos los tiempos: <ul style="list-style-type: none"> <li>• /etc/crontab, un fichero que define cuándo se ejecutarán los trabajos automatizados</li> <li>• /etc/fstab, una tabla de dispositivos de almacenamiento y sus puntos de montaje asociados</li> <li>• /etc/passwd, una lista de las cuentas de usuario</li> </ul>
/hogar	En configuraciones normales, a cada usuario se le asigna un directorio en /hogar. Los usuarios comunes solo pueden escribir archivos en sus directorios de inicio. Esta limitación protege el sistema de la actividad errática del usuario.
/Lib	Contiene archivos de biblioteca compartidos utilizados por los programas principales del sistema.
	Son similares a los archivos DLL de Windows.
/perdido+encontrado Linux-	Cada partición o dispositivo formateado que usa un archivo de sistema, como ext3, tendrá este directorio. Se utiliza en el caso de una recuperación parcial de un evento de corrupción del sistema de archivos. A menos que algo realmente malo haya ocurrido en su sistema, este directorio permanecerá vacío.
/Medio	En los sistemas Linux modernos, el /Medio contendrá los puntos de montaje para medios extraíbles como unidades USB, CD-ROM, etc. que se montan automáticamente en la inserción.
/mnt	En los sistemas Linux más antiguos, el método /mnt contiene puntos de montaje para dispositivos extraíbles que se han montado manualmente.

*/optar*

El */optar* se utiliza para instalar software "opcional". Esto se utiliza principalmente para almacenar productos de software comerciales que pueden estar instalados en su sistema.

---

Cuadro 3-4 (*continuación* )

Directorio	Comentarios
/Proc	El /Proc directorio es especial. No es un sistema de archivos real en el sentido de los archivos almacenados en su disco duro. Más bien, es un sistema de archivos virtual mantenido por el kernel de Linux. Los "archivos" que contiene son mirillas en el propio núcleo. Los archivos son legibles y le darán una imagen de cómo el kernel ve su computadora.
/raíz	Este es el directorio de inicio de la cuenta raíz.
/sbin	Este directorio contiene binarios del "sistema". Estos son programas que realizan tareas vitales del sistema que generalmente están reservadas para el superusuario.
/Tmp	El /Tmp El directorio está diseñado para el almacenamiento de archivos temporales y transitorios creados por varios programas. Algunas condecoraciones hacen que este directorio se vacíe cada vez que se reinicia el sistema.
/Usr	El /Usr El árbol de directorios es probablemente el más grande en un sistema Linux. Contiene todos los programas y archivos de soporte utilizados por los usuarios habituales.
/usr/bin	/usr/bin contiene los programas ejecutables instalados por su distribución de Linux. No es raro que este directorio contenga miles de programas.
/usr/lib	Las bibliotecas compartidas para los programas en /usr/bin.
/usr/local	El /usr/local árbol es donde los programas que no son incluidos con su distribución, pero están destinados para su uso en todo el sistema. Los programas compilados a partir del código fuente normalmente se instalan en /usr/local/bin. En un sistema Linux recién instalado, este árbol existe, pero estará vacío hasta que el administrador del sistema ponga algo en él.
/usr/sbin	Contiene más programas de administración del sistema.
/usr/compartir por	/usr/compartir contiene todos los datos compartidos utilizados por los programas en /usr/bin. Esto incluye cosas como archivos de configuración predeterminados, iconos, fondos de pantalla, archivos de sonido, etc.
/usr/compartir/doc algunos	La mayoría de los paquetes instalados en el sistema incluirán tipo de documentación. En /usr/share/doc,

encontraremos los archivos de documentación organizados por paquetes.

(*continuación*)

Cuadro 3-4 (*continuación* )

Directorio	Comentarios
/Var	Con la excepción de /Tmp y /hogar, los directorios que hemos examinado hasta ahora permanecen relativamente estáticos; es decir, sus contenidos no cambian. El /Var El árbol de directorios es donde se almacenan los datos que probablemente cambien. Aquí se encuentran varias bases de datos, archivos de cola de impresión, correo de usuario, etc.
/var/registro sistemas	/var/registro Contiene <i>Archivos de registro</i> , registros de varios actividad. Son muy importantes y deben ser monitoreados de vez en cuando. El más útil es /var/log/messages. Tenga en cuenta que, por razones de seguridad en algunos sistemas, debe ser el superusuario para ver los archivos de registro.

## Enlaces simbólicos

Al mirar a nuestro alrededor, es probable que veamos una lista de directorio con una entrada como esta:

---

```
lrwxrwxrwx 1 raíz raíz 11 2012-08-11 07:34 libc.so.6 -> libc-2.6.so
```

---

Observe cómo la primera letra de la lista es l y la entrada parece ¿Tiene dos nombres de archivo? Este es un tipo especial de archivo llamado *enlace simbólico* (también conocido como *enlace suave* o *enlace simbólico*). En la mayoría de los sistemas tipo Unix es posible tener un archivo referenciado por varios nombres. Si bien el valor de esto puede no ser obvio ahora, es realmente una característica útil.

Imagínese este escenario: Un programa requiere el uso de un recurso compartido de algún tipo contenido en un archivo llamado *foo*, pero *foo* tiene cambios de versión frecuentes. Sería bueno incluir el número de versión en el nombre del archivo para que el administrador u otra parte interesada pueda ver qué versión de *foo* está instalada. Esto presenta un problema. Si cambiamos el nombre de la propiedad compartida recurso, tenemos que rastrear todos los programas que puedan usarlo y cambiarlo para buscar un nuevo nombre de recurso cada vez que se instala una nueva versión del recurso. Eso no suena divertido en absoluto.

Aquí es donde los enlaces simbólicos salvan el día. Digamos que instalamos la versión 2.6 de *foo*, que tiene el nombre de archivo *foo-2.6*, y luego creamos un enlace simbólico llamado *simplemente foo* que apunta a *foo-2.6*. Esto significa que cuando un programa abre el archivo *foo*, en realidad está abriendo el archivo *foo-2.6*. Ahora todo el mundo está contento. Los programas que dependen de *foo* pueden encontrarlo, y aún podemos ver qué versión real está instalada. Cuando llega el momento de actualizar a *foo-2.7*, simplemente agregamos el archivo a nuestro sistema, eliminamos el

enlace simbólico *foo* y creamos uno nuevo que apunte a la nueva versión. Esto no solo resuelve el problema de la actualización de la versión, sino que también nos permite mantener ambas versiones en nuestra máquina. Imagínate que *foo-2.7* tiene un error (¡malditos sean esos desarrolladores!) y tenemos que volver a lo antiguo

Versión. De nuevo, simplemente eliminamos el enlace simbólico que apunta a la nueva versión y creamos un nuevo enlace simbólico que apunta a la versión anterior.

La lista de directorios anterior (del directorio `/lib` de un sistema Fedora) muestra un enlace simbólico llamado `libc.so.6` que apunta a un archivo de biblioteca compartida llamado `libc-2.6.so`. Esto significa que los programas que buscan `libc.so.6` realmente obtendrán el archivo `libc-2.6.so`.

Aprenderemos cómo crear enlaces simbólicos en el próximo capítulo.

## ENLACE DUROS

Ya que estamos en el tema de los enlaces, debemos mencionar que existe un segundo tipo de enlace llamado *enlace duro*. Los enlaces físicos también permiten que los archivos tengan varios nombres, pero lo hacen de una manera diferente. Hablaremos más sobre las diferencias entre enlaces simbólicos y duros en el próximo capítulo.



# 4

## MANIPULACIÓN DE ARCHIVOS Y DIRECTORIOS

En este punto, ¡estamos listos para un trabajo real!  
En este capítulo se presentarán los siguientes comandos:

- cp: copiar archivos y directorios.
- mv: mover/cambiar el nombre de archivos y directorios.
- mkdir: crea directorios.
- rm: elimina archivos y directorios.
- ln: crea vínculos físicos y simbólicos.

Estos cinco comandos se encuentran entre los comandos de Linux más utilizados. Se utilizan para manipular tanto archivos como directorios.

Ahora, para ser francos, algunas de las tareas realizadas por estos comandos se realizan más fácilmente con un administrador de archivos gráfico. Con un gestor de archivos podemos arrastrar y soltar un archivo de un directorio a otro, cortar y pegar archivos, eliminar archivos, etc. Entonces, ¿por qué usar estos viejos programas de línea de comandos?

La respuesta es potencia y flexibilidad. Si bien es fácil realizar manipulaciones de archivos simples con un administrador de archivos gráfico, las tareas complicadas pueden ser más fáciles con los programas de línea de comandos. Por ejemplo, ¿cómo podríamos copiar todos los archivos HTML de un directorio a otro, pero solo aquellos que no existen en el directorio de destino o que son más recientes que las versiones del directorio de destino? Bastante difícil con un administrador de archivos. Bastante fácil con la línea de comandos:

---

```
cp -u *.html destino
```

---

## Comodín

Antes de comenzar a usar nuestros comandos, debemos hablar sobre la característica del shell que hace que estos comandos sean tan poderosos. Debido a que el shell usa tanto los nombres de archivo, proporciona caracteres especiales para ayudarlo a especificar rápidamente grupos de nombres de archivo. Estos caracteres especiales se denominan *comodines*. El uso de comodines (también conocidos como *globbing*) le permite seleccionar nombres de archivo en función de patrones de caracteres. En la tabla 4-1 se enumeran los comodines y lo que seleccionan.

Tabla 4-1: Comodines

Comodín	Partidos
*	Cualquier carácter
?	Cualquier carácter individual
[Caracteres]	Cualquier carácter que sea miembro del conjunto
Caracteres [!Caracteres]	Cualquier carácter que no sea miembro del conjunto
Caracteres [:clase:]	Cualquier carácter que sea miembro de la clase especificada <i>clase</i>

En la tabla 4-2 se enumeran las clases de caracteres más utilizadas.

Tabla 4-2: Clases de caracteres de uso común

Carácter Clase	Partidos
[:alnum:]	Cualquier carácter alfanumérico
[:alfa:]	Cualquier carácter alfabético
[:d igit:]	Cualquier numeral
[:inferior:]	Cualquier letra minúscula
[:superior:]	Cualquier letra mayúscula

El uso de comodines permite construir criterios de selección muy sofisticados para los nombres de archivo. En la Tabla 4-3 se enumeran algunos ejemplos de patrones y con qué coinciden.

Tabla 4-3: Ejemplos de comodines

Patrón	Partidos
*	Todos los archivos
g*	Cualquier archivo que comience con <i>g</i>
b*.txt	Cualquier archivo que comience con <i>b</i> seguido de cualquier carácter y terminando con <i>.Txt</i>
???	Cualquier archivo que comience con <i>Datos</i> seguido exactamente de tres caracteres
[ABC]*	Cualquier archivo que comience con cualquiera de los siguientes archivos <i>un, bo c</i>
COPIA DE SEGURIDAD. [0-9][0-9][0-9] DE SEGURIDAD. seguido de	Cualquier archivo que comience con <i>COPIA DE SEGURIDAD.</i> seguido de exactamente tres números
[:superior:]* mayúscula	Cualquier archivo que comience con una letra
[! [:d igit:]]*	Cualquier archivo que no comience con un número
*[:inferior:]123]	Cualquier archivo que termine con una letra minúscula o el Números 1, 2o 3

Los comodines se pueden usar con cualquier comando que acepte nombres de archivo como argumentos, pero hablaremos más sobre eso en el Capítulo 7.

## RANGOS DE CHARACTE R

Si vienes de otro entorno similar a Unix o has estado leyendo otros libros sobre este tema, es posible que te hayas encontrado con las notaciones de rango de caracteres [A-Z] o [a-z]. Estas son notaciones tradicionales de Unix y también funcionaban en versiones anteriores de Linux. Todavía pueden funcionar, pero hay que tener mucho cuidado con ellos porque no producirán los resultados esperados a menos que se configuren correctamente. Por ahora, debe evitar usarlos y usar clases de caracteres en su lugar.

## LOS COMODINES TAMBIÉN FUNCIONAN EN LA INTERFAZ GRÁFICA DE USUARIO

Los comodines son especialmente valiosos, no solo porque se usan con tanta frecuencia en la línea de comandos, sino también porque son compatibles con algunos administradores de archivos gráficos.

- En **Nautilus** (el administrador de archivos de GNOME), puede seleccionar archivos usando Editar □ Seleccionar patrón. Simplemente ingrese un patrón de selección de archivos con comodines, y los archivos en el directorio visto actualmente se resaltarán para su selección.
- En algunas versiones de **Dolphin** y **Konqueror** (los administradores de archivos para KDE), puede introducir comodines directamente en la barra de direcciones. Por ejemplo, si desea ver todos los archivos que comienzan con una u minúscula en el directorio `/usr/bin`, escriba `/usr/bin/u*` en la barra de direcciones y se mostrará el resultado.

Muchas ideas que se encuentran originalmente en la interfaz de línea de

## **mkdir: crear directorios**

El comando `mkdir` se utiliza para crear directorios. Funciona de la siguiente manera:

`mkdir...`

**Una nota sobre la notación:** En este libro, cuando tres puntos siguen a un argumento en la descripción de un comando (como arriba), significa que el argumento puede repetirse; así, en este caso,

`mkdir dir1`

crearía un único directorio llamado `dir1`, mientras que

`mkdir dir1 dir2 dir3`

crearía tres directorios denominados `dir1`, `dir2` y `dir3`.

## **cp: copiar archivos y directorios**

El comando `cp` copia archivos o directorios. Se puede utilizar de dos maneras diferentes:

`CP ARTÍCULO1 ARTÍCULO2`

Para copiar el archivo único o el directorio `item1` en el archivo o directorio `item2` y:

`CP artículo... directorio`

para copiar varios elementos (ya sean archivos o directorios) en un directorio.

En las tablas 4-4 y 4-5 se enumeran algunas de las opciones más utilizadas (la opción corta y la opción larga equivalente) para `cp`.

**Tabla 4-4: Opciones cp**

Opción	Significado
<code>-a, --archivo</code>	Copie los archivos y directorios y todos sus atributos, incluyendo propiedades y permisos. Normalmente, las copias toman los atributos predeterminados del usuario que realiza la copia.
<code>-Yo--interactivo</code>	Antes de sobrescribir un archivo existente, solicite al usuario confirmación. Si no se especifica esta opción, <code>cp</code> sobrescribirá silenciosamente los archivos.
<code>-r, --recursivo</code>	Copie directorios y su contenido de forma recursiva. Éste (o la opción <code>-a</code> ) es necesaria al copiar directorios.
<code>-u, --actualización</code>	Al copiar archivos de un directorio a otro, copie Solo los archivos que no existen o que son más recientes que los archivos correspondientes existentes en el directorio de destino.
<code>-v, --verbose</code>	Mostrar mensajes informativos a medida que se realiza la copia.

**Tabla 4-5: Ejemplos de cp**

Mandar	Resultados
<code>Cp archivo1 archivo2</code>	Copiar <i>archivo1</i> Para <i>Archivo2</i> . Si <i>Archivo2</i> existe, se sobrescribe con el contenido de <i>file1</i> . Si <i>file2</i> no existe, se crea.
<code>Cp -i archivo1 archivo2</code>	Igual que el anterior, excepto que si <i>Archivo2</i> existe, el usuario es antes de que se sobreesciba.
<code>cp fichero1 fichero2 dir1 mosto</code>	Copiar <i>archivo1</i> y <i>Archivo2</i> en el directorio <i>dir1</i> . <i>dir1</i> ya existen.
<code>cp dir1/* dir2</code>	Usando un comodín, todos los archivos en <i>dir1</i> se copian en <i>dir2</i> . <i>dir2</i> ya debe existir.
<code>cp -r dir1 dir2</code>	Copiar directorio <i>dir1</i> (y su contenido) al directorio <i>dir2</i> . Si el directorio <i>dir2</i> no existe, se crea y contendrá el mismo contenido que el directorio <i>dir1</i> .

## **mv: mover y cambiar el nombre de los archivos**

El comando mv realiza tanto el movimiento de archivos como el cambio de nombre de archivos, dependiendo de cómo se utilice. En cualquier caso, el nombre de archivo original ya no existe después de la operación. MV se usa de la misma manera que CP:

MV Artículo1 Artículo2

para mover o cambiar el nombre del archivo o directorio *item1* a *item2* o

MV *artículo...* *directorio*

para mover uno o más elementos de un directorio a otro.

mv comparte muchas de las mismas opciones que cp, como se muestra en las Tablas 4-6 y 4-7.

**Tabla 4-6: Opciones de mv**

Opción	Significado
-Yo--interactivo	Antes de sobrescribir un archivo existente, solicite al usuario confirmación. Si no se especifica esta opción, mv sobrescribirá los archivos de forma silenciosa.
-u, --actualización	Al mover archivos de un directorio a otro, mueva Solo los archivos que no existen en el directorio de destino o que son más recientes que los archivos correspondientes existentes en el directorio de destino.
-v, --verbose movimiento	Mostrar mensajes informativos a medida que se realiza el Realizado.

**Tabla 4-7: Ejemplos de mv**

Mandar	Resultados
Mv archivo1 archivo2	Mover <i>archivo1</i> Para <i>Archivo2</i> . Si <i>Archivo2</i> existe, se sobrescribe con el contenido de <i>file1</i> . Si <i>file2</i> no existe, se crea. En cualquier caso, <i>file1</i> deja de existir.
Mv -i archivo1 archivo2	Igual que el anterior, excepto que si <i>Archivo2</i> existe, el usuario es antes de que se sobrescriba.
Mv archivo1 archivo2 dir1	Mover <i>archivo1</i> y <i>Archivo2</i> en el directorio <i>dir1</i> . <i>dir1</i> ya existen.
MV dir1 dir2	Mover directorio <i>dir1</i> (y su contenido) en el directorio <i>dir2</i> . Si el directorio <i>dir2</i> no existe, cree el directorio <i>dir2</i> , mueva el contenido del directorio <i>dir1</i> a <i>dir2</i> y

elimine el directorio *dir1*.

---

## rm: eliminar archivos y directorios

El comando rm se utiliza para eliminar (eliminar) archivos y directorios, de la siguiente manera:

```
rm artículo...
```

donde *item* es el nombre de uno o más archivos o directorios.

### ¡SÉ COCHE EFU L CON RM!

Los sistemas operativos tipo Unix, como Linux, no tienen un comando de recuperación. Una vez que eliminas algo con rm, se va. Linux asume que eres inteligente y que sabes lo que estás haciendo.

Tenga especial cuidado con los comodines. Consideremos este ejemplo clásico. Supongamos que desea eliminar solo los archivos HTML de un directorio. Para ello, escriba:

```
rm *.html
```

lo cual es correcto, pero si accidentalmente coloca un espacio entre el \* y el .html así:

```
rm * .html
```

El comando rm eliminará todos los archivos del directorio y luego se quejará de que no hay ningún archivo llamado .html.

**Aquí hay un consejo útil:** Siempre que use comodines con rm (¡además de verificar cuidadosamente su escritura!), Pruebe el comodín primero con ls. Esto le permitirá ver los archivos que se eliminarán. Luego presione la tecla de flecha hacia arriba para recuperar el comando y reemplazar ls por rm.

En las tablas 4-8 y 4-9 se enumeran algunas de las opciones comunes para rm.

Tabla 4-8: Opciones de rm

Opción	Significado
-Yo--interactivo	Antes de eliminar un archivo existente, solicite al usuario confirmación. Si no se especifica esta opción, rm eliminará los archivos de forma silenciosa.
-r, --recursivo	Eliminar directorios de forma recursiva. Esto significa que si
un	El directorio que se elimina tiene subdirectorios, elimínelos también. Para eliminar un directorio, se debe especificar esta opción.
-f, --forceignore	archivos inexistentes y no se le solicita. Éste anula el método --interactivo opción.
-v, --verbose	Mostrar mensajes informativos a medida que se realiza la eliminación

Realizado.

---

**Tabla 4-9: Ejemplos de rm**

Mandar	Resultados
Archivo RM1	Borrar <i>archivo1</i> silenciosamente.
micrómetro -Yo archivo1	Antes de eliminar <i>archivo1</i> , solicite al usuario confirmación.
rm -r fichero1 dir1	Borrar <i>archivo1</i> y <i>dir1</i> y sus contenidos.
micrómetro-Archivo RF1 dir1	Igual que el anterior, excepto que si alguno de los dos <i>archivo1</i> o <i>dir1</i> no existe, RM continuará en silencio.

## In: crear vínculos

El comando *ln* se utiliza para crear enlaces físicos o simbólicos. Se utiliza de dos maneras:

*Enlace al archivo LN*

para crear un enlace físico y

*ln -s enlace de elemento*

para crear un enlace simbólico donde *item* es un archivo o un directorio.

### *Enlaces duros*

Los enlaces duros son la forma original de Unix de crear enlaces; Los enlaces simbólicos son más modernos. De forma predeterminada, cada archivo tiene un único enlace físico que le da nombre al archivo. Cuando creamos un enlace físico, creamos una entrada de directorio adicional para un archivo. Los enlaces duros tienen dos limitaciones importantes:

- Un enlace duro no puede hacer referencia a un archivo fuera de su propio sistema de archivos. Esto significa que un enlace no puede hacer referencia a un archivo que no esté en la misma parte del disco que el propio enlace.
- Un enlace físico no puede hacer referencia a un directorio.

Un enlace físico es indistinguible del archivo en sí. A diferencia de una lista de directorios que contiene un enlace simbólico, una lista de directorios que contiene un enlace físico no muestra ninguna indicación especial del enlace. Cuando se elimina un vínculo físico, se elimina el vínculo, pero el contenido del archivo en sí continúa existiendo (es decir, su espacio no se desasigna) hasta que se eliminan todos los vínculos al archivo.

Es importante tener en cuenta los enlaces duros porque es posible que los encuentre de vez en cuando, pero la práctica moderna prefiere los enlaces simbólicos, de los que hablaremos a continuación.

### *Enlaces simbólicos*

Los enlaces simbólicos se crearon para superar las limitaciones de los enlaces físicos. Los enlaces simbólicos funcionan creando un tipo especial de archivo que contiene un puntero de texto

al archivo o directorio al que se hace referencia. En este sentido, funcionan de la misma manera que un acceso directo de Windows, aunque, por supuesto, son anteriores a la función Windows por muchos años. ;-)

Un archivo señalado por un enlace simbólico y el propio enlace simbólico son en gran medida indistinguibles entre sí. Por ejemplo, si escribe algo en el enlace simbólico, también se escribe en el archivo al que se hace referencia. Sin embargo, cuando se elimina un enlace simbólico, solo se elimina el enlace, no el archivo en sí. Si el archivo se elimina antes del enlace simbólico, el enlace seguirá existiendo, pero no apuntará a nada. En este caso, se dice que el enlace está *roto*. En muchas implementaciones, el comando ls mostrará los enlaces rotos en un color distintivo, como el rojo, para revelar su presencia.

El concepto de enlaces puede parecer confuso, pero persiste. Vamos a probar todo esto y, con suerte, se aclarará.

## Construyamos un patio de recreo

Ya que vamos a hacer una manipulación de archivos real, construyamos un lugar seguro para "jugar" con nuestros comandos de manipulación de archivos. Primero necesitamos un directorio para trabajar. Crearemos uno en nuestro directorio de inicio y lo llamaremos *playground*.

### Creación de directorios

El comando mkdir se utiliza para crear un directorio. Para crear nuestro *directorio de playground*, primero nos aseguraremos de que estamos en nuestro directorio de inicio y luego crearemos el nuevo directorio:

---

```
[me@linuxbox ~]$ cd  
[me@linuxbox ~]$ mkdir patio de juegos
```

---

Para hacer el *patio de recreo* un poco más interesante, vamos a crear un par de directorios dentro de él llamados *dir1* y *dir2*. Para ello, cambiaremos nuestro directorio de trabajo actual a *playground* y ejecutaremos otro mkdir:

---

```
[me@linuxbox ~]$ CD Patio de juegos  
[me@linuxbox patio de juegos]$ mkdir dir1 dir2
```

---

Tenga en cuenta que el comando mkdir aceptará múltiples argumentos, lo que nos permite crear ambos directorios con un solo comando.

### Copia de archivos

A continuación, introduzcamos algunos datos en nuestro patio de recreo. Lo haremos copiando un archivo. Usando el comando cp, copiaremos el archivo *passwd* del directorio */etc* al directorio de trabajo actual.

---

```
[me@linuxbox patio de recreo]$ cp /etc/passwd .
```

---

Observe cómo usamos la abreviatura para el directorio de trabajo actual, el período final único. Así que ahora si realizamos un ls, veremos nuestro archivo:

---

```
[me@linuxbox parque infantil]$ ls -l
Total 12
Drxervser-s 2m m 4096 2012-01-10 16:40 dir1
Drxervser-s 2m m 4096 2012-01-10 16:40 dir2
-rw-r--r-- 1 me me 1650 2012-01-10 16:07 passwd
```

---

Ahora, solo por diversión, repitamos la copia usando la opción **-v** (verbose) para ver qué hace:

---

```
[me@linuxbox patio de recreo]$ cp -v /etc/passwd .
'/etc/passwd' -> './passwd'
```

---

El comando cp volvió a realizar la copia, pero esta vez mostró un mensaje conciso que indicaba qué operación estaba realizando. Observe que cp sobrescribió la primera copia sin ninguna advertencia. De nuevo, este es un caso de cp asumiendo que sabes lo que estás haciendo. Para obtener una advertencia, incluiremos la opción **-i** (interactiva):

---

```
[me@linuxbox patio de recreo]$ cp -i /etc/passwd .
CP: ¿Sobrescribir './passwd'?
```

---

Responder a la solicitud introduciendo un y hará que el archivo se sobrescriba; cualquier otro carácter (por ejemplo, n) hará que cp deje el archivo en paz.

### ***Mover y cambiar el nombre de archivos***

Ahora bien, el nombre *passwd* no parece muy juguetón y esto es un patio de recreo, así que cambiémoslo por otra cosa:

---

```
[me@linuxbox parque infantil]$ MV Passwd Diversión
```

---

Pasemos un poco la diversión moviendo nuestro archivo renombrado a cada uno de los directorios y viceversa:

---

```
[me@linuxbox Patio de juegos]$ MV diversión dir1
```

---

Lo mueve primero al directorio *dir1*. Entonces

---

```
[me@linuxbox Playground]$ MV dir1/fun dir2
```

---

Lo mueve de *dir1* a *dir2*. Entonces

---

```
[me@linuxbox parque infantil]$ MV dir2/diversión .
```

---

finalmente lo devuelve al directorio de trabajo actual. A continuación, veamos el efecto de mv en los directorios. Primero moveremos nuestro fichero de datos a *dir1* de nuevo:

---

```
[me@linuxbox Patio de juegos]$ MV diversión dir1
```

---

Y luego mueva *dir1* a *dir2* y confírmelo con ls:

---

```
[me@linuxbox Patio de juegos]$ MV dir1
dir2 [me@linuxbox Patio de juegos]$ ls -l
dir2 Total 4
drwxrwxr-x 2 me me4096 2012-01-11 06:06 dir1
[me@linuxbox playground]$      ls -l dir2/dir1
Total 4
-rw-r--r-- 1 yo yo      1650 2012-01-10 16:33 diversión
```

---

Tenga en cuenta que debido a que *dir2* ya existía, mv movió *dir1* a *dir2*. Si *dir2* no hubiera existido, mv habría cambiado el nombre de *dir1* a *dir2*. Por último, volvamos a poner todo:

---

```
[me@linuxbox patio de recreo]$ MV dir2/dir1 .
[me@linuxbox parque infantil]$ MV dir1/diversión .
```

---

### *Creación de enlaces duros*

Ahora vamos a probar algunos enlaces. Primero los enlaces duros: Crearemos algunos enlaces a nuestro archivo de datos de la siguiente manera:

---

```
[me@linuxbox Playground]$ ln Diversión Diversión-Difícil
[me@linuxbox Zona de juegos]$ ln Diversión
dir1/diversión-difícil [me@linuxbox Zona de juegos]$ ln
Diversión dir2/Diversión-Difícil
```

---

Así que ahora tenemos cuatro instancias de la diversión de archivos. Echemos un vistazo a nuestro *Directorio de juegos* infantiles:

---

```
[me@linuxbox parque infantil]$ ls -l
Total 16
drwxrwxr-x 2 me    me    4096 14/01/2012 16:17 dir1
drwxrwxr-x 2 me    me    4096 14/01/2012 16:17 dir2
-rw-r--r-- 4 yo    me    1650 2012-01-10 16:33 diversión
-rw-r--r-- 4 yo    me    1650 2012-01-10 16:33
```

---

Una cosa que observa es que el segundo campo de la lista para *fun* y *fun-hard* contiene un 4, que es el número de enlaces físicos que ahora existen para el archivo. Recordarás que un archivo siempre tendrá al menos un enlace porque el nombre del archivo se crea mediante un enlace. Entonces, ¿cómo sabemos que *fun* y *fun-hard* son, de hecho, el mismo archivo? En este caso, ls no es muy útil. Si bien podemos ver que *fun* y *fun-hard* tienen el mismo tamaño (campo 5), nuestra lista no proporciona forma de estar seguro de que sean el mismo archivo. Para resolver este problema, vamos a tener que indagar un poco más.

Al pensar en enlaces físicos, es útil imaginar que los archivos se componen de dos partes: la parte de datos que contiene el contenido del archivo y la parte del nombre, que contiene el nombre del archivo. Cuando creamos enlaces físicos, en realidad estamos creando partes de nombre adicionales que se refieren a la misma parte de datos. El sistema asigna una cadena de bloques de disco a lo que se denomina un *inodo*, que luego se asocia con la parte del nombre. Por lo tanto, cada enlace físico se refiere a un inodo específico que contiene el contenido del archivo.

El comando `ls` tiene una forma de revelar esta información. Se invoca con la opción `-i`:

---

```
[me@linuxbox parque infantil]$ ls -li
Total 16
12353539 drwxrwxr-x 2 me    me   4096 14/01/2012 16:17 dir1
12353540 drwxrwxr-x 2 me    me   4096 14/01/2012 16:17 dir2
12353538 -rw-r--r-- 4 yo    me   1650 2012-01-10 16:33 diversión
12353538 -rw-r--r-- 4 yo    me   1650 2012-01-10 16:33
```

---

En esta versión de la lista, el primer campo es el número de inodo y, como podemos ver, tanto *fun* como *fun-hard* comparten el mismo número de inodo, lo que confirma que son el mismo archivo.

### **Creación de enlaces simbólicos**

Los enlaces simbólicos se crearon para superar las dos desventajas de los enlaces físicos: los enlaces físicos no pueden abarcar dispositivos físicos, y los enlaces físicos no pueden hacer referencia a directorios, solo a archivos. Los enlaces simbólicos son un tipo especial de archivo que contiene un puntero de texto al archivo o directorio de destino.

La creación de enlaces simbólicos es similar a la creación de enlaces físicos:

---

```
[me@linuxbox parque infantil]$ ln -s diversión fun-
sym [me@linuxbox parque infantil]$ ln -s .. /fun
dir1/fun-sym [me@linuxbox patio de recreo]$ ln -s ..
/fun dir2/fun-sym
```

---

El primer ejemplo es bastante sencillo: simplemente añadimos la opción `-s` para crear un enlace simbólico en lugar de un enlace duro. Pero, ¿qué pasa con los dos siguientes? Recuerde, cuando creamos un enlace simbólico, estamos creando una descripción de texto de dónde se encuentra el archivo de destino en relación con el enlace simbólico. Es más fácil de ver si nos fijamos en la salida `ls`:

---

```
[me@linuxbox patio de recreo]$ ls -l dir1
Total 4
-RW-R--r-- 4 me    me   1650 2012-01-10 16:33
lrwxrwxrwx 1 me    me      6 2012-01-15 15:17 fun-sym ->... /diversión
```

---

La lista de *fun-sym* en *dir1* muestra que es un enlace simbólico por la 1 inicial en el primer campo y el hecho de que apunta a .. */fun*, lo cual es correcto.

En relación con la ubicación de *fun-sym*, *fun* está en el directorio de arriba. Tenga en cuenta también que la longitud del archivo de enlace

simbólico es 6, el número de caracteres en la cadena .. /fun en lugar de la longitud del archivo al que apunta.

Al crear enlaces simbólicos, puede usar nombres de ruta absolutos, como este:

---

```
[me@linuxbox playground]$ ln -s /home/me/playground/fun dir1/fun-sym
```

---

o nombres de ruta relativos, como hicimos en nuestro ejemplo anterior. El uso de nombres de ruta relativos es más deseable porque permite que un directorio que contiene enlaces simbólicos sea renombrado y/o movido sin romper los enlaces.

Además de los archivos normales, los enlaces simbólicos también pueden hacer referencia a directorios:

---

```
[me@linuxbox patio de juegos]$ ln -s dir1 dir1-sym
[me@linuxbox parque infantil]$ ls -l
Total 16
drwxrwxr-x 2 me    me    4096 2012-01-15 15:17 dir1
lrwxrwxrwx 1 me    me      4 2012-01-16 14:45 dir1-wi -> dir1
drwxrwxr-x 2 me    me    4096 2012-01-15 15:17 dir2
-rw-r--r-- 4 yo   me    1650 2012-01-10 16:33 diversión
-rw-r--r-- 4 yo   me    1650 2012-01-10 16:33
lrwxrwxrwx 1 me    me      3 2012-01-15 15:15 fun-sym -> diversión
```

---

## **Eliminación de archivos y directorios**

Como cubrimos anteriormente, el comando rm se utiliza para eliminar archivos y directorios. Vamos a usarlo para limpiar un poco nuestro patio de recreo. Primero, eliminemos uno de nuestros enlaces físicos:

---

```
[me@linuxbox patio de recreo]$ rm divertido-duro
[me@linuxbox parque infantil]$ ls -l
Total 12
drwxrwxr-x 2 me    me    4096 2012-01-15 15:17 dir1
lrwxrwxrwx 1 me    me      4 2012-01-16 14:45 dir1-wi -> dir1
drwxrwxr-x 2 me    me    4096 2012-01-15 15:17 dir2
-rw-r--r-- 3 yo   me    1650 2012-01-10 16:33 diversión
lrwxrwxrwx 1 me    me      3 2012-01-15 15:15 fun-sym -> diversión
```

---

Eso funcionó como se esperaba. El archivo *fun-hard* ha desaparecido y el número de enlaces que se muestra para *fun* se reduce de cuatro a tres, como se indica en el segundo campo de la lista del directorio. A continuación, eliminaremos el archivo *fun*, y solo por placer, incluiremos la opción *-i* para mostrar lo que hace:

---

```
[me@linuxbox patio de juegos]$ rm -i diversión
RM: ¿Eliminar el archivo regular 'fun'?
```

---

Escriba y en el símbolo del sistema y el archivo se eliminará. Pero veamos ahora la salida de ls . ¿Te das cuenta de lo que le pasó a *fun-sym*? Dado que es un enlace simbólico que apunta a un archivo que ahora no existe, el enlace está *roto*:

---

```
[me@linuxbox parque infantil]$ ls -l
Total 8
drwxrwxr-x 2 me    me    4096 2012-01-15 15:17 dir1
lrwxrwxrwx 1 me    me      4 2012-01-16 14:45 dir1-wi -> dir1
```

---

```
drwxrwxr-x 2 me    me   4096 2012-01-15 15:17 dir2
lwxrwxrwx 1 me    me     3 2012-01-15 15:15 fun-sym -> diversión
```

---

La mayoría de las distribuciones de Linux configuran ls para mostrar enlaces rotos. ¡En una caja Fedora, los enlaces rotos se muestran en texto rojo parpadeante! La presencia de un enlace roto no es peligrosa en sí misma, pero es bastante desordenada. Si intentamos usar un enlace roto, veremos esto:

---

```
[me@linuxbox patio de juegos]$ menos fun-sym  
fun-sym: No existe tal archivo o directorio
```

---

Limpiemos un poco. Eliminaremos los enlaces simbólicos:

---

```
[me@linuxbox playground]$ rm fun-sym dir1-sym  
[me@linuxbox parque infantil]$ ls -l  
Total 8  
drwxrwxr-x 2 me    me   4096 2012-01-15 15:17 dir1  
drwxrwxr-x 2 me    me   4096 2012-01-15 15:17 dir2
```

---

Una cosa que hay que recordar sobre los enlaces simbólicos es que la mayoría de las operaciones de archivos se llevan a cabo en el destino del enlace, no en el enlace en sí. Sin embargo, rm es una excepción. Cuando se elimina un vínculo, es el vínculo el que se elimina, no el destino.

Por último, retiraremos nuestro *parque infantil*. Para ello, volveremos a nuestro directorio de inicio y usaremos rm con la opción recursiva (-r) para eliminar *play-ground* y todo su contenido, incluyendo sus subdirectorios:

---

```
[me@linuxbox patio de recreo]$ cd  
[me@linuxbox ~]$ rm -r patio de recreo
```

---

### CREACIÓN DE ENLACES SIMBÓLICOS CON LA INTERFAZ GRÁFICA DE USUARIO

Los administradores de archivos tanto en GNOME como en KDE proporcionan un método fácil y automático para crear enlaces simbólicos. Con GNOME, mantener presionadas las teclas CTRL y SHIFT mientras arrastra un archivo creará un enlace en lugar de copiar (o mover) el archivo. En KDE, aparece un pequeño menú cada vez que se suelta un archivo, ofreciendo la

## Nota final

Hemos cubierto mucho terreno aquí, y la información puede tardar un tiempo en asimilarse por completo. Realice el ejercicio del patio de recreo una y otra vez hasta que tenga sentido. Es importante obtener una buena comprensión de los comandos básicos de manipulación de archivos y los comodines. Siéntase libre de ampliar el ejercicio del patio de recreo agregando más archivos y directorios, usando comodines para especificar archivos para diversas operaciones. El concepto de enlaces puede ser un poco confuso al principio, pero tómate el tiempo para aprender cómo funcionan. Pueden ser un verdadero salvavidas.

# 5

## TRABAJAR CON COMANDOS

Hasta este punto, hemos visto una serie de comandos misteriosos, cada uno con sus propias opciones y argumentos misteriosos. En este capítulo, intentaremos eliminar parte de ese misterio e incluso crear algunos de nuestros propios mandamientos. Los comandos introducidos en este capítulo son los siguientes:

- type: indica cómo se interpreta el nombre de un comando.
- which: muestra qué programa ejecutable se ejecutará.
- man: muestra la página de manual de un comando.
- apropos: muestra una lista de los comandos adecuados.
- info: muestra la entrada de información de un comando.
- whatis: muestra una descripción muy breve de un comando.
- alias: cree un alias para un comando.



# ¿Qué son exactamente los comandos?

Un comando puede ser una de estas cuatro cosas:

- **Un programa ejecutable** como todos esos archivos que vimos en `/usr/bin`. Dentro de esta categoría, los programas pueden ser *binarios compilados*, como programas escritos en C y C++, o programas escritos en *lenguajes de scripting*, como shell, Perl, Python, Ruby, etc.
- **Un comando integrado en el propio shell.** bash soporta una serie de comandos llamados internamente *Shell Builtins*. El comando `cd`, por ejemplo, es un shell incorporado.
- **Una función de shell.** Las *funciones de shell* son scripts de shell en miniatura incorporados al *entorno*. Cubriremos la configuración del entorno y la escritura de funciones de shell en capítulos posteriores, pero por ahora solo tenga en cuenta que existen.
- **Un alias.** Un *alias* es un comando que podemos definir nosotros mismos, construido a partir de otros comandos.

## Identificación de comandos

A menudo es útil saber exactamente cuál de los cuatro tipos de comandos se está utilizando, y Linux proporciona un par de formas de averiguarlo.

### `type: muestra el tipo de un comando`

El comando `type` es una orden interna de shell que muestra el tipo de comando que ejecutará el shell, dado un nombre de comando particular. Funciona de la siguiente manera:

Comando de tipo

donde *comando* es el nombre del comando que desea examinar. Estos son algunos ejemplos:

---

```
[me@linuxbox ~]$ type type  
type es un shell  
incorporado [  
me@linuxbox ~]$ type ls  
ls tiene un alias 'ls --color=tty'  
[me@linuxbox ~]$ type cp  
cp es /bin/cp
```

---

Aquí vemos los resultados de tres comandos diferentes. Nótese que el comando `ls` (tomado de un sistema Fedora) es en realidad un alias para el comando `ls` con la opción `--color=tty` añadida. ¡Ahora sabemos por qué la salida de `ls` se muestra en color!

## ***which: muestra la ubicación de un ejecutable***

A veces, hay más de una versión de un programa ejecutable instalada en un sistema. Si bien esto no es muy común en los sistemas de escritorio, no es inusual en servidores grandes. Para determinar la ubicación exacta de un ejecutable determinado, se utiliza el siguiente comando:

---

```
[me@linuxbox -]$ que ls  
/bin/ls
```

---

que funciona solo para programas ejecutables, no para incorporados o alias que son sustitutos de programas ejecutables reales. Cuando intentamos usar which en un shell builtin (por ejemplo, cd), no obtenemos ninguna respuesta o un mensaje de error:

---

```
[me@linuxbox -]$ que cd  
/effect/bin/which: no hay niños en (/opt/jare1.6.0_03/bin:/effect/lib/qut-  
3.3/bin:/effect/kerberos/bin:/opt/jare1.6.0_03/bin:/effect/lab/catch:/effect/local/  
bin:/effect/bin/bin:/home  
/me/bin)
```

---

Esta es una forma elegante de decir "comando no encontrado".

## **Obtener la documentación de un comando**

Con este conocimiento de lo que es un comando, ahora podemos buscar la documentación disponible para cada tipo de comando.

### ***help—Obtener ayuda para los Shell: Builtins***

bash tiene una función de ayuda incorporada para cada uno de los embellecidos del shell. Para usarlo, escriba

**help** seguido del nombre de la orden interna del shell. Por ejemplo:

---

```
[me@linuxbox -]$ cd de ayuda  
cd: cd [-L|-P] [dir]  
Cambio el directorio actual a DIR. La variable $HOME es el DIR predeterminado.  
La variable CDPATH define la ruta de búsqueda para el directorio que contiene  
DIR. Los nombres de directorio alternativos en CDPATH están separados por dos  
puntos (:). Un nombre de directorio nulo es el mismo que el directorio  
actual, es decir, '.'. Si DIR comienza con una barra diagonal (/), no se  
utiliza CDPATH. Si no se encuentra el directorio y se establece la opción  
de shell 'cdable_vars' , pruebe con la palabra como nombre de  
variable. Si esa variable tiene un valor, entonces cd al valor de esa variable.  
La opción -P dice que se use la estructura de directorio física en lugar de seguir  
enlaces simbólicos; la opción -L obliga a seguir enlaces simbólicos.
```

---

**Una nota sobre la notación:** cuando aparecen corchetes en la descripción de la sintaxis de un comando, indican elementos opcionales. Un carácter de barra vertical indica elementos mutuamente excluyentes. Un ejemplo es el comando cd anterior: `cd [-L|-P] [dir]`.

Esta notación dice que el comando cd puede ser seguido opcionalmente por un -L o un -P y además, opcionalmente seguido por el argumento dir.

Si bien la salida de la ayuda para el comando cd es concisa y precisa, de ninguna manera es un tutorial y, como podemos ver, ¡también parece mencionar muchas cosas de las que aún no hemos hablado! No te preocupes. Lo conseguiremos.

### **--help: muestra información de uso**

Muchos programas ejecutables soportan una opción --help que muestra una descripción de la sintaxis y las opciones soportadas del comando. Por ejemplo:

---

```
[me@linuxbox ~]$ mkdir --help
Uso: mkdir [OPCIÓN] DIRECTORIO...
Cree el/los Directorio(s), si aún no existen.
```

-Z, --context=CONTEXT (SELinux) establece el contexto de seguridad en CONTEXT Los argumentos obligatorios para las opciones largas también son obligatorios para las opciones cortas.  
-m, --modo=MODO Establecer archivo modo (como en chmod), no a = rwx - Umask  
-p, --padres No hay error si existente, haga directorios principales según sea necesario  
-v, --verbose Imprimir un mensaje para cada directorio creado  
 --Ayuda Mostrar esta ayuda y salir  
 --Versión información de la versión de salida  
y salida Notificar errores a <bug-coreutils@gnu.org>.

---

Algunos programas no admiten la opción --help, pero pruébelo de todos modos. A menudo, da como resultado un mensaje de error que revelará la misma información de uso.

### **man: muestra la página de manual de un programa**

La mayoría de los programas ejecutables destinados al uso de la línea de comandos proporcionan una documentación formal llamada *manual* o *página de manual*. Para verlos, se utiliza un programa de paginación especial llamado man , de la siguiente manera:

Programa MAN

donde *program* es el nombre del comando que se va a ver.

Las páginas de manual varían un poco en formato, pero generalmente contienen un título, una sinopsis de la sintaxis del comando, una descripción del propósito del comando y una lista y descripción de cada una de las opciones del comando. Las páginas de manual, sin embargo, no suelen incluir ejemplos, y pretenden ser una referencia, no un tutorial. A modo de ejemplo, intentemos ver la página del comando man para el comando ls:

---

```
[me@linuxbox ~]$ hombre ls
```

---

En la mayoría de los sistemas Linux, el hombre usa menos para mostrar la página de manual, por lo que todos los comandos menos conocidos funcionan mientras se muestra la página.

El "manual" que muestra el hombre está dividido en secciones y cubre

no solo los comandos de usuario, sino también los comandos de administración del sistema, las interfaces de programación, los formatos de archivo y más. En la Tabla 5-1 se describe el diseño del manual.

**Tabla 5-1: Organización de la página de manual**

Sección	Contenido
1	Comandos de usuario
2	Interfaces de programación para llamadas al sistema del kernel
3	Interfaces de programación para la biblioteca C
4	Archivos especiales, como nodos de dispositivos y controladores
5	Formatos de archivo
6	Juegos y diversiones como protectores de pantalla
7	Misceláneo
8	Comandos de administración del sistema

A veces necesitamos buscar en una sección específica del manual para encontrar lo que estamos buscando. Esto es particularmente cierto si estamos buscando un formato de archivo que también sea el nombre de un comando. Si no especificamos un número de sección, siempre obtendremos la primera instancia de una coincidencia, probablemente en la sección 1. Para especificar un número de sección, usamos `man` de la siguiente manera:

Sección de hombre *search\_term*

Por ejemplo:

---

```
[me@linuxbox -]$ hombre 5 passwd
```

---

Mostrará la página del manual que describe el formato de archivo del archivo */etc/passwd*.

### ***apropos: mostrar comandos apropiados***

También es posible buscar en la lista de páginas del manual posibles coincidencias en función de un término de búsqueda. Aunque rudimentario, este enfoque a veces es útil. A continuación, se muestra un ejemplo de una búsqueda de páginas de manual utilizando el término de búsqueda *floppy*:

---

```
[me@linuxbox -]$ apropos floppy
create_floppy_devices (8) - Llamada udev para crear todos los
                           dispositivo de disquete basado en el tipo CMOS
fdformat          (8) - Formateo de bajo nivel de un disquete
disquette         (8) - formatear disquetes
gfloppy          (1) - un formateador de disquete simple para los
mbablocks de GNOME (1) - prueba un disquete y marca el
                           bloques en el FAT
Formato m        (1) - añadir un sistema de archivos MSDOS a un
                           disquete formateado de bajo nivel
```

---

El primer campo de cada línea de salida es el nombre de la página del comando `man` y el segundo campo muestra la sección. Nótese que el comando `man` con la opción `-k` realiza exactamente la misma función que

apropos.

## ***whatis: muestra una descripción muy breve de un comando***

El programa whatis muestra el nombre y una descripción de una línea de una página de manual que coincide con una palabra clave especificada:

```
[me@linuxbox ~]$ ¿qué es ls  
ls (1) -lista Contenido del directorio
```

### **LA PÁGINA DE HOMBRE MÁS BRUTAL DE TODOS ELLOS**

Como hemos visto, las páginas de manual suministradas con Linux y otros sistemas similares a Unix están pensadas como documentación de referencia y no como tutoriales. Muchas páginas de manual son difíciles de leer, pero creo que el gran premio de la dificultad tiene que ir a la página de manual para bash. Mientras investigaba para este libro, le di una revisión cuidadosa para asegurarme de que estaba cubriendo la mayoría de sus temas. Cuando se imprime, tiene más de 80 páginas y es extremadamente denso, y su estructura no tiene absolutamente ningún sentido para un nuevo usuario.

Por otro lado, es muy preciso y conciso, además de ser extremadamente completo. Así que échale un vistazo si te atreves, y espera con ansias el día en que puedas leerlo y todo tenga sentido.

## ***info: muestra la entrada de información de un programa***

El Proyecto GNU proporciona una alternativa a las páginas de manual llamadas *páginas de información*. Las páginas de información se muestran con un programa de lectura llamado, apropiadamente, info. Las páginas de información tienen *hipervínculos* al igual que las páginas web. He aquí un ejemplo:

```
Archivo: coreutils.info, Nodo: invocación ls, Siguiiente: invocación de directorio, Subir: Listado de directorio
```

### **10.1 'ls': Lista de contenidos del directorio**

El programa 'ls' enumera información sobre archivos (de cualquier tipo, incluidos directorios). Las opciones y los argumentos de archivo se pueden mezclar arbitrariamente, como de costumbre.

En el caso de los argumentos de línea de comandos que no son de opción y que son directorios, de forma predeterminada, 'ls' enumera el contenido de los directorios, no de forma recursiva, y omite los archivos con nombres que comienzan con '.'. Para otros argumentos que no son de opción, de forma predeterminada 'ls' enumera solo el nombre del archivo. Si no se especifica ningún argumento que no sea de opción, 'ls' opera en el directorio actual, actuando como si se hubiera invocado con un solo argumento de '.'.

De forma predeterminada, la salida se ordena alfabéticamente, de acuerdo con el  
--zz-Info: (coreutils.info.gz)ls invocación, 63 líneas --Top-----

El programa de información lee los *archivos de información*, que están estructurados en árbol en *nodos individuales*, cada uno de los cuales contiene un solo tema. Los archivos de información contienen hipervínculos que pueden moverte de un nodo a otro. Un hipervínculo se puede identificar por su asterisco inicial y se activa colocando el cursor sobre él y presionando la tecla ENTER.

Para invocar info, escriba **info** seguido opcionalmente por el nombre de un programa. En la tabla 5-2 se enumeran los comandos utilizados para controlar el lector mientras se muestra una página de información.

**Tabla 5-2: Comandos info**

Mandar	Acción
?	Mostrar la ayuda de comandos.
SUBIR PÁGINA O RETROCESO	Mostrar la página anterior.
BAJAR PÁGINA o barra espaciadora	Mostrar página siguiente.
n	Siguiente: muestra el siguiente nodo.
p	Anterior: muestra el nodo anterior.
u	Arriba: muestra el nodo principal del nodo que se muestra actualmente, normalmente un menú.
ENTRAR	Siga el hipervínculo en la ubicación del cursor.
q	Renunciar.

La mayoría de los programas de línea de comandos que hemos discutido hasta ahora son parte del paquete coreutils del Proyecto GNU, por lo que puede encontrar más información sobre ellos escribiendo

---

```
[me@linuxbox ~]$ info coreutils
```

---

que mostrará una página de menú que contiene hipervínculos a la documentación de cada programa proporcionado por el paquete coreutils.

### ***README y otros archivos de documentación del programa***

Muchos paquetes de software instalados en su sistema tienen archivos de documentación que residen en el directorio */usr/share/doc*. La mayoría de estos se almacenan en formato de texto plano y se pueden ver con menos. Algunos de los archivos están en formato HTML y se pueden ver con un navegador web. Es posible que encontremos algunos archivos que terminan con una extensión *.gz*. Esto indica que se han comprimido con el programa de compresión *gzip*. El paquete *gzip* incluye una versión especial de less llamada *zless*, que mostrará el contenido de los archivos de texto comprimidos con *gzip*.

## Creación de sus propios comandos con alias

¡Ahora para nuestra primera experiencia con la programación! Crearemos un comando propio usando el comando alias. Pero antes de comenzar, debemos revelar un pequeño truco de línea de comandos. Es posible poner más de un comando en una línea separando cada comando con un carácter de punto y coma. Funciona de la siguiente manera:

*comando1; comando2; comando3...*

Este es el ejemplo que usaremos:

---

```
[me@linuxbox ~]$ cd /usr; ls; CD -
bote juegos Kerberos lib64 Compartir
localmente Tmp etc incluyen lib libexec sbin
Fuente
/inicio/yo
[me@linuxbox ~]$
```

---

Como podemos ver, hemos combinado tres comandos en una línea. Primero cambiamos el directorio a */usr*, luego listamos el directorio, y finalmente volvemos al directorio original (usando *cd -*) así que terminamos donde empezamos. Ahora vamos a convertir esta secuencia en un nuevo comando usando alias. Lo primero que tenemos que hacer es idear un nombre para nuestro nuevo comando. Intentemos probar. Antes de hacer eso, sería una buena idea averiguar si la prueba del nombre ya se está utilizando. Para averiguarlo, podemos volver a usar el comando *type*:

---

```
[me@linuxbox ~]$ prueba de tipo
test es un shell incorporado
```

---

¡Vaya! La prueba del nombre ya está tomada. Probemos

---

```
foo: [me@linuxbox ~]$ tipo foo
bash: tipo: foo: no encontrado
```

---

¡Bien! *foo* no se toma. Así que vamos a crear nuestro alias:

---

```
[me@linuxbox ~]$ alias foo='cd /usr; ls; CD -'
```

---

Observe la estructura de este comando:

*alias name='cadena'*

Después del comando *alias*, le damos al alias un nombre seguido inmediatamente (no se permiten espacios en blanco) por un signo igual, que es seguido inmediatamente por una cadena entrecomillada que contiene el significado que se asignará al nombre. Después de definir nuestro alias, se puede usar en cualquier lugar donde el shell esperaría un comando.

Vamos a probarlo:

---

```
[me@linuxbox ~]$ foo
bote juegos Kerberos lib64 Compartir
localmente Tmp etc incluyen lib libexec sbin
Fuente
/inicio/yo
[me@linuxbox ~]$
```

---

También podemos volver a usar el comando type para ver nuestro alias:

---

```
[me@linuxbox ~]$ tipo foo
foo tiene el alias 'cd /usr; ls ; CD -'
```

---

Para eliminar un alias, se utiliza el comando unalias, de la siguiente manera:

---

```
[me@linuxbox ~]$ unalias foo
[me@linuxbox ~]$ type foo
bash: type: foo: not found
```

---

Si bien evitamos deliberadamente nombrar nuestro alias con un nombre común existente, a veces es deseable hacerlo. Esto se hace a menudo para aplicar una opción comúnmente deseada a cada invocación de un comando común. Por ejemplo, vimos anteriormente cómo el comando ls a menudo tiene un alias para agregar soporte de color:

---

```
[me@linuxbox ~]$ tipo ls
ls tiene el alias 'ls --color=tty'
```

---

Para ver todos los alias definidos en el entorno, utilice el comando alias sin argumentos. Estos son algunos de los alias definidos de forma predeterminada en un sistema Fedora. Trata de averiguar qué hacen todos:

---

```
[me@linuxbox ~]$ alias
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
```

---

Hay un pequeño problema con la definición de alias en la línea de comandos. Desaparecen cuando finaliza la sesión de shell. En un capítulo posterior veremos cómo añadir nuestros propios alias a los archivos que establecen el entorno cada vez que nos conectamos, pero por ahora, ¡disfrutad del hecho de que hemos dado nuestro primer, aunque pequeño, paso en el mundo de la programación de shell!

## Volver a visitar a viejos amigos

Ahora que hemos aprendido a encontrar la documentación de los comandos, vaya y busque la documentación de todos los comandos que hemos encontrado hasta ahora. ¡Estudie qué opciones adicionales están disponibles y pruébelas!



# 6

## REDIRECCIONAMIENTO

En esta lección vamos a dar rienda suelta a lo que puede ser la característica más genial de la línea de comandos: la *redirección de E/S*. La *E/S* significa *entrada/salida*, y con esta función puede redirigir la entrada y la salida de

comandos hacia y desde archivos, así como conectar varios comandos para crear potentes *canalizaciones de comandos*. Para mostrar esta facilidad, introduciremos los siguientes comandos:

- **cat**: concatenar archivos.
- **sort**: ordena las líneas de texto.
- **uniq**: informe u omita líneas repetidas.
- **wc**: imprime recuentos de saltos de línea, palabras y bytes para cada archivo.
- **grep**: imprime líneas que coinciden con un patrón.
- **head**: genera la primera parte de un archivo.
- **tail**: genera la última parte de un archivo.
- **tee**: lea desde la entrada estándar y escriba en la salida y los archivos estándar.

## Entrada, salida y error estándar

Muchos de los programas que hemos utilizado hasta ahora producen algún tipo de salida. Esta salida suele constar de dos tipos. En primer lugar, tenemos los resultados del programa; es decir, los datos para los que está diseñado el programa. En segundo lugar, tenemos mensajes de estado y error que nos indican cómo va el programa. Si nos fijamos en un comando como `ls`, podemos ver que muestra sus resultados y sus mensajes de error en la pantalla.

Siguiendo con el tema de Unix de "todo es un archivo", programas como `ls` en realidad envían sus resultados a un archivo especial llamado *salida estándar* (a menudo expresado como *stdout*) y sus mensajes de estado a otro archivo llamado *estándar*

*Error (stderr)*. De forma predeterminada, tanto la salida estándar como el error estándar están vinculados a la pantalla y no se guardan en un archivo de disco.

Además, muchos programas toman la entrada de una función llamada *entrada estándar (stdin)*, que está, de forma predeterminada, conectada al teclado.

La redirección de E/S nos permite cambiar dónde va la salida y de dónde viene la entrada. Normalmente, la salida va a la pantalla y la entrada proviene del teclado, pero con la redirección de E/S podemos cambiar eso.

### Redireccionamiento de la salida estándar

La redirección de E/S nos permite redefinir dónde va la salida estándar. Para redirigir la salida estándar a otro archivo en lugar de a la pantalla, usamos el operador de redirección `>` seguido del nombre del archivo.

¿Por qué querríamos hacer esto? A menudo es útil almacenar la salida de un comando en un archivo. Por ejemplo, podríamos decirle al shell que envíe la salida del comando `ls` al archivo `ls-output.txt` en lugar de a la pantalla:

---

```
[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt
```

---

Aquí, creamos una lista larga del directorio `/usr/bin` y enviamos los resultados al archivo `ls-output.txt`. Examinemos la salida redirigida del comando:

---

```
[me@linuxbox ~]$ ls -l ls-output.txt
-rw-rw-r-- 1 me   me 167878 01/02/2012 15:07 ls-output.txt
```

---

Bueno: un archivo de texto grande y bonito. Si nos fijamos en el fichero con menos, veremos que el fichero `ls-output.txt` sí contiene los resultados de nuestro comando `ls`:

---

```
[me@linuxbox ~]$ menos ls-output.txt
```

---

Ahora, repitamos nuestra prueba de redirección, pero esta vez con

un giro. Cambiaremos el nombre del directorio por uno que no exista:

---

```
[me@linuxbox ~]$ ls -l /bin/usr > ls-output.txt  
ls: no se puede acceder a /bin/usr: No existe tal archivo o directorio
```

Hemos recibido un mensaje de error. Esto tiene sentido porque especificamos el directorio inexistente `/bin/usr`, pero ¿por qué se mostró el mensaje de error en la pantalla en lugar de ser redirigido al archivo `ls-output.txt`? La respuesta es que el programa `ls` no envía sus mensajes de error a la salida estándar. En cambio, como la mayoría de los programas Unix bien escritos, envía sus mensajes de error al error estándar. Dado que redirigimos solo la salida estándar y no el error estándar, el mensaje de error aún se envió a la pantalla. Veremos cómo redirigir el error estándar en solo un minuto, pero primero, veamos qué sucedió con nuestro archivo de salida:

---

```
[me@linuxbox ~]$ ls -l ls-output.txt  
-rw-rw-r-- 1 me me 0 01/02/2012 15:08 ls-output.txt
```

---

¡El archivo ahora tiene longitud cero! Esto se debe a que, cuando redirigimos la salida con el operador de redirección `>`, el archivo de destino siempre se reescribe desde el principio. Dado que nuestro comando `ls` no generó ningún resultado y solo un mensaje de error, la operación de redirección comenzó a reescribir el archivo y luego se detuvo debido al error, lo que resultó en su truncamiento. De hecho, si alguna vez necesitamos truncar un archivo (o crear un nuevo archivo vacío) podemos usar un truco como este:

---

```
[me@linuxbox ~]$ > ls-output.txt
```

---

Simplemente usando el operador de redirección sin ningún comando que lo preceda, se truncará un archivo existente o se creará un nuevo archivo vacío.

Entonces, ¿cómo podemos agregar la salida redirigida a un archivo en lugar de sobrescribir el archivo desde el principio? Para eso, usamos el operador de redirección `>>`, así:

---

```
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt
```

---

El uso del operador `>>` dará como resultado que la salida se anexe al archivo. Si el archivo aún no existe, se crea como si se hubiera utilizado el operador `>`. Pongámoslo a prueba:

---

```
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt  
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt  
[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt  
[me@linuxbox ~]$ ls -l ls-output.txt  
-rw-rw-r-- 1 me me 503634 01/02/2012 15:45 ls-output.txt
```

---

Repetimos el comando tres veces, lo que resultó en un archivo de salida tres veces más grande.

### **Redireccionamiento de error estándar**

El error estándar de redireccionamiento carece de la facilidad de usar un operador de redireccionamiento dedicado. Para redirigir el error estándar debemos referirnos a su *descriptor de archivo*. Un programa puede producir

resultados en cualquiera de varios flujos de archivos numerados. Mientras

Nos hemos referido a los tres primeros de estos flujos de archivos como entrada, salida y error estándar, el shell los hace referencia internamente como descriptores de archivo 0, 1 y 2, respectivamente. El shell proporciona una notación para redirigir archivos utilizando el número de descriptor de archivo. Dado que el error estándar es el mismo que el descriptor de archivo 2, podemos redirigir el error estándar con esta notación:

---

```
[me@linuxbox ~]$ ls -l /bin/usr 2> ls-error.txt
```

---

El descriptor de archivo 2 se coloca inmediatamente antes del operador de redirección para realizar la redirección del error estándar al *ls-error.txt* de archivo.

### ***Redireccionamiento de la salida estándar y el error estándar a un archivo***

Hay casos en los que es posible que deseemos capturar toda la salida de un comando en un solo archivo. Para hacer esto, debemos redirigir tanto la salida estándar como el error estándar al mismo tiempo. Hay dos formas de hacerlo. Primero, aquí está la forma tradicional, que funciona con versiones antiguas del shell:

---

```
[me@linuxbox ~]$ ls -l /bin/usr > ls-output.txt 2>&1
```

---

Con este método, realizamos dos redirecciones. Primero redirigimos la salida estándar al *ls-output.txt* de archivos, y luego redirigimos el descriptor de archivo 2 (error estándar) al descriptor de archivo 1 (salida estándar) usando la notación *2>&1*.

**Nota:** *Tenga en cuenta que el orden de las redirecciones es significativo. La redirección del error estándar siempre debe producirse después de redirigir la salida estándar o no funciona. En el ejemplo anterior, > ls-output.txt 2>&1 redirige el error estándar al ls-output.txt de archivos, pero si el orden se cambia a 2>&1 > ls-output.txt, el error estándar se dirige a la pantalla.*

Las versiones recientes de bash proporcionan un segundo método más simplificado para realizar esta redirección combinada:

---

```
[me@linuxbox ~]$ ls -l /bin/usr &> ls-output.txt
```

---

En este ejemplo, usamos la notación única *&>* para redirigir tanto la salida estándar como el error estándar al archivo *ls-output.txt*.

### ***Eliminación de salidas no deseadas***

A veces el silencio es realmente oro, y no queremos que salga de una comunión, simplemente queremos tirarla a la basura. Esto se aplica especialmente a los mensajes de error y de estado. El sistema proporciona una forma de hacer esto redirigiendo la salida a un archivo especial llamado

`/dev/null`. Este archivo es un dispositivo del sistema llamado cubo de *bits*, que acepta entradas y no hace nada con ellas. Para suprimir los mensajes de error de un comando, hacemos lo siguiente:

---

```
[me@linuxbox ~]$ ls -l /bin/usr 2> /dev/null
```

## /DE V/ NULL EN LA CULTURA UNIX

El cubo de bits es un concepto antiguo de Unix, y debido a su universalidad ha aparecido en muchas partes de la cultura Unix. Así que cuando alguien dice que está enviando tus comentarios a "dev null", ahora sabes lo que significa. Para más ejemplos, véase el artículo de Wikipedia en <http://en.wikipedia.org/wiki/Dev/null>.

### *Redireccionamiento de entrada estándar*

Hasta ahora, no hemos encontrado ningún comando que haga uso de la entrada estándar (de hecho, lo hemos hecho, pero revelaremos esa sorpresa un poco más adelante), por lo que debemos introducir uno.

#### **cat: concatenar archivos**

El comando cat lee uno o más archivos y los copia a la salida estándar de la siguiente manera:

```
cat [archivo...]
```

En la mayoría de los casos, se puede pensar en cat como un análogo al comando TYPE en DOS. Puede usarlo para mostrar archivos sin paginación. Por ejemplo

```
[me@linuxbox ~]$ gato ls-output.txt
```

mostrará el contenido del archivo *ls-output.txt*. cat se utiliza a menudo para mostrar archivos de texto cortos. Dado que cat puede aceptar más de un archivo como argumento, también se puede usar para unir archivos.

Supongamos que hemos descargado un archivo grande que se ha dividido en varias partes (los archivos multimedia a menudo se dividen de esta manera en Usenet) y queremos volver a unirlos. Si los archivos se nombraron

```
Película.Mpeg.001 Película.Mpeg.002 .... Movie.mpeg.099
```

Podríamos volver a unirlos con este comando:

```
[me@linuxbox ~]$ gato movie.mpeg.0* > movie.mpeg
```

Dado que los comodines siempre se expanden en orden ordenado, los argumentos se organizarán en el orden correcto.

Todo esto está muy bien, pero ¿qué tiene que ver esto con la entrada estándar? Todavía nada, pero intentemos algo más. ¿Qué pasa si entramos cat sin argumentos?

```
[me@linuxbox ~]$ gato
```

No pasa nada, simplemente se queda ahí como si estuviera colgado. Puede parecer así, pero en realidad está haciendo exactamente lo que se supone que debe hacer.

Si a cat no se le da ningún argumento, lee desde la entrada estándar, y dado que la entrada estándar está, de forma predeterminada, adjunta al teclado, ¡está esperando que escribamos algo!



Pruebe esto:

---

```
[me@linuxbox ~]$ gato  
El rápido zorro marrón saltó sobre el perro perezoso.
```

---

A continuación, escriba CTRL-D (es decir, mantenga presionada la tecla CTRL y presione D) para decirle al gato que ha alcanzado el *fin de archivo (EOF)* en la entrada estándar:

---

```
[me@linuxbox ~]$ gato  
El rápido zorro marrón saltó sobre el perro perezoso.  
El rápido zorro marrón saltó sobre el perro perezoso.
```

---

En ausencia de argumentos de nombre de archivo, cat copia la entrada estándar a la salida estándar, por lo que vemos nuestra línea de texto repetida. Podemos usar este comportamiento para crear archivos de texto cortos. Digamos que queremos crear un archivo llamado *lazy\_dog.txt* que contiene el texto de nuestro ejemplo. Nosotros haríamos lo siguiente:

---

```
[me@linuxbox ~]$ gato > lazy_dog.txt  
El rápido zorro marrón saltó sobre el perro perezoso.
```

---

Introducimos el comando seguido del texto que queremos colocar en el archivo. Recuerde escribir CTRL-D al final. ¡Usando la línea de comandos, hemos implementado el procesador de textos más tonto del mundo! Para ver nuestros resultados, podemos usar cat para copiar el archivo a la salida estándar nuevamente:

---

```
[me@linuxbox ~]$ gato lazy_dog.txt  
El rápido zorro marrón saltó sobre el perro perezoso.
```

---

Ahora que sabemos cómo cat acepta la entrada estándar además de los argumentos de nombre de archivo, intentemos redirigir la entrada estándar:

---

```
[me@linuxbox ~]$ gato < lazy_dog.txt  
El rápido zorro marrón saltó sobre el perro perezoso.
```

---

Usando el operador de redirección <, cambiamos la fuente de entrada estándar del teclado al archivo *lazy\_dog.txt*. Vemos que el resultado es el mismo que pasar un solo argumento de nombre de archivo. Esto no es particularmente útil en comparación con pasar un argumento de nombre de archivo, pero sirve para demostrar el uso de un archivo como fuente de entrada estándar. Otros comandos hacen un mejor uso de la entrada estándar, como pronto veremos.

Antes de continuar, echa un vistazo a la página de manual de gato, ya que tiene varias opciones interesantes.

## Tuberías

La capacidad de los comandos para leer datos de la entrada estándar y enviarlos a la salida estándar es utilizada por una característica de shell

llamada *canalizaciones*. Uso del operador de tubería | (barra vertical), la salida estándar de un comando se puede *canalizar* a la entrada estándar de otro.

*comando1 | Comando2*

Para demostrar completamente esto, vamos a necesitar algunos comandos. ¿Recuerdas que dijimos que había uno que ya conocíamos que aceptaba entradas estándar? Es menos. Podemos usar less para mostrar, página por página, la salida de cualquier comando que envíe sus resultados a la salida estándar:

---

```
[me@linuxbox ~]$ ls -l /usr/bin | menos
```

---

¡Esto es extremadamente útil! Usando esta técnica, podemos examinar convenientemente la salida de cualquier comando que produzca una salida estándar.

## Filtros

Las canalizaciones se utilizan a menudo para realizar operaciones complejas con los datos. Es posible poner varios comandos juntos en una tubería. Con frecuencia, los comandos utilizados de esta manera se denominan *filtros*. Los filtros toman la entrada, la cambian de alguna manera y, a continuación, la emiten. El primero que intentaremos es ordenar. Imaginemos que queremos hacer una lista combinada de todos los programas ejecutables en */bin* y */usr/bin*, colóquelos en orden ordenado y luego vea la lista:

---

```
[me@linuxbox ~]$ ls /bin /usr/bin | ordenar | menos
```

---

Dado que especificamos dos directorios (*/bin* y */usr/bin*), la salida de ls habría consistido en dos listas ordenadas, una para cada directorio. Al incluir la ordenación en nuestra canalización, cambiamos los datos para producir una única lista ordenada.

### *uniq: informar u omitir líneas repetidas*

El comando uniq se utiliza a menudo junto con sort. UNIQ acepta una lista ordenada de datos de una entrada estándar o de un único argumento de nombre de archivo (consulte la página del manual de UNIQ para obtener más detalles) y, de forma predeterminada, elimina cualquier duplicado de la lista. Por lo tanto, para asegurarnos de que nuestra lista no tenga duplicados (es decir, cualquier programa con el mismo nombre que aparezca en los *directorios /bin* y */usr/bin*) agregaremos uniq a nuestra canalización:

---

```
[me@linuxbox ~]$ ls /bin /usr/bin | ordenar | uniq | menos
```

---

En este ejemplo, usamos uniq para eliminar cualquier duplicado de la salida del comando de ordenación. Si queremos ver la lista de duplicados en su lugar, agregamos la opción -d a uniq de la siguiente manera:

---

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq -d | less
```

---

### *wc: recuentos de líneas, palabras y bytes de impresión*

El comando wc (recuento de palabras) se utiliza para mostrar el número de

líneas, palabras y bytes contenidos en los archivos. Por ejemplo:

---

```
[me@linuxbox ~]$ WC ls-output.txt  
7902 64566 503634 ls-output.txt
```

---

En este caso, imprime tres números: líneas, palabras y bytes contenidos en *ls-output.txt*. Al igual que nuestros comandos anteriores, si se ejecuta sin argumentos de línea de comandos, wc acepta entradas estándar. La opción **-l** limita su salida solo a las líneas de informe. Agregarlo a una canalización es una forma práctica de contar cosas. Para ver el número de elementos que tenemos en nuestra lista ordenada, podemos hacer lo siguiente:

---

```
[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | wc -l  
2728
```

---

### **grep: líneas de impresión que coinciden con un patrón**

grep es un poderoso programa que se utiliza para encontrar patrones de texto dentro de los archivos, como este:

```
Patrón grep [archivo...]
```

Cuando grep encuentra un "patrón" en el archivo, imprime las líneas que lo contienen. Los patrones que puede coincidir grep pueden ser muy complejos, pero por ahora nos concentraremos en coincidencias de texto simples. Cubriremos los patrones avanzados, llamados *expresiones regulares*, en el Capítulo 19.

Supongamos que queremos encontrar todos los archivos de nuestra lista de programas que tienen la palabra *zip* en el nombre. Esta búsqueda podría darnos una idea de qué programas de nuestro sistema tienen algo que ver con la compresión de archivos. Nosotros haríamos lo siguiente:

---

```
[me@linuxbox ~]$ ls /bin /usr/bin | ordenar | uniq | grep zip  
bunzip2  
bzip2  
gunzip  
gzip  
unzip  
zip  
zipcloak  
zipgrep  
zipinfo  
zipnote  
zipsplit
```

---

Hay un par de opciones útiles para grep: **-i**, que hace que grep ignore las mayúsculas y minúsculas al realizar la búsqueda (normalmente las búsquedas distinguen entre mayúsculas y minúsculas) y **-v**, que le dice a grep que imprima solo las líneas que no coincidan con el patrón.

### **head/tail: imprime la primera/última parte de los archivos**

A veces no desea toda la salida de un comando. Es posible que desee solo las primeras líneas o las últimas líneas. El comando head imprime las primeras 10 líneas de un archivo y el comando tail imprime las últimas 10 líneas. De forma predeterminada, ambos comandos imprimen 10 líneas de texto, pero esto se puede ajustar con la opción **-n**:

---

```
[me@linuxbox ~]$ cabeza -n 5 ls-output.txt
```

---

Total 343496

-rwxr-xr-x 1 raíz raíz

31316 2011-12-05 08:58 [

```
-rwxr-xr-x 1 raíz raíz          8240 2011-12-09 13:39 411toppm  
-rwxr-xr-x 1 raíz raíz          111276 26/11/2011 14:27 a2p  
-rwxr-xr-x 1 raíz raíz          25368 06/10/2010 20:16 a52dic  
[me@linuxbox ~]$ cola -n 5 ls-output.txt  
-rwxr-xr-x 1 raíz raíz          5234 27/06/2011 10:56 znuevo  
-rwxr-xr-x 1 raíz raíz          691 10/09/2009 04:21 zonetab2pot.py  
-rw-r--r-- 1 raíz raíz          930 2011-11-01 12:23 zonetab2pot.pyc  
-rw-r--r-- 1 raíz raíz          930 2011-11-01 12:23 zonetab2pot.pyo  
lrwxrwxrwx 1 raíz raíz          6 2012-01-31 05:22 zsoelim -> soelim
```

---

Estos también se pueden usar en tuberías:

```
[me@linuxbox ~]$ ls /usr/bin | cola -n 5  
znew  
zonetab2pot.py  
zonetab2pot.pyc  
zonetab2pot.pyo  
zsoelim
```

---

Tail tiene una opción que te permite ver archivos en tiempo real. Esto es útil para observar el progreso de los archivos de registro a medida que se escriben. En el siguiente ejemplo, veremos el archivo *de mensajes* en */var/log*. Los privilegios de superusuario son necesarios para hacer esto en algunas distribuciones de Linux, ya que el *El archivo /var/log/messages* puede contener información de seguridad.

```
[me@linuxbox ~]$ tail -f /var/log/messages  
Feb 8 13:40:05 twin4 dhclient: DHCPACK de 192.168.1.1  
Feb 8 13:40:05 twin4 dhclient: vinculado a 192.168.1.4 -- renovación en 1652 segundos.  
8 de febrero 13:55:32 twin4 mountd[3953]: /var/NFSv4/musicbox exportado a 192.168.1.0/24 y twin7.localdomain en 192.168.1.0/24,twin7.localdomain 8 de febrero 14:07:37 twin4 dhclient: DHCPREQUEST en eth0 al puerto 67 de 192.168.1.1  
Feb 8 14:07:37 twin4 dhclient: DHCPACK de 192.168.1.1  
Feb 8 14:07:37 twin4 dhclient: vinculado a 192.168.1.4 -- renovación en 1771 segundos.  
Feb 8 14:09:56 twin4 smartd[3468]: Dispositivo: /dev/hda, SMART Prefailure Atributo: 8 Seek_Time_Performance cambiado de 237 a 236  
8 de febrero 14:10:37 twin4 mountd[3953]: /var/NFSv4/musicbox exportado a 192.168.1.0/24 y twin7.localdomain en 192.168.1.0/24,twin7.localdomain 8 de febrero 14:25:07 twin4 sshd(pam_unix)[29234]: sesión abierta para el usuario me por (uid=0)  
Feb 8 14:25:36 twin4 su(pam_unix)[29279]: sesión abierta para el usuario root por me(uid=500)
```

---

Usando la opción **-f**, tail continúa monitoreando el archivo y cuando se agregan nuevas líneas, aparecen inmediatamente en la pantalla. Esto continúa hasta que escriba **CTRL-C**.

### **tee: lectura de Stdin y salida a stdout y archivos**

De acuerdo con nuestra analogía de plomería, Linux proporciona un comando llamado **tee** que crea un accesorio en "T" en nuestra tubería. El programa **tee** lee la entrada estándar y la copia tanto en la salida estándar (lo que permite que los datos continúen por la tubería) como en uno o más archivos. Esto es útil para capturar el contenido de una canalización en una

etapa intermedia del procesamiento. Aquí repetimos

Uno de nuestros ejemplos anteriores, esta vez incluyendo TEE para capturar toda la lista de directorios en el archivo *ls.txt* antes de que grep filtre el contenido de la canalización:

```
[me@linuxbox ~]$ ls /usr/bin | Tee ls.txt | grep zip
bunzip2
bzip2
gunzip
gzip
unzip
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
```

## Nota final

Como siempre, consulte la documentación de cada uno de los comandos que hemos cubierto en este capítulo. Hemos visto solo su uso más básico, y todos tienen una serie de opciones interesantes. A medida que vayamos adquiriendo experiencia en Linux, veremos que la función de redirección de la línea de comandos es extremadamente útil para resolver problemas especializados. Muchos comandos hacen uso de entrada y salida estándar, y casi todos los programas de línea de comandos utilizan el error estándar para mostrar sus mensajes informativos.

### LINUX TIENE QUE VER CON LA IMAGINACIÓN

Cuando me piden que explique la diferencia entre Windows y Linux, a menudo uso una analogía con un juguete.

Windows es como una Game Boy. Vas a la tienda y compras uno nuevo y brillante en la caja. Lo llevas a casa, lo enciendes y juegas con él. Bonitos gráficos, lindos sonidos. Sin embargo, después de un tiempo, te cansas del juego que venía con él, así que vuelves a la tienda y compras otro. Este ciclo se repite una y otra vez. Finalmente, regresas a la tienda y le dices a la persona detrás del mostrador: "¡Quiero un juego que haga esto!", solo para que te digan que no existe tal juego porque no hay "demanda de mercado" para él. Entonces dices: "¡Pero solo necesito cambiar esta cosa!" La persona detrás del mostrador dice que no puedes cambiarlo. Todos los juegos están sellados en sus cartuchos. Descubres que tu juguete se limita a los juegos que otros han decidido que necesitas y nada más.

Linux, por otro lado, es como el conjunto de constructores más grande del mundo. Lo abres y es solo una gran colección de piezas: muchos puntales de acero, tornillos, tuercas, engranajes, poleas y motores, y algunas sugerencias sobre qué construir. Así que comienzas a jugar con él. Construyes una de las sugerencias y luego otra. Después de un tiempo, descubres que tienes tus propias ideas de qué hacer. No tienes que volver nunca a la tienda, porque ya tienes todo lo que necesitas. El Erector Set toma la forma de su imaginación. Hace lo que quieras.

Su elección de juguetes es, por supuesto, algo personal, entonces, ¿qué juguete le resultaría más satisfactorio?

# 7

## VER EL MUNDO COMO LO VE EL CAPARAZÓN

En este capítulo vamos a ver algo de la "magia" que se produce en la línea de comandos cuando se pulsa la tecla ENTER . Si bien examinaremos varias características interesantes y complejas del shell, lo haremos con un solo comando nuevo:

- echo: muestra una línea de texto.

### Expansión

Cada vez que escribe una línea de comando y presiona la tecla ENTER , bash realiza varios procesos sobre el texto antes de llevar a cabo su comando. Hemos visto un par de casos de cómo una simple secuencia de caracteres, por ejemplo \*, puede tener mucho significado para el shell. El proceso que hace que esto suceda se llama *expansión*. Con la expansión, entras en algo, y se expande en otra cosa antes de que el caparazón actúe sobre ello. Para demostrar lo que

Con esto, echemos un vistazo al comando echo. echo es un shell que realiza una tarea muy simple: imprime sus argumentos de texto en una salida estándar.

---

```
[me@linuxbox ~]$ echo esto es una prueba  
Esto es una prueba
```

---

Eso es bastante sencillo. Cualquier argumento que se pase a echo se muestra. Veamos otro ejemplo:

---

```
[me@linuxbox ~]$ echo *  
Documentos de escritorio ls-output.txt música imágenes Plantillas públicas Videos
```

---

Entonces, ¿qué acaba de pasar? ¿Por qué no se imprimió el eco \*? Como recordará de nuestro trabajo con comodines, el carácter \* significa "coincidir con cualquier carácter en un nombre de archivo", pero lo que no vimos en nuestra discusión original fue cómo el shell hace eso. La respuesta simple es que el shell expande el \* a otra cosa (en este caso, los nombres de los archivos en la directriz de trabajo actual) antes de que se ejecute el comando echo. Cuando se presiona la tecla ENTER, el shell expande automáticamente cualquier carácter calificador en la línea de comandos antes de que se lleve a cabo el comando, por lo que el comando echo nunca vio el \*, solo su resultado expandido. Sabiendo esto, podemos ver que echo se comportó como se esperaba.

### ***Expansión de nombre de ruta***

El mecanismo por el cual funcionan los comodines se denomina *expansión de nombre de ruta*. Si

Probamos algunas de las técnicas que empleamos en nuestros capítulos anteriores, veremos que realmente son expansiones. Dado un directorio de inicio que se ve así:

---

```
[me@linuxbox ~]$ ls  
Escritorio  ls-output.txt  imágenes  Plantillas  
Documentos  Música       Público    Videos
```

---

Podríamos llevar a cabo las siguientes ampliaciones:

---

```
[me@linuxbox ~]$ echo D*  
Documentos de escritorio
```

---

y

---

```
[me@linuxbox ~]$ echo *s  
Documentos  imágenes  Plantillas  Vídeos
```

---

o incluso

---

```
[me@linuxbox ~]$ echo [[::superior:]]*  
Documentos de escritorio  Música  imágenes  Plantillas públicas  Videos
```

---

Y mirando más allá de nuestro directorio de inicio:

```
[me@linuxbox ~]$ echo /usr/*/share  
/usr/kerberos/share /usr/local/share
```

## EXPANSIÓN DEL NOMBRE DE RUTA DE HID DE N FIES

Como sabemos, los nombres de archivo que comienzan con un carácter de punto están ocultos. La expansión del nombre de ruta también respeta este comportamiento. Una expansión como

```
echo *
```

No revela archivos ocultos.

Podría parecer a primera vista que podríamos incluir archivos ocultos en una expansión iniciando el patrón con un punto inicial, así:

```
echo .*
```

Casi funciona. Sin embargo, si examinamos los resultados de cerca, veremos que los nombres . y .. también aparecerán en los resultados. Dado que estos nombres hacen referencia al directorio de trabajo actual y a su directorio principal, es probable que el uso de este patrón produzca un resultado incorrecto. Podemos ver esto si probamos el comando

```
ls -d .* | menos
```

Para realizar correctamente la expansión del nombre de ruta en esta situación, tenemos que emplear un patrón más específico. Esto funcionará correctamente:

```
ls -d .[!.]?*
```

Este patrón se expande a todos los nombres de archivo que comienzan con un punto, no incluyen un segundo punto, contienen al menos un carácter adicional y pueden ir seguidos de cualquier otro carácter.

### Expansión de tilde

Como recordará de nuestra introducción al comando cd, el carácter de tilde (~) tiene un significado especial. Cuando se usa al principio de una palabra, se expande al nombre del directorio principal del usuario designado o, si no se nombra a ningún usuario, al directorio principal del usuario actual:

```
[me@linuxbox ~]$ echo ~  
/inicio/yo
```

Si el usuario *foo* tiene una cuenta, entonces

---

```
[me@linuxbox ~]$ echo ~foo  
/inicio/foo
```

---

## Expansión aritmética

El shell permite realizar operaciones aritméticas por expansión. Esto nos permite usar el símbolo del shell como calculadora:

---

```
[me@linuxbox ~]$ echo $((2 + 2))  
4
```

---

La expansión aritmética utiliza la siguiente forma:

$\$((expresión))$

donde *expresión* es una expresión aritmética que consta de valores y operadores aritméticos.

La expansión aritmética sólo admite números enteros (números enteros, sin decimales), pero puede realizar un buen número de operaciones diferentes. En la tabla 7-1 se enumeran algunos de los operadores admitidos.

**Tabla 7-1: Operadores aritméticos**

Operador	Descripción
+	Adición
-	Sustracción
*	Multiplicación
/	División (Pero recuerde, debido a que la expansión solo admite aritmética de enteros, los resultados son números enteros).
%	Modulo, que simplemente significa <i>resto</i>
**	Exponenciación

Los espacios no son significativos en las expresiones aritméticas y las expresiones pueden estar anidadas. Por ejemplo, multiplica 52 por 3:

---

```
[me@linuxbox ~]$ echo $(((5**2) * 3))  
75
```

---

Los paréntesis simples se pueden usar para agrupar varias subexpresiones. Con esta técnica, podemos reescribir el ejemplo anterior y obtener el mismo resultado usando una sola expansión en lugar de dos:

---

```
[me@linuxbox ~]$ echo $((5**2) * 3))  
75
```

---

A continuación, se muestra un ejemplo en el que se utilizan los operadores de división y resto. Observe el efecto de la división de números enteros:

---

```
[me@linuxbox ~]$ echo Cinco dividido por dos es igual a $((5/2))  
Cinco dividido por dos es igual a 2
```

---

```
[me@linuxbox ~]$ echo con ${((%2))} sobrante.  
con 1 sobrante.
```

---

La expansión aritmética se trata con mayor detalle en el capítulo 34.

### **Expansión de la abrazadera**

Quizás la expansión más extraña se llama *expansión de abrazaderas*. Con él, puede crear varias cadenas de texto a partir de un patrón que contenga llaves. He aquí un ejemplo:

```
[me@linuxbox ~]$ echo Delantero-[A,B,C]-Trasero  
Delantero-A-Trasero Delantero-B-Trasero Delantero-C-Trasero
```

---

Los patrones que se van a expandir entre llaves pueden contener una parte inicial denominada *preámbulo* y una parte final denominada *posdata*. La propia expresión entre llaves puede contener una lista de cadenas separadas por comas o un intervalo de enteros o caracteres individuales. Es posible que el patrón no contenga espacios en blanco incrustados. A continuación, se muestra un ejemplo utilizando un rango de números enteros:

```
[me@linuxbox ~]$ echo Number_{1..5}  
Number_1 Number_2 Number_3 Number_4 Number_5
```

---

Aquí obtenemos un rango de letras en orden inverso:

```
[me@linuxbox ~]$ echo {Z.. A}  
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

---

Las expansiones de llaves pueden estar anidadas:

```
[me@linuxbox ~]$ echo a{A{1,2},B{3,4}}b  
aA1b aA2b aB3b aB4b
```

---

Entonces, ¿para qué sirve esto? La aplicación más común es hacer listas de archivos o directorios a crear. Por ejemplo, si fuéramos fotógrafos y tuviéramos una gran colección de imágenes que quisiéramos organizar por años y meses, lo primero que podríamos hacer es crear una serie de directorios nombrados en formato numérico año-mes. De esta manera, los nombres de los directorios se ordenarán en orden cronológico . Podríamos escribir una lista completa de directorios, pero eso es mucho trabajo y también es propenso a errores. En su lugar, podríamos hacer lo siguiente:

```
[me@linuxbox ~]$ mkdir Fotos  
[me@linuxbox ~]$ cd Fotos  
[me@linuxbox Pics]$ mkdir {2009..2011}-{0{1..9}} {2009..2011}-{10..12}  
[me@linuxbox Fotos]$ ls  
2009-01 2009-07 2010-01 2010-07 2011-01 2011-07  
2009-02 2009-08 2010-02 2010-08 2011-02 2011-08  
2009-03 2009-09 2010-03 2010-09 2011-03 2011-09  
2009-04 2009-10 2010-04 2010-10 2011-04 2011-10  
2009-05 2009-11 2010-05 2010-11 2011-05 2011-11  
2009-06 2009-12 2010-06 2010-12 2011-06 2011-12
```

---

¡Bastante hábil!

## **Expansión de parámetros**

Solo vamos a tocar brevemente la expansión de parámetros en este capítulo, pero la cubriremos ampliamente más adelante. Es una característica que es más útil en scripts de shell que directamente en la línea de comandos. Muchas de sus capacidades tienen que ver con la capacidad del sistema para almacenar pequeños fragmentos de datos y dar un nombre a cada fragmento. Muchos de estos fragmentos, más propiamente llamados *variables*, están disponibles para su examen. Por ejemplo, la variable denominada USER contiene su nombre de usuario. Para

---

invocar la expansión de parámetros y revelar el contenido de USER, debe hacer lo siguiente:

```
[me@linuxbox ~]$ echo $USER  
me
```

---

Para ver una lista de las variables disponibles, pruebe lo siguiente:

```
[me@linuxbox ~]$ printenv | menos
```

Es posible que haya notado que con otros tipos de expansión, si escribe mal un patrón, la expansión no tendrá lugar y el comando echo simplemente mostrará el patrón mal escrito. Con la expansión de parámetros, si escribe mal el nombre de una variable, la expansión aún se llevará a cabo, pero dará como resultado una cadena vacía:

---

```
[me@linuxbox ~]$ echo $SUER  
[me@linuxbox ~]$
```

## **Sustitución de comandos**

La sustitución de comandos nos permite usar la salida de un comando como una expansión:

---

```
[me@linuxbox ~]$ echo $(ls)  
Documentos de escritorio ls-output.txt música imágenes Plantillas públicas Videos
```

Uno de mis favoritos dice algo así:

---

```
[me@linuxbox ~]$ ls -l $(que cp)  
-rwxr-xr-x 1 root root 71516 2012-12-05 08:58 /bin/cp
```

Aquí pasamos los resultados de los cuales cp como argumento al comando ls, obteniendo así la lista del programa cp sin tener que conocer su nombre de ruta completo. No nos limitamos a simples comandos. Se pueden usar tuberías completas (solo se muestra una salida parcial):

---

```
[me@linuxbox ~]$ fichero $(ls /usr/bin/* | grep zip)  
/usr/bin/bunzip2:    Enlace simbólico a 'Bzip2'  
/effect/bin/buzzp2: Ejecutable ELF LSB de 32 bits, Intel 80386, versión 1 (SYSV  
, enlazado dinámicamente (usa bibliotecas compartidas), para GNU/Linux 2.6.9,  
despojado  
/usr/bin/bzip2recover: Ejecutable ELF LSB de 32 bits, Intel 80386, versión 1  
(SYSV), enlazado dinámicamente (usa bibliotecas compartidas), para  
GNU/Linux 2.6.9, despojado  
/usr/bin/funzip:    Ejecutable ELF LSB de 32 bits, Intel 80386, versión 1 (SYSV
```



), enlazado dinámicamente (usa bibliotecas compartidas), para GNU/Linux 2.6.9, despojado

/usr/bin/gpg-cremaller:	Ejecutable de texto de script de shell Bourne
/usr/bin/gunzip:	enlace simbólico a '... /.. /papelera/gunzip'
/usr/bin/gzip:	enlace simbólico a '... /.. /bin/gzip'
/usr/bin/mzip:	enlace simbólico a 'mtools'

---

En este ejemplo, los resultados de la canalización se convirtieron en la lista de argumentos del comando file.

Hay una sintaxis alternativa para la sustitución de comandos en programas de shell más antiguos que también es compatible con bash. Utiliza *comillas inversas* en lugar del signo dolar y paréntesis:

---

```
[me@linuxbox ~]$ ls -l ' que cp'
-rwxr-xr-x 1 root root 71516 2012-12-05 08:58 /bin/cp
```

---

## Citando

Ahora que hemos visto de cuántas maneras el shell puede realizar expansiones, es hora de aprender cómo podemos controlarlo. Por ejemplo, tomemos esto:

---

```
[me@linuxbox ~]$ Hazte eco de esto es un      prueba
Esto es una prueba
```

---

O esto:

---

```
[me@linuxbox ~]$ echo El total es de $100.00
El total es 00.00
```

---

En el primer ejemplo, la *división de palabras* por parte del shell eliminó el espacio en blanco adicional de la lista de argumentos del comando echo. En el segundo ejemplo, la expansión de parámetros sustituyó una cadena vacía por el valor de \$1 porque era una variable indefinida. El shell proporciona un mecanismo llamado *comillas* para suprimir selectivamente las expansiones no deseadas.

### Comillas dobles

El primer tipo de comillas que veremos son *las comillas dobles*. Si coloca texto entre comillas dobles, todos los caracteres especiales utilizados por el shell pierden su significado especial y se tratan como caracteres ordinarios. Las excepciones son:

\$ (signo de dólar), \ (barra invertida) y ' (acento invertido). Esto significa que se suprime la división de palabras, la expansión de nombres de ruta, la expansión de tilde y la expansión de llaves, pero se siguen llevando a cabo la expansión de parámetros, la expansión aritmética y la sustitución de comandos. Usando comillas dobles, podemos hacer frente a nombres de archivos que contienen espacios incrustados. Digamos que fuimos la desafortunada víctima de un archivo llamado *dos words.txt*. Si intentamos usar esto en la línea de comandos, la división de palabras haría que esto se

tratará como dos argumentos separados en lugar del único argumento deseado:

```
[me@linuxbox ~]$ ls -l dos words.txt  
ls: no se puede acceder a dos: No existe tal archivo o directorio  
ls: no se puede acceder a words.txt: No existe tal archivo o directorio
```

Mediante el uso de comillas dobles, detenemos la división de palabras y obtenemos el resultado deseado; Además, incluso podemos reparar el daño:

---

```
[me@linuxbox ~]$ ls -l "dos words.txt"
-rw-rw-r-- 1 meme 18 2012-02-20 13:03 dos words.txt
[me@linuxbox ~]$ MV "Dos words.txt" two_words.txt
```

---

¡Allí! Ahora no tenemos que seguir escribiendo esas molestas comillas dobles.

Recuerde: La expansión de parámetros, la expansión aritmética y la sustitución de comandos todavía tienen lugar entre comillas dobles:

---

```
[me@linuxbox ~]$ echo "$USER $((2+2)) $(cal)"
me 4febrero 2012 Su
Mo Tu We Th Fr Sa
```

```
      1 2 3 4
      5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29
```

---

Deberíamos tomarnos un momento para analizar el efecto de las comillas dobles en la sustitución de comandos. Primero, veamos un poco más a fondo cómo funciona la división de palabras. En nuestro ejemplo anterior, vimos cómo la división de palabras parece eliminar espacios adicionales en nuestro texto:

---

```
[me@linuxbox ~]$ Hazte eco de esto es un      prueba
Esto es una prueba
```

---

De forma predeterminada, la división de palabras busca la presencia de espacios, tabulaciones y saltos de línea (caracteres de salto de línea) y los trata como *delimitadores* entre palabras. Esto significa que los espacios sin comillas, las tabulaciones y los saltos de línea no se tienen en cuenta para formar parte del texto. Sirven solo como separadores. Dado que separan las palabras en diferentes argumentos, nuestra línea de comandos de ejemplo contiene un comando seguido de cuatro argumentos distintos. Sin embargo, si añadimos comillas dobles, se suprime la división de palabras y los espacios incrustados no se tratan como delimitadores; más bien, se convierten en parte del argumento:

---

```
[me@linuxbox ~]$ eco "esto es un      prueba"
Esto es un      prueba
```

---

Una vez que se agregan las comillas dobles, nuestra línea de comandos contiene un comando seguido de un solo argumento.

El hecho de que las nuevas líneas se consideren delimitadores por el mecanismo de división de palabras causa un efecto interesante, aunque sutil, en la sustitución de comandos. Tenga en cuenta lo siguiente:

---

```
[me@linuxbox ~]$ echo $(cal)
```

---

```
Febrero 2012 su mo tu we th fr sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17  
18 19 20 21 22 23 24 25 26 27 28 29  
[me@linuxbox ~]$ echo "$(cal)"
```

Su	Mo	Tu	We	Th	Fr	Sa
1	2	3	4			
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29			

---

En el primer caso, la sustitución de comandos sin comillas dio como resultado una línea de comandos que contenía 38 argumentos; En el segundo, el resultado fue una línea de comandos con 1 argumento que incluye los espacios incrustados y las nuevas líneas.

### Comillas simples

Si necesitamos suprimir *todas las expansiones*, usamos *comillas simples*. A continuación se muestra una comparación de comillas sin comillas, comillas dobles y comillas simples:

---

```
[me@linuxbox -]$ echo texto ~/.txt {a,b} $(echo foo) $((2+2)) $USER
Texto /Inicio/Me/ls-output.txt a b Foo 4 Me
[me@linuxbox -]$ echo "texto ~/.txt {a,b} $(echo foo) $((2+2)) $USER"
texto ~/.txt {a,b} foo 4 me
[me@linuxbox -]$ echo 'texto ~/.txt {a,b} $(echo foo) $((2+2)) $USER'
texto ~/.txt {a,b} $(echo foo) $((2+2)) $USER
```

---

Como podemos ver, con cada nivel sucesivo de cotización, se suprime más y más expansiones.

### Personajes de escape

A veces queremos citar un solo personaje. Para hacer esto, podemos preceder a un carácter con una barra invertida, que en este contexto se denomina *carácter de escape*. A menudo, esto se hace entre comillas dobles para evitar selectivamente una expansión.

---

```
[me@linuxbox -]$ echo "El saldo para el $USER del usuario es: \$5.00"
El saldo para el usuario yo es: $5.00
```

---

También es común usar el escape para eliminar el significado especial de un carácter en un nombre de archivo. Por ejemplo, es posible usar caracteres en nombres de archivo que normalmente tienen un significado especial para el shell. Entre ellas se incluyen las siguientes:

\$, !, &, (un espacio), y otros. Para incluir un carácter especial en un nombre de archivo, puede hacer lo siguiente:

---

```
[me@linuxbox -]$ mv malo\&nombre de archivo good_filename
```

---

Para permitir que aparezca un carácter de barra diagonal inversa, escápelo escribiendo \\|. Tenga en cuenta que dentro de las comillas simples, la barra invertida pierde su significado especial y se trata como un carácter ordinario.

## SECUENCIA DE BARRA INVERTIDA ESCAPE

Además de su función como carácter de escape, la barra diagonal inversa también se utiliza como parte de una notación para representar ciertos caracteres especiales denominados códigos de control. Los primeros 32 caracteres del esquema de codificación ASCII se utilizan para transmitir comandos a dispositivos similares al teletipo. Algunos de estos códigos son familiares (tabulación, retroceso, avance de línea y retorno de carro), mientras que otros no lo son (nulo, fin de transmisión y reconocimiento), como se muestra en la Tabla 7-2.

Tabla 7-2: Secuencias de escape de barra invertida

Escapar Secuencia	Significado
\un	Bell ("alerta": hace que la computadora emita un pitido)
\b	Retroceso
\n	Newline (en sistemas tipo Unix, esto produce un salto de línea)
\r	Carruaje devolución
\t	Pestaña

En esta tabla se enumeran algunas de las secuencias de escape de barra invertida comunes. La idea detrás del uso de la barra invertida se originó en el lenguaje de programación C y ha sido adoptada por muchos otros, incluido el shell.

Agregar la opción `-e` a `echo` permitirá la interpretación de las secuencias de escape. También puede colocarlos dentro de `$'`. Aquí, usando el comando `sleep`, un programa simple que solo espera el número especificado de segundos y luego se cierra, podemos crear un temporizador de cuenta regresiva primitivo.

```
dormir 10; echo -e "Se acabó el tiempo\b"
```

También podríamos hacer esto:

```
dormir 10; echo "Se acabó el tiempo" \$\b'
```

## Nota final

A medida que avancemos en el uso del shell, encontraremos que las expansiones y las citas se usarán con una frecuencia cada vez mayor, por lo que tiene sentido obtener una buena comprensión de la forma en que funcionan. De hecho, se podría argumentar que son los temas más importantes para aprender sobre el caparazón. Sin una comprensión adecuada de la expansión, el caparazón siempre será una fuente de misterio y confusión, y gran parte de su poder potencial se desperdiciará.

# 8

## **TRUCOS AVANZADOS DE TECLADO**

A menudo describo en broma a Unix como "el sistema operativo para la gente a la que le gusta escribir". Por supuesto, el hecho de que incluso tenga una línea de comandos es un testimonio de ello. Pero a los usuarios de la línea de comandos no les gusta escribir *tanto*. ¿Por qué, si no, tantos comandos tienen nombres tan cortos, como cp, ls, mv y rm?

De hecho, uno de los objetivos máspreciados de la línea de comandos es la pereza: hacer la mayor cantidad de trabajo con la menor cantidad de pulsaciones de teclas. Otro objetivo es no tener que levantar nunca los dedos del teclado, nunca alcanzar el ratón. En este capítulo, veremos las características de bash que hacen que el uso del teclado sea más rápido y eficiente.

Aparecerán los siguientes comandos:

- clear: borra la pantalla.
- history: muestra el contenido de la lista de historial.

## Edición de línea de comandos

bash usa una biblioteca (una colección compartida de rutinas que diferentes programas pueden usar) llamada *Readline* para implementar la edición de la línea de comandos. Ya hemos visto algo de esto. Sabemos, por ejemplo, que las teclas de flecha mueven el cursor, pero hay muchas más características. Piense en estas como herramientas adicionales que podemos emplear en nuestro trabajo. No es importante aprenderlos todos, pero muchos de ellos son muy útiles. Elija y elija como deseé.

**Nota:** *Algunas de las secuencias de teclas que se indican a continuación (en particular las que utilizan la tecla ALT) pueden ser intercepidas por la GUI para otras funciones. Todas las secuencias de teclas deben funcionar correctamente cuando se utiliza una consola virtual.*

### Movimiento del cursor

En la tabla 8-1 se enumeran las teclas utilizadas para mover el cursor.

Tabla 8-1: Comandos de movimiento del cursor

Llave	Acción
CTRL-U	Mueva el cursor al principio de la línea.
CTRL-E	Mueva el cursor al final de la línea.
CTRL-F	Mueva el cursor hacia adelante un carácter; igual que la tecla de flecha derecha. CTRL-B Mueva el cursor hacia atrás un carácter; igual que la tecla de flecha izquierda. ALT-F Mueva el cursor hacia adelante una palabra.
ALT-B	Mueva el cursor hacia atrás una palabra.
CTRL-L	Despeje la pantalla y muela el cursor a la esquina superior izquierda. El clear hace lo mismo.

### Modificación de texto

En la tabla 8-2 se enumeran los comandos de teclado que se utilizan para editar caracteres en la línea de comandos.

### Cortar y pegar (matar y tirar) texto

La documentación de Readline utiliza los términos *matar* y *tirar* para referirse a lo que comúnmente llamaríamos cortar y pegar. En la Tabla 8-3 se enumeran los comandos para cortar y pegar. Los elementos que se cortan se almacenan en un búfer llamado *anillo de muerte*.

**Tabla 8-2: Comandos de edición de texto**

Llave	Acción
CTRL-D	Elimine el carácter en la ubicación del cursor.
CTRL-T	Transponga (intercambie) el carácter en la ubicación del cursor con el que lo precede.
ALT-T	Transponga la palabra en la ubicación del cursor con la que la precede.
ALT-L	Convierta los caracteres de la ubicación del cursor hasta el final de la palabra a minúsculas.
ALT-U	Convierta los caracteres de la ubicación del cursor al final de la palabra a mayúsculas.

**Tabla 8-3: Comandos de cortar y pegar**

Llave	Acción
CTRL-K línea.	Elimina el texto desde la ubicación del cursor hasta el final de la línea.
CTRL-U	Mata el texto desde la ubicación del cursor hasta el principio de la línea.
ALT-D actual.	Elimina el texto desde la ubicación del cursor hasta el final de la palabra actual.
ALT-RETROCESO	Elimina el texto desde la ubicación del cursor hasta el principio de la palabra actual. Si el cursor está al principio de una palabra, elimine la palabra anterior.
CTRL-Y	Arranca el texto del anillo de muerte e insértalo en la ubicación del cursor.

### La clave meta

Si se aventura en la documentación de Readline, que se puede encontrar en la sección "READLINE" de la página del manual de bash , encontrará el término *meta clave*. En los teclados modernos, esto se asigna a la tecla ALT , pero no siempre fue así.

En los tiempos oscuros (antes de las PC, pero después de Unix) no todo el mundo tenía su propia computadora. Lo que podrían haber tenido era un dispositivo llamado *terminal*. Un terminal era un dispositivo de comunicación que contaba con una pantalla de visualización de texto y un teclado, y tenía en su interior la electrónica suficiente para mostrar caracteres de texto y mover el cursor. Se conectaba (generalmente por cable serie) a una computadora más grande o a la red de comunicación de una computadora más grande. Había muchas marcas diferentes de terminales, y todos tenían diferentes teclados y



Los desarrolladores que desean aplicaciones portátiles escriben en el denominador común más bajo. Los sistemas Unix tienen una forma muy elaborada de tratar con los terminales y sus diferentes características de visualización. Dado que los desarrolladores de Readline no podían estar seguros de la presencia de una tecla de control adicional dedicada, inventaron una y la llamaron *meta*. Si bien la tecla ALT sirve como tecla meta en los teclados modernos, también puede presionar y soltar la tecla ESC para obtener el mismo efecto que mantener presionada la tecla ALT si todavía está usando un terminal (¡lo que aún puede hacer en Linux!).

## Terminación

Otra forma en que el shell puede ayudarte es a través de un mecanismo llamado *compleción*. La finalización se produce cuando se pulsa la tecla TAB mientras se escribe un comando. Veamos cómo funciona esto. Digamos que tu directorio de inicio se ve así:

---

```
[me@linuxbox ~]$ ls
Escritorio  ls-output.txt  Plantillas de imágenes      Videos
Documentos  Música          Público
```

---

Intente escribir lo siguiente, pero *no presione la tecla ENTER*:

---

```
[me@linuxbox ~]$ ls l
```

---

Ahora presione la tecla TAB :

---

```
[me@linuxbox ~]$ ls ls-output.txt
```

---

¿Ves cómo el caparazón completó la línea para ti? Probemos con otro. De nuevo, no pulses ENTER:

---

```
[me@linuxbox ~]$ ls D
```

---

Presione TAB:

---

```
[me@linuxbox ~]$ ls D
```

---

No hay finalización, solo un pitido. Esto sucedió porque D coincide con más de una entrada en el directorio. Para que la finalización sea exitosa, la "pista" que le des tiene que ser inequívoca. Podemos ir más allá:

---

```
[me@linuxbox ~]$ ls Do
```

---

A continuación, pulse TAB:

---

```
[me@linuxbox ~]$ ls Documentos
```

---

La finalización es exitosa.

Si bien este ejemplo muestra la finalización de los nombres de ruta, que es el uso más común de la compleción, la finalización también funcionará en variables (si el comienzo de la palabra es \$), nombres de usuario (si la palabra comienza con -), comandos (si la palabra es la primera palabra en la línea) y nombres de host (si el comienzo de la palabra es @). La finalización de nombre de host solo funciona para los nombres de host enumerados en /etc/hosts.

Una serie de secuencias de teclas de control y meta están asociadas con la finalización (véase la Tabla 8-4).

**Tabla 8-4: Comandos de finalización**

Llave	Acción
ALT-?	Mostrar lista de posibles finalizaciones. En la mayoría de los sistemas, también puede hacer esto presionando el botón PESTAÑA tecla una segunda vez, que es mucho más fácil.
ALT-*	Inserte todas las terminaciones posibles. Esto es útil cuando desea utilizar más de una coincidencia posible.

Hay bastantes más que me parecen bastante oscuros. Puede ver una lista en la página de manual de bash en la sección "READLINE".

### FINALIZACIÓN PROGRAMABLE

Las versiones recientes de bash tienen una función llamada *finalización programable*. La finalización programable le permite a usted (o, más probablemente, a su proveedor de distribución) agregar reglas de finalización adicionales. Por lo general, esto se hace para agregar soporte para aplicaciones específicas. Por ejemplo, es posible agregar finalizaciones para la lista de opciones de un comando o hacer coincidir tipos de archivo particulares que admite una aplicación.

Ubuntu tiene un conjunto bastante grande definido de forma predeterminada. La finalización programable se implementa mediante funciones de shell, una especie de mini script de shell que trataremos en capítulos posteriores. Si tienes curiosidad, prueba

Conjunto I menos

## Uso del historial

Como descubrimos en el Capítulo 1, bash mantiene un historial de comandos que se han introducido. Esta lista de comandos se guarda en su directorio de inicio

en un archivo llamado *.bash\_history*. La función de historial es un recurso útil para reducir la cantidad de escritura que tiene que hacer, especialmente cuando se combina con la edición de la línea de comandos.



## *Historial de búsqueda*

En cualquier momento, podemos ver el contenido de la lista de historial:

---

```
[me@linuxbox ~]$ historial | menos
```

---

De forma predeterminada, bash almacena los últimos 500 comandos que ha introducido. Veremos cómo ajustar este valor en el Capítulo 11. Digamos que queremos encontrar los comandos que usamos para listar `/usr/bin`. Esta es una forma en que podríamos hacer esto:

---

```
[me@linuxbox ~]$ historial | grep /usr/bin
```

---

Y digamos que entre nuestros resultados obtuvimos una línea que contiene un comando interesante como este:

---

```
88 ls -l /usr/bin > ls-output.txt
```

---

El número 88 es el número de línea del comando en la lista del historial. Podríamos usar esto inmediatamente con otro tipo de expansión llamada *expansión de historial*. Para usar nuestra línea descubierta, podríamos hacer lo siguiente:

---

```
[me@linuxbox ~]$ !88
```

---

bash expandirá `!88` al contenido de la línea 88 en la lista de historial. Cubriremos otras formas de expansión de la historia un poco más adelante.

bash también ofrece la posibilidad de buscar en la lista de historial de forma incremental. Esto significa que podemos decirle a bash que busque en la lista de historial a medida que ingresamos caracteres, con cada carácter adicional refinando aún más nuestra búsqueda. Para iniciar una búsqueda incremental, escriba CTRL-R seguido del texto que está buscando. Cuando lo encuentre, puede presionar ENTRAR para ejecutar el comando o presionar CTRL-J para copiar la línea de la lista de historial a la línea de comandos actual. Para encontrar la siguiente aparición del texto (moviéndose "hacia arriba" en la lista del historial), presione CTRL-R nuevamente. Para salir de la búsqueda, presione CTRL-G o CTRL-C. Aquí lo vemos en acción:

---

```
[me@linuxbox ~]$
```

---

Primero presione CTRL-R:

---

(búsqueda inversa)":

---

El mensaje cambia para indicar que estamos realizando una búsqueda incremental inversa. Es "al revés" porque estamos buscando desde el "ahora" hasta algún momento en el pasado. A continuación, comenzamos a escribir nuestro texto de búsqueda, que en este ejemplo es `/usr/bin`:

---

(búsqueda inversa)'**/usr/bin**: ls -l /usr/bin > ls-output.txt

Inmediatamente, la búsqueda devuelve su resultado. Ahora podemos ejecutar el comando presionando ENTER, o podemos copiar el comando a nuestra línea de comandos actual para editarlo más adelante presionando CTRL-J. Vamos a copiarlo. Presione CTRL-J:

```
[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt
```

¡Nuestro indicador de shell regresa y nuestra línea de comandos está cargada y lista para la acción!

En la tabla 8-5 se enumeran algunas de las pulsaciones de teclas utilizadas para manipular la lista de historial.

**Tabla 8-5: Comandos de historial**

Llave	Acción
CTRL-P	Vaya a la entrada del historial anterior. La misma acción que la flecha hacia arriba. <small>Dependiendo del terminal</small>
CTRL-N	Pasar a la siguiente entrada del historial. La misma acción que la flecha hacia abajo. <small>Dependiendo del terminal</small>
ALT->	Vaya al final (abajo) de la lista del historial; es decir, la línea de comandos actual.
CTRL-R	Búsqueda incremental inversa. Busca de forma incremental desde la línea de comandos actual en la lista de historial.
ALT-P	Búsqueda inversa, no incremental. Con esta tecla, escriba la cadena de búsqueda y presione ENTRAR antes de que se realice la búsqueda.
ALT-N	Búsqueda directa, no incremental.
CTRL-O	Ejecute el elemento actual en la lista del historial y avance al siguiente. Esto es útil si está intentando volver a ejecutar una secuencia de comandos en la lista de historial.

### *Expansión de la historia*

El shell ofrece un tipo especializado de expansión para los elementos de la lista de historial mediante el uso de ! carácter. Ya hemos visto cómo el signo de exclamación puede ir seguido de un número para insertar una entrada de la lista del historial. Hay una serie de otras características de expansión (consulte la Tabla 8-6).

Advertiría contra el uso de la cadena !string y !? formas de cadena a menos que esté absolutamente seguro del contenido de los elementos de la lista de historial.

Hay muchos más elementos disponibles en el mecanismo de expansión de la historia, pero este tema ya es demasiado arcano y nuestras cabezas pueden explotar si continuamos. La sección "EXPANSIÓN DEL HISTORIAL"

de la página del manual de bash entra en todos los detalles sangrientos.  
¡Siéntete libre de explorar!

**Tabla 8-6: Comandos de expansión del historial**

Secuencia	Acción
<code>!!</code>	Repite el último comando. Probablemente sea más fácil presionar la flecha hacia arriba y <code>ENTRAR</code> .
<code>!número</code>	Repetir elemento de lista de historial <i>número</i> .
<code>!cuerda</code>	Repetir el último elemento de la lista de historial que comienza con <i>cuerda</i> .
<code>!?cuerda</code>	Repetir el último elemento de la lista de historial que contiene <i>cuerda</i> .

## GUIÓN

Además de la función de historial de comandos en bash, la mayoría de las distribuciones de Linux incluyen un programa llamado `script`, que se puede usar para grabar una sesión de shell completa y almacenarla en un archivo. La sintaxis básica del comando es

```
script [archivo]
```

donde *file* es el nombre del archivo utilizado para almacenar la grabación. Si no se especifica ningún archivo, se utiliza el script *mecanografiado del archivo*. Consulte la página del manual del `script` para obtener una lista completa de las

## Nota final

En este capítulo hemos cubierto *algunos* de los trucos de teclado que proporciona el shell para ayudar a los mecanógrafos hardcore a reducir sus cargas de trabajo. Sospecho que a medida que pase el tiempo y te involucres más con la línea de comandos, consultarás este capítulo para aprender más de estos trucos. Por ahora, considérelos opcionales y potencialmente útiles.

# 9

## PERMISOS

Los sistemas operativos de la tradición Unix difieren de los de la tradición MS-DOS en que no son sólo sistemas *multitarea*, sino también sistemas *multiusuario*.

¿Qué significa esto exactamente? Significa que más de una persona puede usar la computadora al mismo tiempo. Si bien es probable que una computadora típica tenga solo un teclado y un monitor, aún puede ser utilizada por más de un usuario. Por ejemplo, si una computadora está conectada a una red o a Internet, los usuarios remotos pueden iniciar sesión a través de ssh (shell seguro) y operar la computadora. De hecho, los usuarios remotos pueden ejecutar aplicaciones gráficas y hacer que la salida gráfica aparezca en una pantalla remota. El sistema X Window admite esto como parte de su diseño básico.

La capacidad multiusuario de Linux no es una "innovación" reciente, sino más bien una característica que está profundamente integrada en el diseño del sistema operativo.

Teniendo en cuenta el entorno en el que se creó Unix, esto tiene mucho sentido. Hace años, antes de que las computadoras fueran "personales", eran grandes, costosas y centralizadas. Un sistema informático universitario típico, por ejemplo, constaba de una gran computadora central ubicada en un edificio y terminales ubicadas en todo el campus, cada una conectada al gran computadora central. La computadora admitiría muchos usuarios al mismo

tiempo.



Para que esto fuera práctico, se tuvo que idear un método para proteger a los usuarios entre sí. Después de todo, no se podía permitir que las acciones de un usuario bloquearan la computadora, ni un usuario podía interferir con los archivos que pertenecían a otro usuario.

En este capítulo vamos a ver esta parte esencial de la seguridad del sistema e introducir los siguientes comandos:

- `id`: muestra la identidad del usuario.
- `chmod`: cambia el modo de un archivo.
- `umask`: defina los permisos de archivo predeterminados.
- `su`: ejecute un shell como otro usuario.
- `sudo`: ejecute un comando como otro usuario.
- `chown`: cambia el propietario de un archivo.
- `chgrp`: cambie la propiedad del grupo de un archivo.
- `passwd`: cambia la contraseña de un usuario.

## Propietarios, miembros del grupo y todos los demás

Cuando estábamos explorando el sistema en el Capítulo 4, es posible que nos hayamos encontrado con el siguiente problema al intentar examinar un archivo como `/etc/shadow`:

---

```
[me@linuxbox ~]$ fichero /etc/shadow  
/etc/shadow: fichero normal, sin permiso de lectura  
[me@linuxbox ~]$ less /etc/shadow  
/etc/shadow: Permiso denegado
```

---

El motivo de este mensaje de error es que, como usuarios habituales, no tenemos permiso para leer este archivo.

En el modelo de seguridad de Unix, un usuario puede *ser propietario de* archivos y directorios. Cuando un usuario es propietario de un archivo o directorio, el usuario tiene control sobre su acceso. Usuarios pueden, a su vez, pertenecer a un *grupo* formado por uno o más usuarios a los que sus propietarios dan acceso a archivos y directorios. Además de conceder acceso a un grupo, un propietario también puede conceder algún conjunto de derechos de acceso a todo el mundo, lo que en términos de Unix se conoce como el *mundo*. Para obtener información sobre su identidad, utilice el comando `id`:

---

```
[me@linuxbox ~]$ id  
preguntar?500(me) gid=500(me) grupos=500(me)
```

---

Echemos un vistazo a la salida. Cuando se crean cuentas de usuario, a los usuarios se les asigna un número llamado *ID de usuario* o *uid*. Esto luego, por el bien de los humanos, se asigna a un nombre de usuario. Al usuario se le asigna un ID de *grupo principal*, o *gid*, y puede pertenecer a grupos adicionales. El ejemplo anterior es de un sistema Fedora. En otros sistemas, como Ubuntu, la salida puede ser un poco diferente.

---

```
[me@linuxbox ~]$ id  
UID=1000(ME) GID=1000(ME)  
groups=4(ADM),20(dialout),24(CDROM),25(FLOPPY),29(audio),30(DIP),44(video),46(  
plugdev),108(lpadmin),114(admin),1000(ME)
```

---

Como podemos ver, los números uid y gid son diferentes. Esto se debe simplemente a que Fedora comienza su numeración de cuentas de usuario regular en 500, mientras que Ubuntu comienza en 1000. También podemos ver que el usuario de Ubuntu pertenece a muchos más grupos. Esto tiene que ver con la forma en que Ubuntu administra los privilegios para los dispositivos y servicios del sistema.

Entonces, ¿de dónde proviene esta información? Como tantas cosas en Linux, proviene de un par de archivos de texto. Las cuentas de usuario se definen en el *archivo /etc/passwd* y los grupos se definen en el *archivo /etc/group*. Cuando se crean cuentas de usuario y grupos, estos archivos se modifican junto con */etc/shadow*, que contiene información sobre la contraseña del usuario. Para cada cuenta de usuario, el *archivo /etc/passwd* define el nombre de usuario (login), el uid, el gid, el nombre real de la cuenta, el directorio de inicio y el shell de inicio de sesión. Si examina el contenido de */etc/passwd* y */etc/group*, notará que además de las cuentas de usuario regulares, hay cuentas para el superusuario (uid 0) y varios otros usuarios del sistema.

En el Capítulo 10, cuando cubrimos los procesos, verá que algunos de estos otros "usuarios" están, de hecho, bastante ocupados.

Mientras que muchos sistemas tipo Unix asignan usuarios regulares a un grupo común, como *los usuarios*, la práctica moderna de Linux es crear un grupo único de un solo miembro con el mismo nombre que el usuario. Esto facilita la asignación de ciertos tipos de permisión.

## Leer, escribir y ejecutar

Los derechos de acceso a archivos y directorios se definen en términos de acceso de lectura, acceso de escritura y acceso de ejecución. Si nos fijamos en la salida del comando *ls* , podemos obtener alguna pista sobre cómo se implementa esto:

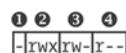
---

```
[me@linuxbox ~]$ > foo.txt  
[me@linuxbox ~]$ ls -l foo.txt  
-rw-rw-r-- 1 yo      me  0 06/03/2012 14:52 foo.txt
```

---

Los primeros 10 caracteres de la lista son los atributos del *archivo* (consulte la Figura 9-1). El primero de estos caracteres es el *tipo de archivo*. La Tabla 9-1 enumera los tipos de archivos que es más probable que vea (también hay otros tipos menos comunes).

Los nueve caracteres restantes de los atributos de archivo , denominados *modo de archivo*, representan



- ① File type (see Table 9-1)
- ② Owner permissions (see Table 9-2)
- ③ Group permissions (see Table 9-2)
- ④ World permissions (see Table 9-2)

Los permisos de lectura, escritura y ejecución para el propietario del archivo, el propietario del grupo del archivo y todos los demás.

*Figura 9-1: Desglose de los atributos del archivo*

Cuando se establecen, los atributos de modo r, w y x tienen ciertos efectos en archivos y directorios, como se muestra en la Tabla 9-2.

**Tabla 9-1: Tipos de archivos**

Atributo	Archivo Tipo
-	Un archivo normal.
d	Un directorio.
l	Un vínculo simbólico. Tenga en cuenta que con los enlaces simbólicos, los atributos de archivo restantes siempre son rwxrwxrwx y son valores ficticios. Los atributos reales del archivo son los del archivo al que apunta el enlace simbólico.
c	Un <i>Archivo especial de caracteres</i> . Este tipo de archivo hace referencia a un dispositivo que maneja datos como un flujo de bytes, como un terminal o un módem.
b	Un <i>Bloquear archivo especial</i> . Este tipo de archivo se refiere a un dispositivo que maneja datos en bloques, como un disco duro o una unidad de CD-ROM.

**Tabla 9-2: Atributos de permiso**

Atributo	Archivos	Directarios
r	Permite abrir y leer un archivo.	Permite la propiedad de un directorio El contenido que se enumerará si El atributo execute es también establecido.
w	Permite que un archivo se escriba o se trunque. De hecho, la mayoría de las personas que se Sin embargo, este atributo no Permitir que se cambie el nombre o se eliminan archivos. La capacidad de eliminar o cambiar el nombre de los archivos es determinado por los atributos del directorio.	Permite archivos dentro de un archivo directorio que se va a crear, eliminado, y renombrado si El atributo execute es también establecido.
x	Permite que un archivo sea tratado como un programa y ejecutado. Archivos de programa writ- diez en lenguajes de scripting también deben establecerse como legible para ser ejecutado.	Permite que un directorio ser introducido; p ej. cd.

La Tabla 9-3 muestra algunos ejemplos de configuraciones de atributos de archivo.

**Tabla 9-3: Ejemplos de atributos de permiso**

Archivo	Atributos	Significado
	-rwx-----	Un archivo normal que es legible, escribible y ejecutable por el propietario del archivo. Nadie más tiene acceso.
	-Rw-----	Un archivo normal que es legible y escribible por el archivo dueño. Nadie más tiene acceso.

### Cuadro 9-3 (continuación )

Archivo Atributos Significado	
-Rw-r--r--	Un archivo normal que es legible y escribible por el propietario del archivo. Los miembros del grupo de propietarios del archivo pueden leerlo. El archivo es legible por todo el mundo.
-rwxr-xr-x	Un archivo normal que es legible, escribible y ejecutable por el propietario del archivo. El archivo puede ser leído y ejecutado por todos los demás.
-rw-rw----	Un archivo normal que solo puede leer y escribir el propietario del archivo y los miembros del grupo de propietarios del archivo.
lrwxrwxrwx	Un vínculo simbólico. Todos los enlaces simbólicos tienen permisos "ficticios". Los permisos reales se mantienen con el archivo real al que apunta el enlace simbólico.
drwxrwx---	Un directorio. El propietario y los miembros del grupo de propietarios pueden entrar en el directorio y crear, cambiar el nombre y eliminar archivos dentro del directorio.
drwxr-x---	Un directorio. El propietario puede ingresar al directorio y crear, cambiar el nombre y eliminar archivos dentro del directorio. Los miembros del grupo de propietarios pueden entrar en el directorio, pero no pueden crear, eliminar ni cambiar el nombre de los archivos.

### *chmod: cambiar el modo de archivo*

Para cambiar el modo (permisos) de un archivo o directorio, se utiliza el comando chmod. Tenga en cuenta que solo el propietario del archivo o el superusuario puede cambiar el modo de un archivo o directorio. chmod admite dos formas distintas de especificar cambios de modo: la representación de números octales y la representación simbólica. Primero cubriremos la representación de números octales.

### **Representación Octal**

Con la notación octal usamos números octales para establecer el patrón de permisiones deseadas. Dado que cada dígito de un número octal representa tres dígitos binarios, esto se corresponde muy bien con el esquema utilizado para almacenar el modo de archivo. La tabla 9-4 muestra lo que queremos decir.

**Tabla 9-4: Modos de archivo en binario y octal**

Octal	Binario	Modo de archivo
-------	---------	-----------------

0	000	- - -
1	001	--x
2	010	-w-

*(continuació  
n)*

Cuadro 9-4 (continuación )

Octal	Binario	Modo de archivo
3	011	-wx
4	100	r--
5	101	R-X
6	110	Rw-
7	111	rwx

### ¡QUÉ DIABLOS ES OCTAL?

El octal (base 8) y su primo el hexadecimal (base 16) son sistemas numéricos que se utilizan a menudo para expresar números en ordenadores. Nosotros, los humanos, debido al hecho de que (o al menos la mayoría de nosotros) nacimos con 10 dedos, contamos usando un sistema de números de base 10. Las computadoras, por otro lado, nacieron con un solo dedo y, por lo tanto, hacen todas sus cuentas en binario (base 2). Su sistema numérico tiene solo dos números, cero y uno. Entonces, en binario, el conteo se ve así: 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011 . . .

En octal, el conteo se hace con los números del cero al siete, así: 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21 . . .

El conteo hexadecimal utiliza los números del cero al nueve más las letras de la A a la F: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13 . . .

Si bien podemos ver el sentido en binario (ya que las computadoras tienen un solo fin), ¿para qué sirven el octal y el hexadecimal? La respuesta tiene que ver con la conveniencia humana. Muchas veces, pequeñas porciones de datos se representan en las computadoras como patrones de *bits*. Tomemos, por ejemplo, un color RGB. En la mayoría de las pantallas de computadora, cada píxel se compone de tres componentes de color: 8 bits de rojo, 8 bits de verde y 8 bits de azul. Un hermoso azul medio sería un número de 24 dígitos: 01000011011011111001101.

¿Te gustaría leer y escribir ese tipo de números todo el día? No lo creo. Aquí es donde otro sistema numérico ayudaría. Cada dígito de un número hexadecimal representa cuatro dígitos en binario. En octal, cada dígito representa tres dígitos binarios. Por lo tanto, nuestro azul medio de 24 dígitos podría condensarse en un número hexadecimal de 6 dígitos: 436FCD. Dado que los dígitos del número hexadecimal se "alinean" con los bits del número binario, podemos ver que el componente rojo de nuestro color es el 43, el verde el 6F y el azul el CD.

Hoy en día, la notación hexadecimal (a menudo llamada *hexadecimal*) es más común que la octal, pero como pronto veremos, la capacidad de octal para expresar tres bits de binario es muy útil.

Mediante el uso de tres dígitos octales, podemos establecer el modo de archivo para el propietario, el propietario del grupo y el mundo.

---

```
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 meme      0 2012-03-06 14:52 foo.txt
[me@linuxbox ~]$ CHMOD 600 foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-RW----- 1 me      me  0 06/03/2012 14:52 foo.txt
```

---

Al pasar el argumento 600, pudimos establecer los permisos de el propietario para leer y escribir mientras elimina todos los permisos del propietario del grupo y del mundo. Aunque recordar el mapeo octal a binario puede parecer un inconveniente, por lo general tendrá que usar solo algunos comunes:

7 (rwx), 6 (rw-), 5 (r-x), 4 (r--) y 0 (---).

### Representación simbólica

chmod también admite una notación simbólica para especificar modos de archivo. La notación simbólica se divide en tres partes: a quién afectará el cambio, qué operación se realizará y qué permiso se establecerá. Para especificar quién se ve afectado, se utiliza una combinación de los caracteres *u*, *g*, *o* y *a*, como se muestra en la Tabla 9-5.

**Tabla 9-5: Notación simbólica chmod**

Símbolo	Significado
<i>u</i>	Abreviatura de <i>usuario</i> pero significa el propietario del archivo o directorio.
<i>g</i>	Propietario del grupo.
<i>o</i>	Abreviatura de <i>otros</i> pero significa mundo.
<i>un</i>	Abreviatura de <i>todo</i> ; La combinación de <i>u</i> , <i>g</i> o <i>o</i> .

Si no se especifica ningún carácter, se asumirá todo. La operación puede ser *un +* que indica que se va a agregar un permiso, *un -* que indica que un permiso

sion se va a quitar, *o a =* que indica que sólo se van a aplicar los permisos especificados y que se van a quitar todos los demás.

Los permisos se especifican con los caracteres r, w y x. En la Tabla 9-6 se enumeran algunos ejemplos de notación simbólica.

**Tabla 9-6: Ejemplos de notación simbólica chmod**

Notación	Significado
<i>u+x</i>	Agregue el permiso de ejecución para el propietario.
<i>U-X</i>	Quite el permiso de ejecución del propietario.
<i>+x</i>	Agregue el permiso de ejecución para el propietario, el grupo y el mundo.
	Equivalente a <i>a+x</i> .

(continuación)

Cuadro 9-6 (*continuación* )

Notación	Significado
O-RW	Quite los permisos de lectura y escritura de cualquier persona que no sea el propietario y el propietario del grupo.
go=rw	Establezca el propietario del grupo y cualquier otra persona que no sea el propietario para que tenga permiso de lectura y escritura. Si el propietario del grupo o el mundo tenían permisos de ejecución anteriormente, elimínelos.
u+x,go=rx	Agregue el permiso de ejecución para el propietario y establezca los permisos para que el grupo y otros usuarios lean y ejecuten. Varias especificaciones pueden estar separadas por comas.

Algunas personas prefieren usar la notación octal; A algunas personas les gusta mucho lo simbólico. La notación simbólica ofrece la ventaja de permitirle establecer un solo atributo sin alterar ninguno de los demás.

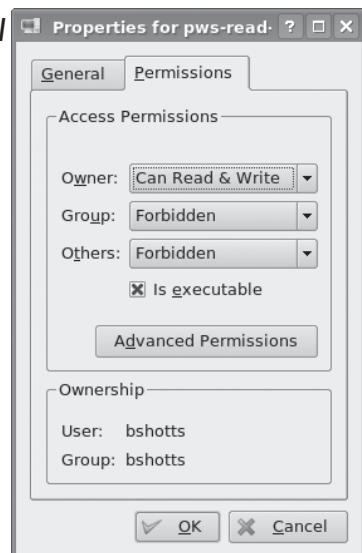
Eche un vistazo a la página del manual de chmod para obtener más detalles y una lista de opciones. Una palabra de precaución con respecto a la opción --recursive: actúa tanto en archivos como en directorios, por lo que no es tan útil como cabría esperar porque rara vez queremos que los archivos y directorios tengan los mismos permisos.

### *Configuración del modo de archivo con la GUI*

Ahora que hemos visto cómo se establecen los permisos en los archivos y directorios, podemos comprender mejor los diálogos de permisos en la GUI. Tanto en Nautilus (GNOME) como en Konqueror (KDE), al hacer clic con el botón derecho del ratón en el ícono de un archivo o directorio, se expondrá un cuadro de diálogo de propiedades. La figura 9-2 es un ejemplo de KDE 3.5.

Aquí podemos ver la configuración para el propietario, el grupo y el mundo. En KDE, al hacer clic en el botón Permisos avanzados, aparece otro cuadro de diálogo que le permite establecer cada uno de los atributos del modo individualmente. ¡Otra victoria para la comprensión que nos trae la línea de comandos!

### *umask: establecer permisos predeterminados*



El comando umask controla los permisos

predeterminados otorgados a un archivo cuando se crea. Utiliza la notación octal para expresar una *máscara* de bits que se eliminarán de los atributos de modo de un archivo.

Figura 9-2: Cuadro de diálogo  
Propiedades de archivo de KDE  
3.5

Echemos un vistazo:

---

```
[me@linuxbox ~]$ rm -f foo.txt
[me@linuxbox ~]$ umask
0002
[me@linuxbox ~]$ > foo.txt
[me@linuxbox ~]$ ls -l foo.txt
-rw-rw-r-- 1 yo      me  06/03/2012 14:53 foo.txt
```

---

Primero retiramos cualquier copia existente de *foo.txt* para asegurarnos de que estábamos empezando de nuevo. A continuación, ejecutamos el comando *umask* sin un argumento para ver el valor actual. Respondió con el valor 0002 (el valor 0022 es otro valor predeterminado común), que es la representación octal de nuestra máscara. A continuación, creamos una nueva instancia del archivo *foo.txt* y observamos sus permisos.

Podemos ver que tanto el propietario como el grupo obtienen permisos de lectura y escritura, mientras que todos los demás solo obtienen permiso de lectura. El mundo no tiene permiso de escritura debido al valor de la máscara. Repitamos nuestro ejemplo, esta vez colocando la mascarilla nosotros mismos:

---

```
[me@linuxbox ~]$ rm foo.txt
[me@linuxbox ~]$ umask 0000
[me@linuxbox ~]$ > foo.txt [me@linuxbox
~]$ ls -l foo.txt
-rw-rw-rw- 1 yo      me  06/03/2012 14:58 foo.txt
```

---

Cuando establecemos la máscara en 0000 (desactivándola efectivamente), vemos que el archivo ahora se puede escribir en todo el mundo. Para entender cómo funciona esto, tenemos que mirar de nuevo los números octales. Si expandimos la máscara a binario y luego la comparamos con los atributos, podemos ver lo que sucede:

Modo de archivo original	--- rw- rw- rw-
Máscara	000 000 000 010
Resultado	--- rw- rw- r--

Ignora por el momento los 0 iniciales (llegaremos a ellos en un minuto) y observa que donde aparece el 1 en nuestra máscara, se eliminó un atributo, en este caso, el permiso de escritura mundial. Eso es lo que hace la mascarilla. En cualquier lugar donde aparezca un 1 en el valor binario de la máscara, se desestablece un atributo. Si nos fijamos en un valor de máscara de 0022, podemos ver lo que hace:

Modo de archivo original	--- rw- rw- rw-
Máscara	000 000 000 010
Resultado	--- rw- r-----

De nuevo, cuando aparece un 1 en el valor binario, el atributo correspondiente no está establecido. Juegue con algunos valores (pruebe con algunos 7) para acostumbrarse a cómo funciona esto. Cuando termines, recuerda limpiar:

---

```
[me@linuxbox ~]$ rm foo.txt; umask 0002
```

---

La mayoría de las veces no tendrás que cambiar la mascara; El valor predeterminado proporcionado por su distribución será correcto. Sin embargo, en algunas situaciones de alta seguridad, querrá controlarlo.

## ALGUNOS PERMISOS ESPECIALES L

Aunque normalmente vemos una máscara de permiso octal expresada como un número de tres dígitos, es técnicamente más correcto expresarla en cuatro dígitos. ¿Por qué?

Porque, además de los permisos de lectura, escritura y ejecución, hay otras opciones de configuración de permisos menos utilizadas.

El primero de ellos es el *bit setuid* (octal 4000). Cuando se aplica a un archivo ejecutable, establece el *ID de usuario efectivo* desde el del usuario real (el usuario que realmente ejecuta el programa) hasta el del propietario del programa. La mayoría de las veces, esto se da a unos pocos programas propiedad del superusuario. Cuando un usuario ordinario ejecuta un programa que es setuid *root*, el programa se ejecuta con los privilegios efectivos del superusuario. Esto permite que el programa acceda a archivos y directorios a los que normalmente se le prohibiría el acceso a un usuario común.

Claramente, debido a que esto plantea problemas de seguridad, el número de programas setuid debe mantenerse al mínimo absoluto.

La segunda configuración menos utilizada es el *bit setgid* (octal 2000). Esto, al igual que el bit setuid, cambia el *ID de grupo efectivo* del ID de *grupo real* del usuario al del propietario del archivo. Si el bit setgid se establece en un directorio, a los archivos recién creados en el directorio se les dará la propiedad de grupo del directorio en lugar de la propiedad de grupo del creador del archivo. Esto es útil en un directorio compartido cuando los miembros de un grupo común necesitan acceder a todos los archivos del directorio, independientemente del grupo principal del propietario del archivo.

El tercero se llama la *broca pegajosa* (octal 1000). Este es un vestigio del antiguo Unix, donde era posible marcar un archivo ejecutable como "no intercambiable". En los archivos, Linux ignora el bit pegajoso, pero si se aplica a un directorio, impide que los usuarios eliminan o cambien el nombre de los archivos a menos que el usuario sea el propietario del directorio, el propietario del archivo o el superusuario. Esto se usa a menudo para controlar el acceso a un directorio compartido, como */tmp*.

Estos son algunos ejemplos del uso de chmod con notación simbólica para establecer estos permisos especiales. En primer lugar, asigne setuid a un programa:

Programa CHMOD U+S

A continuación, asigne setgid a un directorio:

```
chmod g+s dir
```

Por último, asigne el bit pegajoso a un directorio:

```
chmod +t dir
```

Al ver la salida de ls, puede determinar los permisos especiales. He aquí algunos ejemplos. Primero, un programa que es setuid:

```
-rwsr-xr-x
```

Ahora, un directorio que tiene el atributo setgid:

```
drwxrwsr-x
```

Por último, un directorio con el bit pegajoso establecido:

```
drwxrwxrwt
```

## Cambio de identidades

En varias ocasiones, es posible que nos veamos en la necesidad de asumir la identidad de otro usuario. A menudo queremos obtener privilegios de superusuario para llevar a cabo alguna tarea administrativa, pero también es posible "convertirse" en otro usuario regular para realizar tareas como probar una cuenta. Hay tres formas de adoptar una identidad alternativa:

- Cierre la sesión y vuelva a iniciarla como usuario alternativo.
- Utilice el comando su.
- Utilice el comando sudo.

Nos saltaremos la primera técnica porque sabemos cómo hacerla y carece de la comodidad de las otras dos. Desde dentro de su propia sesión de shell, el comando su le permite asumir la identidad de otro usuario e iniciar una nueva sesión de shell con el ID de ese usuario o emitir un solo comando como ese usuario. El comando sudo permite a un administrador configurar un archivo de configuración llamado */etc/sudoers* y definir comandos específicos que usuarios particulares pueden ejecutar bajo una identidad asumida. La elección del comando que se va a utilizar está determinada en gran medida por la distribución de Linux que se utilice. Es probable que su distribución incluya ambos comandos, pero su configuración favorecerá uno u otro. Empezaremos por su.

### *su: ejecute un shell con ID de usuario y grupo sustitutos*

El comando su se utiliza para iniciar un shell como otro usuario. La sintaxis del comando es la siguiente:

```
su [-l] [usuario]
```

Si se incluye la opción `-l`, la sesión de shell resultante es un *shell de inicio de sesión* para el usuario especificado. Esto significa que se carga el entorno del usuario y el directorio de trabajo se cambia al directorio principal del usuario. Esto suele ser lo que queremos. Si no se especifica el usuario, se supone que es el superusuario. Nótese que (curiosamente) la `-l` puede abreviarse como `-`, que es la forma en que se usa con mayor frecuencia. Para iniciar un shell para el superusuario, haríamos lo siguiente:

---

```
[me@linuxbox ~]$ su -  
Contraseña:  
[root@linuxbox ~]#
```

---

Después de ingresar el comando, se nos solicita la contraseña del superusuario. Si se ingresa con éxito, aparece un nuevo indicador de shell que indica que este shell tiene privilegios de superusuario (el `#` final en lugar de un `$`) y que el directorio de trabajo actual es ahora el directorio de inicio para el superusuario (normalmente `/root`). Una vez en el nuevo shell, podemos llevar a cabo comandos como el superusuario. Cuando termine, introduzca `exit` para volver al shell anterior:

---

```
[root@linuxbox ~]# salir  
[me@linuxbox ~]$
```

---

También es posible ejecutar un solo comando en lugar de iniciar un nuevo comando interactivo usando `su` de esta manera:

```
su -c "comando"
```

Con este formulario, se pasa una sola línea de comandos al nuevo shell para su ejecución. Es importante encerrar el comando entre comillas, ya que no queremos que la expansión ocurra en nuestro shell sino en el nuevo shell:

---

```
[me@linuxbox ~]$ su -c 'ls -l /raíz/*'  
Contraseña:  
-rw----- 1 raíz raíz 754 2011-08-11 03:19 /raíz/anaconda-ks.cfg  
  
/root/Correo:  
total 0  
[me@linuxbox ~]$
```

---

### ***sudo: ejecutar un comando como otro usuario***

El comando `sudo` es similar a `su` en muchos aspectos, pero tiene algunas capacidades adicionales importantes. El administrador puede configurar `sudo` para permitir que un usuario ordinario ejecute comandos como un usuario diferente (generalmente el superusuario) de una manera muy controlada. En particular, un usuario puede estar restringido a uno o más comandos específicos y no a otros. Otra diferencia importante es que el uso de `sudo` no requiere acceso a la contraseña del superusuario. Para autenticarse con `sudo`, el usuario introduce su propia contraseña. Supongamos, por ejemplo, que `sudo` se ha configurado para permitirnos ejecutar un programa de copia de

seguridad ficticio llamado `backup_script`, que requiere privilegios de superusuario.

Con sudo se haría así:

---

```
[me@linuxbox ~]$ sudo backup_script  
Contraseña:  
Copia de seguridad del sistema Iniciando...
```

---

Después de ingresar el comando, se nos solicita nuestra contraseña (no la del superusuario), y una vez completada la autenticación, se lleva a cabo el comando especificado. Una diferencia importante entre su y sudo es que sudo no inicia un nuevo shell, ni carga el entorno de otro usuario. Esto significa que los comandos no necesitan entrecomillarse de manera diferente a como lo harían sin usar sudo. Tenga en cuenta que este comportamiento se puede anular especificando varias opciones. Consulte la página del manual sudo para obtener más detalles.

Para ver qué privilegios otorga sudo, use la opción -l para enumerarlos:

---

```
[me@linuxbox ~]$ sudo -l  
El usuario me puede ejecutar los siguientes comandos  
en este host: (TODOS) TODOS
```

---

## UBUNTU Y SUDO

Uno de los problemas recurrentes para los usuarios habituales es cómo realizar ciertas tareas que requieren privilegios de superusuario. Estas tareas incluyen la instalación y actualización de software, la edición de archivos de configuración del sistema y el acceso a dispositivos. En el mundo de las ventanas, esto se hace a menudo otorgando a los usuarios privilegios administrativos. Esto permite a los usuarios realizar estas tareas. Sin embargo, también permite que los programas ejecutados por el usuario tengan las mismas habilidades. Esto es deseable en la mayoría de los casos, pero también permite que *el malware* (software malicioso) como los virus se ejecute libremente en la computadora.

En el mundo Unix, siempre ha habido una mayor división entre usuarios regulares y administradores, debido a la herencia multiusuario de Unix. El enfoque adoptado en Unix es otorgar privilegios de superusuario solo cuando sea necesario. Para ello, se suelen utilizar los comandos su y sudo.

Hasta hace unos años, la mayoría de las distribuciones de Linux dependían de su para este propósito. su no requería la configuración que sudo requería, y tener

una cuenta root es tradicional en Unix. Esto introdujo un problema. Los usuarios se vieron tentados a operar como root innecesariamente. De hecho, algunos usuarios operaban sus sistemas exclusivamente como usuario root, porque eliminaba todos esos molestos mensajes de "permiso denegado". Así es como se reduce la seguridad de un sistema Linux a la de un sistema Windows. No es una buena idea.

Cuando se introdujo Ubuntu, sus creadores tomaron un rumbo diferente. De forma predeterminada, Ubuntu deshabilita los inicios de sesión en la cuenta raíz (al no establecer una contraseña para la cuenta) y, en su lugar, usa sudo para otorgar privilegios de superusuario. A la cuenta de usuario inicial se le concede acceso completo a los privilegios de superusuario a través de sudo y puede

conceder poderes similares a las cuentas de usuario posteriores.

## ***chown: cambiar el propietario del archivo y el grupo***

El comando chown se utiliza para cambiar el propietario y el propietario del grupo de un archivo o directorio. Se requieren privilegios de superusuario para usar este comando. La sintaxis de chown es la siguiente:

```
chown [propietario][:[grupo]] archivo...
```

chown puede cambiar el propietario del archivo y/o el propietario del grupo de archivos dependiendo del primer argumento del comando. En la Tabla 9-7 se enumeran algunos ejemplos.

**Tabla 9-7: Ejemplos de argumentos de Chown**

Argumento	Resultados
bob	Cambia la propiedad del archivo de su propietario actual a usuario <i>bob</i> .
bob:usuarios	Cambia la propiedad del archivo de su propietario actual a usuario <i>bob</i> y cambia el propietario del grupo de archivos a grupo <i>Usuarios</i> .
:administradores	Cambia el propietario del grupo por el grupo <i>Administradores</i> . El propietario del archivo no cambia.
bob:	Cambiar el propietario del archivo de propietario actual a usuario <i>bob</i> y cambia el propietario del grupo al grupo de inicio de sesión del usuario <i>bob</i> .

Digamos que tenemos dos usuarios: *janet*, que tiene acceso a los privilegios de superusuario, y *tony*, que no. La usuaria *janet* quiere copiar un archivo de su directorio de inicio al directorio de inicio del usuario *tony*. Dado que el usuario *janet* quiere que *tony* pueda editar el archivo, *janet* cambia la propiedad del archivo copiado de *janet* a *tony*:

```
[janet@linuxbox ~]$ sudo cp myfile.txt ~tony  
Contraseña:  
[janet@linuxbox ~]$ sudo ls -l ~tony/myfile.txt  
-rw-r--r-- 1 root root 8031 2012-03-20 14:30 /home/tony/myfile.txt [janet@linuxbox  
~]$ sudo chown tony: ~tony/myfile.txt  
[janet@linuxbox ~]$ sudo ls -l ~tony/myfile.txt  
-rw-r--r-- 1 tony tony 8031 2012-03-20 14:30 /inicio/tony/myfile.txt
```

Aquí vemos a la usuaria *janet* copiar el archivo de su directorio al directorio de inicio del usuario *tony*. A continuación, *janet* cambia la propiedad del archivo de *root* (resultado del uso de sudo) a *tony*. Usando los dos puntos finales en el primer argumento, *janet* también cambió la propiedad del grupo del archivo al grupo de inicio de sesión de *tony*, que resulta ser el grupo *tony*.

¿Nota que después del primer uso de sudo, *no se le pidió a janet su contraseña*? Esto se debe a que sudo, en la mayoría de las configuraciones, "confía" en ti durante varios minutos (hasta que se agote el tiempo).

## **chgrp: cambiar la propiedad del grupo**

En versiones anteriores de Unix, el comando chown solo cambiaba la propiedad del archivo, no la propiedad del grupo. Para ese propósito se utilizó un comando separado, chgrp. Funciona de la misma manera que chown, excepto por ser más limitado.

## **Ejercer sus privilegios**

Ahora que hemos aprendido cómo funciona esto de los permisos, es hora de mostrarlo. Vamos a demostrar la solución a un problema común : configuración de un directorio compartido. Imaginemos que tenemos dos usuarios llamados *bill* y *karen*. Ambos tienen colecciones de CD de música y desean configurar un directorio compartido, donde cada uno almacenará sus archivos de música como Ogg Vorbis o MP3. User *Bill* tiene acceso a los privilegios de superusuario a través de sudo.

Lo primero que debe suceder es la creación de un grupo que tenga a *Bill* y *Karen* como miembros. Utilizando la herramienta gráfica de gestión de usuarios de GNOME, *bill* crea un grupo llamado *music* y añade a los usuarios *bill* y *karen* a él, como se muestra en la Figura 9-3.



Figura 9-3: Creación de un nuevo grupo con GNOME

A continuación, *bill* crea el directorio para los archivos de música:

---

```
[bill@linuxbox ~]$ sudo mkdir /usr/local/share/Música  
Contraseña:
```

---

Dado que *bill* está manipulando archivos fuera de su directorio de inicio, se requieren privilegios de superusuario. Una vez creado el directorio, tiene las siguientes propiedades y permisos:

---

```
[bill@linuxbox ~]$ ls -ld /usr/local/share/Música  
drwxr-xr-x 2 root root 4096 2012-03-21 18:05 /usr/local/share/Música
```

---

Como podemos ver, el directorio es propiedad de *root* y tiene 755 permisos. Para que este directorio se pueda compartir, *bill* debe cambiar la propiedad del grupo y los permisos del grupo para permitir la escritura:

```
[bill@linuxbox ~]$ sudo chown :música /usr/local/share/Música  
[bill@linuxbox ~]$ sudo chmod 775 /usr/local/share/Música  
[bill@linuxbox ~]$ ls -ld /usr/local/share/Música  
drwxrwxr-x 2 root music 4096 2012-03-21 18:05 /usr/local/share/Music
```

Entonces, ¿qué significa todo esto? Significa que ahora tenemos un directorio */usr/local/share/Música* que es propiedad de *root* y permite el acceso de lectura y escritura a la música de grupo. El grupo *de música* tiene como miembros a *bill* y *karen*, por lo que *bill* y *karen* pueden crear archivos en el directorio */usr/local/share/Music*. Otros usuarios pueden enumerar el contenido del directorio, pero no pueden crear archivos allí.

Pero todavía tenemos un problema. Con los permisos actuales, los archivos y direcciones creados dentro del directorio de música tendrán los permisos normales de los usuarios *bill* y *karen*:

```
[bill@linuxbox ~]$ > /usr/local/share/Music/test_file  
[bill@linuxbox ~]$ ls -l /usr/local/share/Música  
-rw-r--r-- 1 factura factura 0 24/03/2012 20:03 test_file
```

En realidad, hay dos problemas. En primer lugar, la máscara de usuario predeterminada en este sistema es 0022, lo que impide que los miembros del grupo escriban archivos que pertenezcan a otros miembros del grupo. Esto no sería un problema si la dirección compartida contuviera solo archivos, pero dado que este directorio almacenará música y la música generalmente está organizada en una jerarquía de artistas y álbumes, los miembros del grupo necesitarán la capacidad de crear archivos y directorios dentro de directorios creados por otros miembros. En su lugar, tenemos que cambiar la máscara de usuario utilizada por *Bill* y *Karen* a 0002.

En segundo lugar, cada archivo y directorio creado por un miembro se establecerá en el grupo principal del usuario, en lugar de en la música del grupo. Esto se puede arreglar configurando el bit setgid en el directorio:

```
[bill@linuxbox ~]$ sudo chmod g+s /usr/local/share/Música  
[bill@linuxbox ~]$ ls -ld /usr/local/share/Música  
drwxrwsr-x 2 root music 4096 2012-03-24 20:03 /usr/local/share/Music
```

Ahora probamos para ver si los nuevos permisos solucionan el problema. *Bill* establece su umask en 0002, elimina el archivo de prueba anterior y crea un nuevo archivo de prueba y directorio:

```
[bill@linuxbox ~]$ umask 0002  
[bill@linuxbox ~]$ rm /usr/local/share/Music/test_file  
[bill@linuxbox ~]$ > /usr/local/share/Music/test_file  
[bill@linuxbox ~]$ mkdir /usr/local/share/Music/test_dir  
[bill@linuxbox ~]$ ls -l /usr/local/share/Music  
Factura DRWXRWSR-X 2 música 4096 24/03/2012 20:24 test_dir  
-rw-rw-r-- 1 factura música 0 24/03/2012 20:22  
test_file [bill@linuxbox ~]$
```

Tanto los archivos como los directorios ahora se crean con los permisos correctos para permitir que todos los miembros del grupo *de música* creen archivos y directorios dentro del directorio de *música*.

El único problema pendiente es umask. La configuración necesaria dura solo hasta el final de la sesión y luego debe restablecerse. En el Capítulo 11, veremos cómo hacer que el cambio a umask sea permanente.

## Cambiar la contraseña

El último tema que trataremos en este capítulo es la configuración de contraseñas para usted (y para otros usuarios si tiene acceso a privilegios de superusuario). Para establecer o cambiar una contraseña, se utiliza el comando `passwd`. La sintaxis del comando es la siguiente:

```
passwd [usuario]
```

Para cambiar su contraseña, simplemente ingrese el comando `passwd`. Se le solicitará su contraseña anterior y su nueva contraseña:

---

```
[me@linuxbox ~]$ passwd  
(actual) Contraseña UNIX:  
Nueva contraseña de UNIX:
```

---

El comando `passwd` intentará imponer el uso de contraseñas "seguras". Esto significa que se negará a aceptar contraseñas que sean demasiado cortas, que sean demasiado similares a las contraseñas anteriores, que sean palabras del diccionario o que sean demasiado fáciles de adivinar:

---

```
[me@linuxbox ~]$ passwd  
(actual) Contraseña UNIX:  
Nueva contraseña de UNIX:  
MALA CONTRASEÑA: es demasiado similar a la  
anterior Nueva contraseña de UNIX:  
MALA CONTRASEÑA: es DEMASIADO  
corta Nueva contraseña de UNIX:  
BAD PASSWORD: se basa en una palabra del diccionario
```

---

Si tiene privilegios de superusuario, puede especificar un nombre de usuario como argumento para el comando `passwd` para establecer la contraseña para otro usuario. Hay otras opciones disponibles para el superusuario para permitir el bloqueo de la cuenta, la caducidad de la contraseña, etc. Consulte la página del manual `passwd` para obtener más información.



# 10

## PROCESOS

Los sistemas operativos modernos suelen ser *multitarea*, lo que significa que crean la ilusión de hacer más de una cosa a la vez al cambiar rápidamente de un programa en ejecución a otro. El kernel de Linux gestiona esto mediante el uso de *procesos*. Los procesos son la forma en que Linux organiza los diferentes programas que esperan su turno en la CPU.

A veces, una computadora se vuelve lenta o una aplicación deja de responder. En este capítulo, veremos algunas de las herramientas disponibles en la línea de comandos que nos permiten examinar lo que están haciendo los programas y cómo finalizar los procesos que se están comportando mal.

En este capítulo se presentarán los siguientes comandos:

- `ps`: informe una instantánea de los procesos actuales.
- `top`: tareas de visualización.
- `jobs`: enumera los trabajos activos.

- **bg**: coloque un trabajo en segundo plano.
- **fg**: coloque un trabajo en primer plano.
- **kill**: envía una señal a un proceso.
- **killall**: elimina procesos por nombre.
- **shutdown**: apague o reinicie el sistema.

## Cómo funciona un proceso

Cuando un sistema se inicia, el núcleo inicia algunas de sus propias actividades como procesos y lanza un programa llamado `init`. `init`, a su vez, ejecuta una serie de scripts de shell (ubicados en `/etc`) llamados *scripts de inicio*, que inician todos los servicios del sistema.

Muchos de estos servicios se implementan como *programas demonio*, programas que simplemente se quedan en segundo plano y hacen lo suyo sin tener ninguna interfaz de usuario. Entonces, incluso si no estamos conectados, el sistema está al menos un poco ocupado realizando cosas rutinarias.

El hecho de que un programa pueda lanzar otros programas se expresa en el esquema de proceso como un *proceso padre* que produce un *proceso hijo*.

El kernel mantiene información sobre cada proceso para ayudar a mantener las cosas organizadas. Por ejemplo, a cada proceso se le asigna un número llamado ID de *proceso (PID)*. Los PID se asignan en orden ascendente, y `init` siempre obtiene PID 1. El kernel también realiza un seguimiento de la memoria asignada a cada proceso, así como de la preparación de los procesos para reanudar la ejecución. Al igual que los archivos, los procesos también tienen propietarios e ID de usuario, ID de usuario efectivos, etc.

### Visualización de procesos con ps

El comando más utilizado para ver procesos (hay varios) es `ps`. El programa `ps` tiene muchas opciones, pero en su forma más simple se usa así:

---

```
[me@linuxbox ~]$ ps
  PID TTY          TIEMPO CMD
 5198 pts/1    00:00:00 bash
10129 pts/1    00:00:00 PS
```

---

El resultado de este ejemplo enumera dos procesos: el proceso 5198 y el proceso 10129, que son `bash` y `ps` respectivamente. Como vemos, por defecto `ps` no nos muestra mucho, solo los procesos asociados a la sesión actual del terminal. Para ver más, necesitamos agregar algunas opciones, pero antes de hacer eso, echemos un vistazo a los otros campos producidos por `ps`. `TTY` es la abreviatura de *teletipo* y se refiere al *terminal de control* del proceso. Unix está mostrando su edad aquí. El campo `TIME` es la cantidad de tiempo de CPU consumido por el proceso. Como podemos ver, ninguno de los dos procesos

hace que la computadora trabaje muy duro.

Si añadimos una opción, podemos obtener una imagen más amplia de lo que está haciendo el sistema:

```
[me@linuxbox ~]$ ps x
  PID TTY ESTAD HORA MANDAR
  2799 ? SSL 0:00 /usr/libexec/bonobo-activation-server -ac
  2820 ? SL 0:01 /usr/libexec/evolution-data-server-1.10 --
15647 ? Ss 0:00 /bin/sh/efecto/bin/startkdy
15751 ? Ss 0:00 /usr/bin/ssh-agent /usr/bin/dbus-launch --
15754 ? S 0:00 /usr/bin/dbus-launch --exit-with-session
15755 ? Ss 0:01 /bin/dbus-daemon --fork --print-pid 4 -pr
15774 ? Ss 0:02 /usr/bin/gpg-agent -s -demonio
15793 ? S 0:00 start_kdeinit --nueva-startup
          +kcminit_start
15794 ? Ss 0:00 kdeinit corriendo...
15797 ? S 0:00 dcopserver -nosid
```

y muchos más...

Agregar la opción x (tenga en cuenta que no hay un guión inicial) le dice a ps que muestre todos nuestros procesos, independientemente del terminal (si lo hay) por el que estén controlados. La presencia de un ? en la columna TTY indica que no hay terminal de control. Con esta opción, vemos una lista de todos los procesos que poseemos.

Dado que el sistema está ejecutando muchos procesos, ps produce una larga lista. A menudo es útil canalizar la salida de ps a menos para facilitar la visualización. Algunas combinaciones de opciones también producen largas líneas de salida, por lo que maximizar la ventana del emulador de terminal también puede ser una buena idea.

Se ha agregado una nueva columna titulada STAT a la salida. STAT es la abreviatura de y revela el estado actual del proceso, como se muestra en la Tabla 10-1.

**Tabla 10-1: Estados del proceso**

Estado	Significado
R	Corriente. El proceso se está ejecutando o listo para ejecutarse.
S	Durmiente. El proceso no se está ejecutando; más bien, está esperando un evento, como una pulsación de tecla o un paquete de red.
D	Sueño ininterrumpido. El proceso está esperando E/S, como una unidad de disco.
T	Detenido. Se ha ordenado que el proceso se detenga (más sobre esto más adelante).
Z	Un proceso difunto o "zombie". Se trata de un proceso secundario que ha finalizado, pero que no ha sido limpiado por su proceso primario.
<	Un proceso de alta prioridad. Es posible otorgar más importancia a un proceso, dándole más tiempo en la CPU. Esta propiedad de un

proceso se llama *amabilidad*. Se dice que un proceso con alta prioridad es menos agradable porque toma más tiempo de la CPU, lo que deja menos para todos los demás.

- N Un proceso de baja prioridad. Un proceso con prioridad baja (un proceso bueno) obtendrá tiempo de procesador solo después de que se hayan atendido otros procesos con prioridad más alta.
-

El estado del proceso puede ir seguido de otros caracteres. Estos indican varias características exóticas del proceso. Consulte la página del manual de ps para obtener más detalles.

Otro conjunto popular de opciones es aux (sin un guión inicial). Esto nos da aún más información:

---

[me@linuxbox ~]\$ ps aux							INICIO DE ESTADÍSTICAS	COMANDO DE TIEMPO
USUARIO	PID	%CPU	%MEM	VSZ	RSS	TTY		
raíz	1	0.0	0.0	2136	644	?	Ss Mar05	0:31 inicio
raíz	2	0.0	0.0	0	0	?	S< Mar05	0:00 [kt]
raíz	3	0.0	0.0	0	0	?	S< Mar05	0:00 [mi]
raíz	4	0.0	0.0	0	0	?	S< Mar05	0:00 [ks]
raíz	5	0.0	0.0	0	0	?	S< Mar05	0:06 [wa]
raíz	6	0.0	0.0	0	0	?	S< Mar05	0:36 [EV]
raíz	7	0.0	0.0	0	0	?	S< Mar05	0:00 [kh]

y muchos más...

---

Este conjunto de opciones muestra los procesos que pertenecen a cada usuario. Al usar las opciones sin el guión inicial, se invoca el comando con un comportamiento de "estilo BSD". La versión Linux de ps puede emular el comportamiento del programa ps que se encuentra en varias implementaciones de Unix. Con estas opciones, obtenemos las columnas adicionales que se muestran en la Tabla 10-2.

Tabla 10-2: Encabezados de columna ps de estilo BSD

Encabezado	Significado
USUARIO	ID de usuario. Este es el propietario del proceso.
%CPU	Uso de CPU como porcentaje.
%MEM	Uso de memoria como porcentaje.
VSZ	Tamaño de la memoria virtual.
RSS	Tamaño del conjunto residente. La cantidad de memoria física (RAM) que utiliza el proceso en kilobytes.
EMPEZAR	Hora en que se inició el proceso. Para valores superiores a 24 horas, se utiliza una fecha.

### Visualización dinámica de procesos con top

Si bien el comando ps puede revelar mucho sobre lo que está haciendo la máquina, solo proporciona una instantánea del estado de la máquina en el momento en que se ejecuta el comando ps. Para ver una vista más dinámica de la actividad de la máquina, usamos el comando superior:

---

```
[me@linuxbox ~]$ arriba
```

---

El programa superior muestra una actualización continua (de forma predeterminada, cada 3 segundos) visualización de los procesos del sistema enumerados en orden de actividad

del proceso.

Su nombre proviene del hecho de que el programa top se utiliza para ver los procesos "top" en el sistema. La pantalla superior consta de dos partes: un resumen del sistema en la parte superior de la pantalla, seguido de una tabla de procesos ordenados por actividad de la CPU:

---

```
top - 14:59:20 hasta 6:30, 2 usuarios, promedio de carga: 0.07, 0.02, 0.00
Tareas: 109 en total, 1 corriendo, 106 durmiendo, 0 detenido, 2 CPU
zombie: 0.7%us, 1.0%sy, 0.0%ni, 98.3%id, 0.0%wa, 0.0%hi, 0.0%si Mem:
      319496k en total, 314860k usado,   4636k gratis,  19392k buff
Intercambio:      875500k en total,    149128k usados,  726372k
                gratis,  114676k caché
```

USUARIO	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIEMPO+ MANDAR
6244 yo	39	19	31752	3124	2188	S	6.3	1.0	16:24.42 rastreado
11071 yo	20	0	2304	1092	840	R	1,3	0,3	0:00.14 arriba
6180 yo	20	0	2700	1100	772	S	0,7	0,3	0:03.66 dbus-dae
6321 yo	20	0	20944	7248	6560	S	0,7	2,3	2:51.38 Multifacético
4955 raíz	20	0	104m	9668	5776	S	0,3	3,0	2:19.39 Xorg
1 raíz	20	0	2976	528	476	S	0,0	0,2	0:03.14 Init
2 Raíz	15	-5	0	0	0	S	0,0	0,0	0:00.00 Kthreadd
3 raíz	RT	-5	0	0	0	S	0,0	0,0	0:00.00 Relación Miga
4 Raíz	15	-5	0	0	0	S	0,0	0,0	0:00.72 ksoftirq
5 Raíz	RT	-5	0	0	0	S	0,0	0,0	0:00.04 perro guardián
6 Raíz	15	-5	0	0	0	S	0,0	0,0	0:00.42 eventos/0
7 Raíz	15	-5	0	0	0	S	0,0	0,0	0:00.06 Jugador
41 raíz	15	-5	0	0	0	S	0,0	0,0	0:01.08 kbloqueado/
67 raíz	15	-5	0	0	0	S	0,0	0,0	0:00.00 kseriod
114 raíz	20	0	0	0	0	S	0,0	0,0	0:01.62 pdflush
116 raíz	15	-5	0	0	0	S	0,0	0,0	0:02.44 kswapd0

---

El resumen del sistema contiene muchas cosas buenas; consulte la Tabla 10-3 para obtener un resumen.

**Tabla 10-3: Campos de información principales**

Fila	Campo	Significado
1	Arriba	Nombre del programa.
	14:59:20	Hora actual del día.
	hasta las 6:30	A esto se le llama <i>tiempo de actividad</i> . Es la cantidad de tiempo transcurrido desde La máquina se inició por última vez. En este ejemplo, el método El sistema ha estado activo durante 6 horas y media.
2	2 usuarios	Dos usuarios han iniciado sesión.
	Carga media:	<i>El promedio de carga</i> se refiere al número de procesos que están a la espera de correr; es decir, el número de pro- De esta manera, la mayoría de las personas que se encuentran en un estado de ejecución se encuentran en un estado de ejecución la CPU. Se muestran tres valores, cada uno para un De hecho, la mayoría de las personas que se encuentran en un período de tiempo El primero es el

promedio de la los últimos 60 segundos, los siguientes los 5 minutos anteriores, y finalmente los 15 minutos anteriores. Valores en 1.0 indica que la máquina no está ocupada.

---

(continuación)

Cuadro 10-3 (continuación )

Fila	Campo	Significado
2	Tareas:	Esto resume el número de procesos y sus diversos estados de proceso.
	0.7%nosotros	El 0,7% de la CPU se utiliza para <i>los procesos de usuario</i> . Esto significa procesos fuera del propio kernel.
	1.0%sy	El 1,0% de la CPU se utiliza para los procesos <i>del sistema</i> (kernel).
	0.0%ni	El 0,0% de la CPU está siendo utilizada por procesos agradables (de baja prioridad).
	98.3%id	El 98,3% de la CPU está inactiva.
	0.0%wa	El 0,0% de la CPU está esperando E/S.
4	Mem:	Muestra cómo se utiliza la RAM física.
5	Intercambio:	Muestra cómo se utiliza el espacio de intercambio (memoria virtual).

El programa superior acepta una serie de comandos de teclado. Los dos más interesantes son h, que muestra la pantalla de ayuda del programa, y q, que sale de la parte superior.

Los dos principales entornos de escritorio proporcionan aplicaciones gráficas que muestran información similar a top (de la misma manera que lo hace el Administrador de tareas en Windows), pero creo que top es mejor que las versiones gráficas porque es más rápido y consume muchos menos recursos del sistema. Después de todo, nuestro programa de monitoreo del sistema no debería aumentar la ralentización del sistema que estamos tratando de rastrear.

## Control de procesos

Ahora que podemos ver y monitorear los procesos, obtengamos cierto control sobre ellos. Para nuestros experimentos, vamos a usar un pequeño programa llamado xlogo como nuestro conejillo de indias. El programa xlogo es un programa de muestra suministrado con el sistema X Window (el motor subyacente que hace que los gráficos de nuestra pantalla funcionen), que simplemente muestra una ventana redimensionable que contiene el logotipo X. Primero, conoceremos a nuestro sujeto de prueba:

---

```
[me@linuxbox ~]$ xlogo
```

---

Después de ingresar el comando, debería aparecer una pequeña ventana que contiene el logotipo en algún lugar de la pantalla. En algunos sistemas, xlogo puede imprimir un mensaje de advertencia, pero puede ser ignorado de manera segura.

**Nota:** Si su sistema no incluye el programa xlogo, intente usar gedit o kwrite en su lugar.

Podemos verificar que xlogo se está ejecutando cambiando el tamaño de su ventana. Si el logotipo se vuelve a dibujar en el nuevo tamaño, el programa se está ejecutando.

¿Te das cuenta de que nuestro prompt de shell no ha regresado? Esto se debe a que el shell está esperando a que termine el programa, al igual que todos los demás programas que hemos utilizado hasta ahora. Si cerramos la ventana de xlogo, vuelve el prompt.

### Interrupción de un proceso

Observemos lo que sucede cuando volvemos a ejecutar xlogo. Primero, ingrese el comando xlogo y verifique que el programa se esté ejecutando. A continuación, regrese a la ventana terminal y presione CTRL-C.

---

```
[me@linuxbox ~]$ xlogo  
[me@linuxbox ~]$
```

---

En un terminal, al pulsar CTRL-C se *interrumpe* un programa. Esto significa que le pedimos cortésmente al programa que terminara. Después de presionar CTRL-C, la ventana xlogo se cerró y regresó el indicador de shell.

Muchos (pero no todos) los programas de línea de comandos se pueden interrumpir mediante esta técnica.

### Poner un proceso en segundo plano

Digamos que queremos recuperar el símbolo del shell sin terminar el programa xlogo. Para ello, colocaremos el programa en segundo *plano*. Piense en el terminal como si tuviera un *primer plano* (con cosas visibles en la superficie, como el aviso del shell) y un fondo (con cosas ocultas debajo de la superficie). Para lanzar un programa de modo que se coloque inmediatamente en segundo plano, seguimos el comando con un carácter & (&):

---

```
[me@linuxbox ~]$ xlogo &  
[1] 28236  
[me@linuxbox ~]$
```

---

Después de ingresar el comando, apareció la ventana xlogo y se devolvió el indicador del shell, pero también se imprimieron algunos números divertidos. Este mensaje forma parte de una característica de shell denominada *control de trabajos*. Con este mensaje, el shell nos está diciendo que hemos iniciado el trabajo número 1 ([1]) y que tiene PID 28236. Si ejecutamos ps, podemos ver nuestro proceso:

---

```
[me@linuxbox ~]$ ps  
 PID TTY          TIEMPO CMD  
10603 pts/1    00:00:00 bash  
28236 pts/1    00:00:00 xlogo  
28239 pts/1    00:00:00 PS
```

---

La función de control de trabajos del shell también nos da una forma de enumerar los trabajos que se han lanzado desde nuestro terminal. Usando el comando `jobs`, podemos ver la siguiente lista:

---

```
[me@linuxbox ~]$ empleos
[1]+ Correr           xlogo &
```

---

Los resultados muestran que tenemos un trabajo, numerado 1, que se está ejecutando y que el comando era `xlogo &`.

### *Devolver un proceso al primer plano*

Un proceso en segundo plano es inmune a la entrada del teclado, incluido cualquier intento de interrumpirlo con un `CTRL-C`. Para devolver un proceso al primer plano, utilice el comando `fg`, como en este ejemplo:

---

```
[me@linuxbox ~]$ empleos
[1]+ Runningxlogo & [me@linuxbox ~]$
fg %1
xlogo
```

---

El comando `fg` seguido de un signo de porcentaje y el número de trabajo (llamado *jobspec*) hace el truco. Si solo tenemos un trabajo en segundo plano, la especificación de trabajo es opcional. Para terminar `xlogo`, escriba `CTRL-C`.

### *Detener (pausar) un proceso*

A veces querremos detener un proceso sin terminarlo. Esto se hace a menudo para permitir que un proceso en primer plano se mueva a un segundo plano. Para detener un proceso en primer plano, escriba `CTRL-Z`. Vamos a intentarlo. En el símbolo del sistema, escriba `xlogo`, presione la tecla ENTRAR y, a continuación, escriba `CTRL-Z`:

---

```
[me@linuxbox ~]$ xlogo
[1]+ Detenido           xlogo
[me@linuxbox ~]$
```

---

Después de detener `xlogo`, podemos verificar que el programa se ha detenido intentando cambiar el tamaño de la ventana de `xlogo`. Veremos que aparece bastante muerto. Podemos restaurar el programa al primer plano, usando el comando `fg`, o mover el programa a un segundo plano con el comando `bg`:

---

```
[me@linuxbox ~]$ bg %1
[1]+ xlogo &
[me@linuxbox ~]$
```

---

Al igual que con el comando `fg`, *jobspec* es opcional si solo hay un trabajo. Mover un proceso del primer plano al fondo es útil si lanzamos un programa gráfico desde el comando pero olvidamos colocarlo en el fondo anexando el `&` final.

¿Por qué querría lanzar un programa gráfico desde la línea de comunicación? Hay dos razones. En primer lugar, es posible que el programa que desea ejecutar no aparezca en los menús del administrador de ventanas (como `xlogo`).

En segundo lugar, al iniciar un programa desde la línea de comandos, es posible que pueda ver mensajes de error que serían invisibles si el programa se iniciara gráficamente. A veces, un programa no se inicia cuando se inicia desde el menú gráfico. Al iniciar lo desde la línea de comandos, es posible que veamos un mensaje de error que revelará el problema. Además, algunos programas gráficos tienen muchas opciones de línea de comandos interesantes y útiles.

## Señales

El comando `kill` se utiliza para "matar" (terminar) procesos. Esto nos permite finalizar la ejecución de un programa que se está comportando mal o que se niega a terminar por sí solo. He aquí un ejemplo:

---

```
[me@linuxbox ~]$ xlogo &
[1] 28401
[me@linuxbox ~]$ matar 28401
[1]+ Terminado          xlogo
```

---

Primero lanzamos `xlogo` en segundo plano. El shell imprime la especificación de trabajo y el PID del proceso en segundo plano. A continuación, usamos el comando `kill` y especificamos el PID del proceso que queremos terminar. También podríamos haber especificado el proceso usando una especificación de trabajo (por ejemplo, `%1`) en lugar de un PID.

Si bien todo esto es muy sencillo, hay más. El comando de apagado no "mata" exactamente los procesos, sino que les envía *señales*. Las señales son una de las varias formas en que el sistema operativo se comunica con los programas. Ya hemos visto señales en acción con el uso de `CTRL-C` y `CTRL-Z`. Cuando el terminal recibe una de estas pulsaciones de teclas, envía una señal al programa en primer plano. En el caso de `CTRL-C`, se envía una señal llamada `INT` (Interrupt); con `CTRL-Z`, se envía una señal llamada `TSTP` (Terminal Stop). Los programas, a su vez, "escuchan" las señales y pueden actuar sobre ellas a medida que se reciben. El hecho de que un programa pueda escuchar y actuar sobre las señales le permite hacer cosas como guardar el trabajo en curso cuando se le envía una señal de terminación.

### *Envío de señales a procesos con kill*

La sintaxis más común para el comando `kill` es la siguiente:

`kill [-signal] PID...`

Si no se especifica ninguna señal en la línea de comandos, la señal `TERM` (Termin-ate) se envía de forma predeterminada. El comando `kill` se utiliza con mayor frecuencia para enviar las señales que se muestran en la Tabla 10-4.

Tabla 10-4: Señales comunes

Número	Nombre	Significado
1	HUP	<p>Cuelga. Este es un vestigio de los viejos tiempos cuando los terminales estaban conectados a computadoras remotas con líneas telefónicas y módems. La señal se utiliza para indicar a los programas que el terminal de control se ha "colgado". El efecto de esta señal se puede demostrar cerrando una sesión de terminal.</p> <p>Al programa en primer plano que se esté ejecutando en el terminal se le enviará la señal y terminará.</p> <p>Esta señal también es utilizada por muchos programas demonio para provocar una reinicialización. Esto significa que cuando se envía esta señal a un demonio, se reiniciará y volverá a leer su archivo de configuración. El servidor web Apache es un ejemplo de un demonio que utiliza la señal HUP de esta manera.</p>
2	INT	Interrumpir. Realiza la misma función que el <b>CTRL-C</b> clave enviada desde el terminal. Por lo general, terminará un programa.
9	MATAR	Matar. Esta señal es especial. Mientras que los programas pueden optar por manejar las señales que se les envían de diferentes maneras, incluso ignorándolas por completo, la señal KILL nunca se envía realmente al programa objetivo. Más bien, el núcleo finaliza inmediatamente el proceso. Cuando un proceso se termina de esta manera, no se le da la oportunidad de "limpiarse" a sí mismo o salvar su trabajo. Por esta razón, la señal KILL debe usarse solo como último recurso cuando fallan otras señales de terminación.
15	TÉRMINO	Terminar. Esta es la señal predeterminada enviada por el comando <b>kill</b> . Si un programa todavía está lo suficientemente "vivo" como para recibir señales, terminará.
18	CONT	Continuar. Esto restaurará un proceso después de un STOP señal.
19	PARAR	Parar. Esta señal hace que un proceso se detenga sin terminar. Al igual que la señal KILL, no se envía al proceso de destino y, por lo tanto, no se puede ignorar.

Probemos el comando de muerte:

---

```
[me@linuxbox ~]$ xlogo &
[1] 13546
[me@linuxbox ~]$ matar -1 13546
[1]+ Colgar          xlogo
```

---

En este ejemplo, iniciamos el programa xlogo en segundo plano y luego le enviamos una señal HUP con kill. El programa xlogo finaliza y el shell indica que el proceso en segundo plano ha recibido una señal de bloqueo. Es posible que tenga que pulsar la tecla ENTER un par de veces antes de ver el mensaje. Tenga en cuenta que las señales pueden especificarse por número o por nombre, incluido el nombre prefijado con las letras *SIG*:

---

```
[me@linuxbox ~]$ xlogo &
[1] 13601
[me@linuxbox ~]$ matar -INT 13601
[1]+ Interruptxlogo [me@linuxbox ~]$ 
[1]+ Interrupcion      xlogo
&
[1] 13608
[me@linuxbox ~]$ matar -SIGINT 13608
[1]+ Interrupcion      xlogo
```

---

Repita el ejemplo anterior y pruebe las otras señales. Recuerde que también puede usar jobspecs en lugar de PID.

Los procesos, al igual que los archivos, tienen dueños, y usted debe ser el propietario de un proceso (o el superusuario) para enviarle señales con kill.

Además de las señales enumeradas en la Tabla 10-4, que se utilizan con mayor frecuencia con kill, el sistema utiliza con frecuencia otras señales. En la Tabla 10-5 se enumeran las otras señales comunes.

**Tabla 10-5: Otras señales comunes**

Número	Nombre	Significado
3	RENUNCIA R	Renunciar.
11	SEGV	Violación de la segmentación. Esta señal se envía si un programa hace un uso ilegal de la memoria; es decir, trató de escribir en algún lugar donde no le estaba permitido.
20	Cucharada de té	Parada de terminal. Esta es la señal enviada por el terminal cuando se presiona CTRL-Z. A diferencia de la señal STOP, el La señal TSTP es recibida por el programa, pero el programa puede optar por ignorarla.

---

28	CABRESTANTE	Cambio de ventana. Esta es una señal enviada por el sistema cuando una ventana cambia de tamaño. Algunos programas, como top y less, responderán a esta señal redibujándose a sí mismos para adaptarse a la nueva ventana Dimensiones.
----	-------------	--

---

Para los curiosos, se puede ver una lista completa de señales con el siguiente comando:

---

```
[me@linuxbox ~]$ matar -l
```

---

### **Envío de señales a múltiples procesos con killall**

También es posible enviar señales a varios procesos que coincidan con un programa o nombre de usuario especificado mediante el comando **killall**. Esta es la sintaxis:

```
killall [-u usuario] [-signal] nombre...
```

Para demostrarlo, iniciaremos un par de instancias del programa **xlogo** y luego las terminaremos:

---

```
[me@linuxbox ~]$ xlogo &
[1] 18801
[me@linuxbox ~]$ xlogo &
[2] 18802
[me@linuxbox ~]$ killall xlogo
[1]- Terminado          xlogo
[2]+ Terminado          xlogo
```

---

Recuerde que, al igual que con **kill**, debe tener privilegios de superusuario para enviar imágenes a procesos que no le pertenecen.

## **Más comandos relacionados con el proceso**

Dado que el monitoreo de procesos es una tarea importante de administración del sistema, hay muchos comandos para ello. En la Tabla 10-6 se enumeran algunos con los que jugar.

**Tabla 10-6: Otros comandos relacionados con el proceso**

<b>Mandar</b>	<b>Descripción</b>
psárbol	Genera una lista de procesos organizada en un patrón en forma de árbol que muestra las relaciones padre/hijo entre los procesos.
vmstat	Genera una instantánea del uso de recursos del sistema, incluida la memoria, el intercambio y la E/S de disco. Para ver una visualización continua, siga el comando con un retardo de tiempo (en segundos) para las actualizaciones (p. ej., VMSTAT 5). Termine la salida con CTRL-C.
xload	Un programa gráfico que dibuja un gráfico que muestra la

---

carga del sistema a lo largo del tiempo.

tload	Similar a la función xload programa, sino que dibuja el gráfico en el terminal. Termine la salida con CTRL-C.
-------	---

# **PARTE 2**

**C ONFIGURACIÓN Y MEDIO  
AMBIENTE**



# 11

## EL MEDIO AMBIENTE

Como hemos comentado anteriormente, el shell mantiene un cuerpo de información durante nuestra sesión de shell llamado *el entorno*. Los datos almacenados en el entorno son utilizados por los programas para determinar hechos sobre nuestra configuración.

Si bien la mayoría de los programas usan *archivos de configuración* para almacenar la configuración del programa, algunos programas también buscarán valores almacenados en el entorno para ajustar su comportamiento. Sabiendo esto, podemos usar el entorno para personalizar nuestra experiencia de shell.

En este capítulo, trabajaremos con los siguientes comandos:

- `printenv`: imprime parte o la totalidad del entorno.
- `set`: defina las opciones de shell.
- `export`: exporte el entorno a los programas ejecutados posteriormente.
- `alias`: cree un alias para un comando.

## ¿Qué se almacena en el entorno?

El shell almacena dos tipos básicos de datos en el entorno, aunque, con bash, los tipos son en gran medida indistinguibles. Son *variables de entorno* y *variables de shell*. Las variables de shell son bits de datos colocados allí por bash, y las variables de entorno son básicamente todo lo demás. Además de las variables, el shell también almacena algunos datos programáticos, a saber, *alias* y *funciones del shell*. Cubrimos los alias en el Capítulo 5, y las funciones de shell (que están relacionadas con las secuencias de comandos de shell) se tratarán en la Parte 4.

### Examinando el medio ambiente

Para ver lo que se almacena en el entorno, podemos usar el conjunto integrado en bash o el programa printenv. El comando set mostrará tanto el shell como las variables de entorno, mientras que printenv mostrará solo esta última. Dado que la lista de contenidos del entorno será bastante larga, es mejor canalizar la salida de cualquiera de los comandos a menos:

---

```
[me@linuxbox ~]$ printenv | menos
```

---

Al hacerlo, deberíamos obtener algo que se vea así:

---

```
KDE_MULTIHEAD=falso
SSH_AGENT_PID=6666
HOSTNAME=linuxbox
GPG_AGENT_INFO=/tmp/gpg-PdOt7g/S.gpg-agent:6689:1
SHELL=/bin/bash
TERM=xterm
XDG_MENU_PREFIX=kde-
HISTSIZE=1000
XDG_SESSION_COOKIE=6d7b05c65846c3eaf3101b0046bd2b00-1208521990.996705-11770561
99
GTK2_RC_FILES=/etc/gtk-2.0/gtkrc:/home/me/.gtkrc-2.0:/home/me/.kde/share/config/gtkrc-2.0
GTK_RC_FILES=/etc/gtk/gtkrc:/home/me/.gtkrc:/home/me/.kde/share/config/gtkrc
GS_LIB=/home/me/.fonts
WINDOWID=29360136
QTDIR=/usr/lib/qt-3.3
QTINC=/usr/lib/qt-3.3/include
KDE_FULL_SESSION=true
USUARIO=yo LS_COLORS=no=00:fi=00:di=00; 34:ln=00; 36: pi=40; 33:so=00; 35: bd = 40;
33; 01: cd=40; 33
;01:o=01; 05; 37; 41:mi=01; 05; 37; 41:ex=00; 32:*.cmd=00; 32:*.exe:
```

---

Lo que vemos es una lista de variables de entorno y sus valores. Por ejemplo, vemos una variable llamada **USER**, que contiene el valor **me**. El comando printenv también puede enumerar el valor de una variable específica:

---

---

```
[me@linuxbox ~]$ printenv USUARIO  
me
```

El comando `set`, cuando se usa sin opciones o argumentos, mostrará tanto las variables de shell como las de entorno, así como cualquier función de shell definida.

---

```
[me@linuxbox ~]$ set | menos
```

---

A diferencia de `printenv`, su producción se ordena cortésmente en orden alfabetico.

También es posible ver el contenido de una sola variable usando el `echo` comando, como este:

---

```
[me@linuxbox ~]$ echo $HOME  
/inicio/yo
```

---

Un elemento del entorno que no se muestra ni `set` ni `printenv` son los alias. Para verlos, introduzca el comando `alias` sin argumentos:

---

```
[me@linuxbox ~]$ alias  
alias l='ls -d .* --color=tty'  
alias ll='ls -l --color=tty'  
alias ls='ls --color=tty'  
alias vi=vim  
alias which="alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
```

---

### *Algunas variables interesantes*

El entorno contiene bastantes variables, y aunque su entorno puede diferir del que se presenta aquí, es probable que vea las variables que se muestran en la Tabla 11-1 en su entorno.

Tabla 11-1: Variables de entorno

Variable	Contenido
MONITOR	El nombre de la pantalla si se está ejecutando un entorno gráfico. Por lo general, esto es <code>:0</code> , es decir, la primera visualización generada por el servidor X.
EDITOR	El nombre del programa que se utilizará para la edición de texto.
CÁSCARA	El nombre del programa de shell.
HOGAR	El nombre de la ruta de acceso del directorio principal.
LANG	Define el conjunto de caracteres y el orden de intercalación del idioma.
OLD_PWD	El directorio de trabajo anterior.
BUSCAPERSONAS	Nombre del programa que se va a utilizar para la salida de paginación. A menudo, esto se establece en <code>/usr/bin/less</code> .
CAMINO	Lista de directorios separados por dos puntos en los que se realiza una búsqueda al escribir el nombre de un programa ejecutable.

(continuación)



Cuadro 11-1 (*continuación* )

Variable	Contenido
PS1 de shell.	Cadena de solicitud 1. Esto define el contenido de su símbolo del sistema  Como veremos más adelante, esto se puede personalizar ampliamente.
PWD	El directorio de trabajo actual.
TÉRMINO	El nombre del tipo de terminal. Los sistemas tipo Unix soportan muchos protocolos de terminal; Esta variable establece el protocolo que se usará con el emulador de terminal.
TZ	Especifica la zona horaria. La mayoría de los sistemas tipo Unix mantienen el reloj interno de la computadora en <i>Tiempo Universal Coordinado (UTC)</i> y, a continuación, muestre la hora local aplicando un desplazamiento especificado por esta variable.
USUARIO	Tu nombre de usuario.

No se preocupe si faltan algunos de estos valores. Varían según la distribución.

## ¿Cómo se establece el entorno?

Cuando nos conectamos al sistema, el programa bash se inicia y lee una serie de scripts de configuración llamados archivos de *inicio*, que definen el entorno predeterminado compartido por todos los usuarios. A esto le siguen más archivos de inicio en nuestro Directorio de inicio que definen nuestro entorno personal. La secuencia exacta depende del tipo de sesión de shell que se esté iniciando.

### *Shells de inicio de sesión y sin inicio de sesión*

Hay dos tipos de sesiones de shell: una sesión de shell de inicio de sesión y una sesión de shell sin inicio de sesión.

Una sesión de *shell de inicio de sesión* es aquella en la que se nos solicita nuestro nombre de usuario y contraseña; por ejemplo, cuando iniciamos una sesión de consola virtual. Una *sesión de shell sin inicio de sesión* generalmente ocurre cuando iniciamos una sesión de terminal en la GUI.

Los shells de inicio de sesión leen uno o más archivos de inicio, como se muestra en la Tabla 11-2.

Tabla 11-2: Archivos de inicio para sesiones de shell de inicio de sesión

Archivo	Contenido
/etc/perfil	Un script de configuración global que se aplica a todos los usuarios.

<code>~/.bash_profile</code>	Archivo de inicio personal de un usuario. Se puede utilizar para ampliar o invalidar la configuración del script de configuración global.
------------------------------	---

Cuadro 11-2 (*continuación* )

Archivo	Contenido
<code>~/.bash_login</code>	Si <code>~/.bash_profile</code> no se encuentra, juerga intenta leer este script.
<code>~/.perfil</code>	Si ninguno de los dos <code>~/.bash_profile</code> ni <code>~/.bash_login</code> se encuentra, juerga intentos de leer este archivo. Este es el valor predeterminado en las distribuciones basadas en Debian, como Ubuntu.

Las sesiones de shell sin inicio de sesión leen los archivos de inicio como se muestra en la Tabla 11-3.

Tabla 11-3: Archivos de inicio para sesiones de shell sin inicio de sesión

Archivo	Contenido
<code>/etc/bash.bashrc</code> usuarios.	Un script de configuración global que se aplica a todos los
<code>~/.bashrc</code> ampliar o	Archivo de inicio personal de un usuario. Se puede utilizar para Anule la configuración en el script de configuración global.

Además de leer los archivos de inicio anteriores, los shells que no son de inicio de sesión heredan el entorno de su proceso principal, generalmente un shell de inicio de sesión.

Eche un vistazo a su sistema y vea cuál de estos archivos de inicio tiene. Recuerde: Dado que la mayoría de los nombres de archivo enumerados anteriormente comienzan con un punto (lo que significa que están ocultos), deberá usar la opción `-a` cuando use `ls`.

El archivo `~/.bashrc` es probablemente el archivo de inicio más importante desde el punto de vista del usuario común, ya que casi siempre se lee. Los shells que no son de inicio de sesión lo leen de forma predeterminada, y la mayoría de los archivos de inicio de sesión para los shells de inicio de sesión están escritos de tal manera que también leen el archivo `~/.bashrc`.

### *¿Qué hay en un archivo de inicio?*

Si echamos un vistazo al interior de una `.bash_profile` típica (tomada de un sistema CentOS-4), se ve algo así:

```
# .bash_profile

# Obtener los alias y funciones
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# Entorno de usuario específico y programas de inicio
PATH=$PATH:$HOME/bin
```

export PATH

---

Las líneas que comienzan con # son *comentarios* y no son leídas por el shell. Estos están ahí para la legibilidad humana. Lo primero interesante ocurre en la cuarta línea, con el siguiente código:

---

```
si [ -f ~/.bashrc ]; entonces  
    . ~/.bashrc  
fi
```

---

A esto se le llama un *comando compuesto if*, que cubriremos completamente cuando llegar a shell scripting en la Parte 4, pero por ahora vamos a traducir:

---

**Si el archivo " ~/.bashrc" existe, lea el archivo " ~/.bashrc".**

---

Podemos ver que este fragmento de código es la forma en que un shell de inicio de sesión obtiene el contenido de *.bashrc*. Lo siguiente en nuestro archivo de inicio tiene que ver con la variable PATH.

¿Alguna vez te has preguntado cómo el shell sabe dónde encontrar comandos cuando los ingresamos en la línea de comandos? Por ejemplo, cuando introducimos ls, el shell no busca en todo el equipo /bin/ls (el nombre completo de la ruta del comando ls), sino que busca en una lista de directorios que están contenidos en la variable PATH.

La variable PATH se establece a menudo (pero no siempre, dependiendo de la distribución) mediante el archivo *de inicio /etc/profile* y con este código:

---

```
RUTA=$PATH:$HOME/bin
```

---

PATH se modifica para agregar el directorio \$HOME/bin al final de la lista. Este es un ejemplo de expansión de parámetros, que mencionamos en el Capítulo 7. Para demostrar cómo funciona esto, pruebe lo siguiente:

---

```
[me@linuxbox ~]$ foo="Esto es algo"  
[me@linuxbox ~]$ echo $foo  
Se trata de algunos  
[me@linuxbox ~]$ foo=$foo" texto."  
[me@linuxbox ~]$ echo $foo  
Este es un texto .
```

---

Con esta técnica, podemos agregar texto al final del contenido de una variable.

Al agregar la cadena \$HOME/bin al final del contenido de la variable PATH, el directorio \$HOME/bin se agrega a la lista de directorios buscados cuando se ingresa un comando. Esto significa que cuando queremos crear un directorio dentro de nuestro directorio de inicio para almacenar nuestros propios programas privados, el shell está listo para acomodarnos. Todo lo que tenemos que hacer es llamarlo bin, y estamos listos para comenzar.

**Nota:** Muchas distribuciones proporcionan esta configuración de PATH de forma predeterminada. Algunas distribuciones basadas en Debian, como Ubuntu, prueban la existencia del directorio ~/bin al iniciar sesión y lo añaden dinámicamente a

*la variable PATH si se encuentra el directorio.*

Por último, tenemos esto:

---

```
export PATH
```

---

El comando `export` le dice al shell que haga que el contenido de `PATH` esté disponible para los procesos secundarios de este shell.

## Modificación del entorno

Dado que sabemos dónde están los archivos de inicio y qué contienen, podemos modificarlos para personalizar nuestro entorno.

### ¿Qué archivos debemos modificar?

Como regla general, para agregar directorios a su `PATH` o definir variables de entorno adicionales, coloque esos cambios en `.bash_profile` (o equivalente, de acuerdo con su distribución, por ejemplo, Ubuntu usa `.profile`). Para todo lo demás, coloque los cambios en `.bashrc`. A menos que sea el administrador del sistema y necesite cambiar los valores predeterminados para todos los usuarios del sistema, restrinja sus modificaciones a los archivos de su directorio principal. Ciertamente es posible cambiar los archivos en `/etc` como `profile`, y en muchos casos sería sensato hacerlo, pero por ahora vamos a ir a lo seguro.

### Editores de texto

Para editar (es decir, modificar) los archivos de inicio del shell, así como la mayoría de los otros archivos de configuración en el sistema, usamos un programa llamado editor de *texto*. Un editor de texto es un programa que es, en cierto modo, como un procesador de textos en el sentido de que permite editar las palabras en la pantalla con un cursor móvil. Se diferencia de un procesador de textos en que solo admite texto puro y, a menudo, contiene características diseñadas para escribir programas. Los editores de texto son la herramienta central utilizada por los desarrolladores de software para escribir código y por los administradores de sistemas para administrar los archivos de configuración que controlan el sistema.

Hay muchos editores de texto disponibles para Linux; Es probable que su sistema tenga varios instalados. ¿Por qué tantos diferentes?

Probablemente porque a los programadores les gusta escribirlos, y dado que los programadores utilizan ampliamente los editores, les gusta expresar sus propios deseos sobre cómo deberían trabajar los editores.

Los editores de texto se dividen en dos categorías básicas: gráficos y basados en texto.

GNOME y KDE incluyen algunos editores gráficos populares. GNOME viene con un editor llamado `gedit`, que generalmente se llama Editor de texto en el menú de GNOME. KDE generalmente se envía con tres, que son (en orden creciente de complejidad) `kedit`, `kwrite` y `kate`.

Hay muchos editores basados en texto. Los más populares que encontrarás son `nano`, `vi` y `emacs`. El `nano` editor es un editor simple y fácil de usar diseñado

como reemplazo del editor pico suministrado con la suite de correo electrónico PINE. El editor vi (en la mayoría de los sistemas Linux reemplazado por un programa llamado vim, que es la abreviatura *de Vi IMproved*) es el editor tradicional para sistemas tipo Unix. Es el

tema del Capítulo 12. El editor de emacs fue escrito originalmente por Richard Stallman. Es un entorno de programación gigantesco, polivalente, que lo hace todo. Aunque está disponible, rara vez se instala en la mayoría de los sistemas Linux de forma predeterminada.

### **Usar un editor de texto**

Todos los editores de texto se pueden invocar desde la línea de comandos escribiendo el nombre del editor seguido del nombre del archivo que desea editar. Si el archivo aún no existe, el editor asumirá que desea crear un nuevo archivo. A continuación, se muestra un ejemplo de uso de gedit:

---

```
[me@linuxbox ~]$ gedit some_file
```

---

Este comando iniciará el editor de texto gedit y cargará el archivo llamado *some\_file*, si existe.

Todos los editores de texto gráfico se explican por sí mismos, por lo que no los cubriremos aquí. En su lugar, nos concentraremos en nuestro primer editor de texto basado en texto, nano. Vamos a encender nano y editar el archivo *.bashrc*. Pero antes de hacer eso, practiquemos algo de computación segura. Siempre que editamos un archivo de configuración importante, siempre es una buena idea crear primero una copia de seguridad del archivo. Esto nos protege en caso de que estropeemos el archivo mientras editamos. Para crear una copia de seguridad del archivo *.bashrc*, haga lo siguiente:

---

```
[me@linuxbox ~]$ cp .bashrc .bashrc.bak
```

---

No importa cómo llames al archivo de copia de seguridad, simplemente elige un nombre comprensible. Las extensiones *.bak*, *.sav*, *.old* y *.orig* son formas populares de indicar un archivo de copia de seguridad. Ah, y recuerde que *cp* *sobrescribirá los archivos existentes de forma silenciosa*.

Ahora que tenemos un archivo de copia de seguridad, iniciaremos el editor:

---

```
[me@linuxbox ~]$ nano .bashrc
```

---

Una vez que nano se inicie, obtendremos una pantalla como esta:

```
ÑU nano 2.0.3          Archivo:  
.bashrc # .bashrc  
  
# Fuentes de definiciones  
globales si [ -f  
/etc/bashrc ]; entonces  
    . /etc/bashrc  
Fi  
  
# Alias y funciones específicas del usuario
```

[ Leer 8 Líneas ]

^G Obtener ayuda ^O WriteOut ^R leer fil ^Y prev pag ^K Cut text ^C cur pos  
^X Salida ^J Justificar ^W Donde Esta ^V Siguiente Pag ^U UnCut Te ^T Para Deletrear

**Nota:** Si su sistema no tiene nano instalado, puede utilizar un editor gráfico en su lugar.

La pantalla consta de un encabezado en la parte superior, el texto del archivo que se está editando en el medio y un menú de comandos en la parte inferior. Dado que nano fue diseñado para reemplazar el editor de texto suministrado con un cliente de correo electrónico, es bastante corto en funciones de edición.

El primer comando que debe aprender en cualquier editor de texto es cómo salir del programa. En el caso de nano, presiona CTRL-X para salir. Esto se indica en el menú en la parte inferior de la pantalla. La notación ^X significa CTRL-X. Esta es una notación común para los caracteres de control utilizados por muchos programas.

El segundo comando que necesitamos saber es cómo guardar nuestro trabajo. Con nano es CTRL-O. Con este conocimiento en nuestro cinturón, estamos listos para hacer algo de edición. Con la tecla de flecha abajo o la tecla para avanzar págs, mueva el cursor hasta el final del archivo y, a continuación, agregue las siguientes líneas a la carpeta Archivo .bashrc :

---

```
umask 0002
export HISTCONTROL=ignoredups
export HISTSIZE=1000
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
```

---

**Nota:** Es posible que tu distribución ya incluya algunos de estos, pero los duplicados no dañarán nada.

La Tabla 11-4 enumera los significados de nuestras adiciones.

Tabla 11-4: Adiciones a nuestro archivo .bashrc

Línea	Significado
Umask 0002Conjuntos	el Umask para resolver el problema con los directorios compartidos que discutimos en el Capítulo 9.
export HISTCONTROL=ignoredups	Provoca el registro del historial del shell para ignorar un comando si se acaba de grabar el mismo comando.
exportar HISTSIZE=1000Aumentos	El tamaño del historial de comandos desde el valor predeterminado de 500 líneas hasta 1000 líneas.
alias l.='ls -d .* --color=auto'	Crea un nuevo comando llamado l., que muestra todas las entradas del directorio que comienzan con un punto.
alias ll='ls -l -color=auto'	Crea un nuevo comando llamado LL, que muestra una lista de directorio de formato largo.

Como podemos ver, muchas de nuestras adiciones no son intuitivamente obvias, por lo que sería una buena idea agregar algunos comentarios a nuestro *archivo .bashrc* para ayudar a explicar las cosas a los humanos. Usando el editor, cambie nuestras adiciones para que se vean así:

---

```
# Cambiar umask para facilitar el uso compartido de directorios
umask 0002

# Ignorar duplicados en el historial de comandos y
# aumentar # el tamaño del historial a 1000 líneas
export HISTCONTROL=ignoredups
export HISTSIZE=1000

# Añade algunos alias útiles
alias l='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
```

---

¡Ah, mucho mejor! Una vez completados los cambios, pulsamos CTRL-O para guardar nuestro archivo *.bashrc* modificado y CTRL-X para salir de nano.

### *Activando nuestros cambios*

Los cambios que hayamos realizado en nuestro *.bashrc* no tendrán efecto hasta que cerremos nuestra sesión de terminal y comencemos una nueva, porque el *archivo .bashrc* solo se lee al comienzo de una sesión. Sin embargo, podemos forzar a bash a releer el archivo *.bashrc* modificado con el siguiente comando:

---

```
[me@linuxbox ~]$ fuente .bashrc
```

---

Después de hacer esto, deberíamos ser capaces de ver el efecto de nuestros cambios. Pruebe uno de los nuevos alias:

---

```
[me@linuxbox ~]$ ll
```

---

### **¿POR QUÉ LOS COMENTARIOS SON IMPORTANTES?**

Siempre que modifique los archivos de configuración, es una buena idea agregar algunos comentarios para documentar los cambios. Seguro que recordarás lo que cambiaste mañana, pero ¿qué pasa dentro de seis meses? Hazte un favor y añade algunos comentarios. Mientras lo haces, no es una mala idea llevar un registro de los cambios que haces.

Los scripts de shell y los archivos de inicio de bash usan un símbolo # para comenzar un comentario.

Otros archivos de configuración pueden utilizar otros símbolos. La mayoría de los archivos de configuración tendrán comentarios. Úsalos como guía.

A menudo verá líneas en los archivos de configuración que se comentan para pre-

Sintaxis de configuración. Por ejemplo, el archivo `.bashrc` de Ubuntu 8.04 contiene estas líneas:

```
# algunos alias ls más
#alias ll='ls -l'
#alias la='ls -A'
#alias l='ls -CF'
```

Las tres últimas líneas son definiciones de alias válidas que se han comentado. Si eliminas los símbolos `#` iniciales de estas tres líneas, una técnica llamada *descomentar*, activarás los alias. Por el contrario, si se añade un símbolo `#` al principio de una línea, se puede desactivar una línea de configuración conservando la información que contiene.

## Nota final

En este capítulo aprendimos una habilidad esencial: editar archivos de configuración con un editor de texto. A medida que leemos las páginas del manual en busca de comandos, tome nota de las variables de entorno que admiten los comandos. Puede haber una joya o dos. En capítulos posteriores aprenderemos sobre las funciones de shell, una característica poderosa que también puede incluir en los archivos de inicio de bash para agregar a su arsenal de comandos personalizados.



# 12

## **UNA AMABLE INTRODUCCIÓN A VI**

Hay un viejo chiste sobre un visitante de la ciudad de Nueva York que le pregunta a un transeúnte cómo llegar al famoso lugar de música clásica de la ciudad:

Visitante: Disculpe, ¿cómo llego al Carnegie Hall?  
Transeúnte: ¡Practica, practica, practica!

Aprender la línea de comandos de Linux, como convertirse en un pianista consumado, no es algo que aprendamos en una tarde. Se necesitan años de práctica. En este capítulo, presentaremos el editor de texto vi (pronunciado "vee eye"), uno de los programas centrales en la tradición Unix. Vi es algo notorio por su difícil interfaz de usuario, pero cuando vemos a un maestro sentarse en el teclado y comenzar a "tocar", seremos testigos de un gran arte. No nos convertiremos en maestros en este capítulo, pero cuando terminemos, sabremos cómo jugar a los "Palillos" en vi.

## Por qué debemos aprender vi

En esta era moderna de editores gráficos y editores basados en texto fáciles de usar como nano, ¿por qué deberíamos aprender vi? Hay tres buenas razones:

- Vi siempre está disponible. Esto puede ser un salvavidas si tenemos un sistema sin interfaz gráfica, como un servidor remoto o un sistema local con una configuración X rota. El nano, aunque cada vez más popular, todavía no es universal. POSIX, un estándar para la compatibilidad de programas en sistemas Unix, requiere que vi esté presente.
- Vi es ligero y rápido. Para muchas tareas, es más fácil abrir vi que encontrar el editor de texto gráfico en los menús y esperar a que se carguen sus múltiples megabytes. Además, vi está diseñado para la velocidad de escritura. Como veremos, un usuario experto de vi nunca tiene que levantar los dedos del teclado mientras edita.
- No queremos que otros usuarios de Linux y Unix piensen que somos mariquitas. Bien, tal vez dos buenas razones.

## Un poco de historia

La primera versión de vi fue escrita en 1976 por Bill Joy, un estudiante de la Universidad de California, Berkeley, que más tarde cofundó Sun Microsystems. Vi deriva su nombre de la palabra *visual*, porque estaba destinado a permitir la edición en un terminal de video con un cursor móvil. Antes de *los editores visuales*, existían *los editores de línea*, que operaban con una sola línea de texto a la vez. Para especificar un cambio, le decímos a un editor de línea que vaya a una línea en particular y describa qué cambio hacer, como agregar o eliminar texto. Con la llegada de los terminales de vídeo (en lugar de los terminales basados en impresoras como los teletipos), la edición visual se hizo posible. De hecho, vi incorpora un potente editor de líneas llamado ex, y podemos usar comandos de edición de líneas mientras usamos vi.

La mayoría de las distribuciones de Linux no incluyen vi real, sino que se envían con un reemplazo mejorado llamado vim (que es la abreviatura de *Vi IMproved*) escrito por Bram Moolenaar. vim es una mejora sustancial con respecto al Unix VI tradicional y suele estar vinculado simbólicamente (o alias) al nombre vi en los sistemas Linux. En las discusiones que siguen, asumiremos que tenemos un programa llamado vi es realmente vim.

## Arranque y parada vi

Para iniciar vi, simplemente introducimos lo siguiente:

---

```
[me@linuxbox ~]$ vi
```

---

Debería aparecer una pantalla como esta:

---

```
~  
~  
~ VIM - Vi Mejorado  
~  
~ Versión 7.1.138  
~ por Bram Moolenaar et al.  
~ Vim es de código abierto y se puede distribuir libremente  
~  
~ ¡Patrocina el desarrollo de Vim!  
~ escriba :patrocinador de ayuda<Enter> A título informativo  
~  
~ escriba :q<Enter> para salir  
~ escriba :help<Enter> o <F1> para obtener ayuda en línea  
~ escriba :help version7<Enter> Para obtener información sobre la versión  
~  
~ Funcionando en modo compatible con Vi  
~ escriba :set nocp<Enter> para los valores predeterminados de Vim  
~ escriba :help cp-default<Enter> para obtener información sobre esto  
~  
~
```

---

Al igual que hicimos con nano antes, lo primero que hay que aprender es cómo salir. Para salir, introducimos el siguiente comando (tenga en cuenta que el carácter de dos puntos es parte del comando):

---

:q

---

El símbolo del shell debería regresar. Si, por alguna razón, vi no renuncia (generalmente porque hicimos un cambio en un archivo que aún no se ha guardado), podemos decirle a vi que realmente lo decimos en serio agregando un signo de exclamación al comando:

---

:q!

---

**Nota:** Si te "pierdes" en vi, intenta presionar la tecla *ESC* dos veces para encontrar el camino nuevamente.

## Modos de edición

Vamos a arrancar vi de nuevo, esta vez pasándole el nombre de un archivo inexistente. Así es como podemos crear un nuevo archivo con vi:

---

```
[me@linuxbox ~]$ rm -f foo.txt  
[me@linuxbox ~]$ vi foo.txt
```

---

Si todo va bien, deberíamos conseguir una pantalla como esta:

"foo.txt" [Nuevo archivo]

Los caracteres de tilde inicial (~) indican que no existe texto en esa línea. Esto muestra que tenemos un archivo vacío. ¡No escribas nada todavía!

La segunda cosa más importante que hay que aprender sobre vi (después de aprender a salir) es que vi es un *editor modal*. Cuando vi se inicia, comienza en *modo de comando*. En este modo, casi todas las teclas son un comando, por lo que si empezáramos a escribir, vi básicamente se volvería loco y haría un gran lío.

## **Entrar en el modo de inserción**

Para poder añadir algo de texto a nuestro archivo, primero debemos entrar en *el modo de inserción*. Para ello, pulsamos la tecla I (í). Después, deberíamos ver lo siguiente en el botón de la pantalla si vim se está ejecutando en su modo mejorado habitual (esto no aparecerá en el modo compatible con vi):

--INSERTAR--

Ahora podemos introducir algo de texto. Pruebe esto:

**El rápido zorro marrón saltó sobre el perro perezoso.**

Para salir del modo de inserción y volver al modo de comando, pulse la tecla **ESC**.

## *Salvando nuestro trabajo*

Para guardar el cambio que acabamos de hacer en nuestro archivo, debemos ingresar un *comando ex* mientras estamos en modo de comando. Esto se hace fácilmente presionando la tecla `:`. Después de hacer esto, debería aparecer un carácter de dos puntos en la parte inferior de la pantalla:

---

•

---

Para escribir nuestro archivo modificado, seguimos los dos puntos con una **w**, luego ENTER:

---

:w

---

El archivo se escribirá en el disco duro, y deberíamos obtener una confirmación en la parte inferior de la pantalla, así:

---

"foo.txt" [Nuevo] 1L, 46C escrito

---

**Nota:** Si lees la documentación de vim , notarás que (confusamente) el modo de comando se llama modo normal y los comandos ex se llaman modo de comando. Cuidado.

### COM PAT IBILI TY MODE

En la pantalla de inicio de ejemplo que se muestra al principio de esta sección (tomada de Ubuntu 8.04), vemos el texto Ejecutándose en modo compatible con Vi. Esto significa que vim se ejecutará en un modo que está más cerca del comportamiento normal de vi en lugar del comportamiento mejorado de vim. Para los propósitos de este capítulo, querremos ejecutar vim con su comportamiento mejorado. Para ello, tienes un par de opciones:

- Intente ejecutar vim en lugar de vi (si eso funciona, considere agregar el alias `vi=vim'` a su archivo `.bashrc`).
- Utilice este comando para agregar una línea a su archivo de configuración de vim :

```
echo "set nocp" >> ~/.vimrc
```

Diferentes distribuciones de Linux empaquetan vim de diferentes maneras. Algunas distribuciones instalan una versión mínima de vim de forma predeterminada que solo admite un conjunto limitado de características de vim

## Mover el cursor

Mientras está en modo de comando, vi ofrece un gran número de comandos de movimiento, algunos de los cuales comparte con menos. En la tabla 12-1 se muestra un subconjunto.

Tabla 12-1: Teclas de movimiento del cursor

Llave	Mueve el cursor
L o flecha derecha	A la derecha un carácter
H o flecha izquierda	Un carácter a la izquierda
J o flecha hacia abajo	Abajo una línea
K o flecha hacia arriba	Subir una línea

*(continuación)*

Cuadro 12-1 (*continuación* )

Llave	Mueve el cursor
0 (cero)	Al principio de la línea actual
TURNO-6 (^) actual	Al primer carácter que no sea un espacio en blanco de la línea
TURNO-4 (\$)	Hasta el final de la línea actual
W	Al principio de la siguiente palabra o signo de puntuación
TURNO-W (W)	Al principio de la siguiente palabra, ignorando la puntuación De hecho, la mayoría de las
B	Al principio de la palabra o signo de puntuación anterior
TURNO-B (B)	Al principio de la palabra anterior, ignorando los caracteres de puntuación
CTRL-F O BAJAR PÁGINA	Bajar una página
CTRL-B O SUBIR PÁGINA	Subir una página
número-TURNO-G	A la línea <i>número</i> (Por ejemplo, 1G Se mueve a la primera línea del expediente)
TURNO-G (G)	Hasta la última línea del fichero

¿Por qué se utilizan las teclas H, J, K y L para mover el cursor? Porque cuando se escribió vi , no todos los terminales de video tenían teclas de flecha, y los mecanógrafos expertos podían usar las teclas normales del teclado para mover el cursor sin tener que levantar los dedos del teclado.

Muchos comandos en vi pueden tener el prefijo un número, como con el comando G enumerado en la Tabla 12-1. Al prefijar un comando con un número, podemos especificar el número de veces que se debe llevar a cabo un comando. Por ejemplo, el comando 5j hace que vi mueva el cursor cinco líneas hacia abajo.

## Edición básica

La mayor parte de la edición consiste en unas pocas operaciones básicas, como insertar texto, eliminar texto y mover texto cortando y pegando. Vi, por supuesto, apoya todas estas operaciones a su manera. vi también proporciona una forma limitada de deshacer. Si pulsamos la tecla U mientras estamos en modo de comando, vi deshará el último cambio que hayas realizado. Esto será útil a medida que probemos algunos de los comandos básicos de edición.

## Anexar texto

vi tiene varias formas de entrar en el modo de inserción. Ya hemos utilizado el comando **i** para insertar texto.

Volvamos a nuestro archivo *foo.txt* por un momento:

---

El rápido zorro marrón saltó sobre el perro perezoso.

---

Si quisieramos agregar algo de texto al final de esta oración, descubriríamos que el comando **i** no lo hará, porque no podemos mover el cursor más allá del final de la línea. vi proporciona un comando para anexar texto, el llamado sensato-mente un comando. Si movemos el cursor al final de la línea y escribimos **a**, el cursor se moverá más allá del final de la línea y vi entrará en modo de inserción. Esto nos permitirá añadir algo más de texto:

---

El rápido zorro marrón saltó sobre el perro perezoso. **Fue genial.**

---

Recuerde presionar la tecla **ESC** para salir del modo de inserción.

Dado que casi siempre querremos agregar texto al final de una línea, vi ofrece un atajo para movernos al final de la línea actual y comenzar a agregar. Es el comando **A**. Vamos a intentarlo y añadir algunas líneas más a nuestro archivo.

Primero, moveremos el cursor al comienzo de la línea usando el **0** (cero) comando. Ahora escribimos **A** y añadimos las siguientes líneas de texto:

---

El rápido zorro marrón saltó sobre el perro perezoso. **Fue genial.**

Línea 2

Línea 3

Línea 4

Línea 5

---

Nuevamente, presione la tecla **ESC** para salir del modo de inserción.

Como podemos ver, el comando **A** es más útil porque mueve el cursor al final de la línea antes de iniciar el modo de inserción.

## Abrir una línea

Otra forma en que podemos insertar texto es "abriendo" una línea. Esto inserta una línea en blanco entre dos líneas existentes y entra en el modo de inserción. Esto tiene dos variantes, como se muestra en la Tabla 12-2.

Tabla 12-2: Teclas de apertura de línea

Mandar	Abre
0	La línea debajo de la línea actual
0	La línea por encima de la línea actual

Podemos demostrar esto de la siguiente manera: Coloque el cursor en la línea 3 y luego escriba **o**.

---

El rápido zorro marrón saltó sobre el perro perezoso. Fue  
genial. Línea 2  
Línea 3

Línea 4  
Línea 5

---

Se abrió una nueva línea debajo de la tercera línea y entramos en el modo de inserción. Salga del modo de inserción pulsando la tecla **ESC**. Escriba **u** para deshacer nuestro cambio.

Escriba **O** para abrir la línea sobre el cursor:

---

El rápido zorro marrón saltó sobre el perro perezoso. Fue  
genial. Línea 2

Línea 3  
Línea 4  
Línea 5

---

Salga del modo de inserción presionando la tecla **ESC** y deshaga nuestro cambio escribiendo **u**.

### **Eliminación de texto**

Como era de esperar, vi ofrece una variedad de formas de eliminar texto, todas las cuales contienen una de las dos pulsaciones de teclas. Primero, la tecla **X** eliminará un carácter en la ubicación del cursor. **x** puede ir precedido de un número que especifica el número de caracteres que se van a suprimir. La tecla **D** es de propósito más general. Al igual que **x**, puede ir precedido de un número que especifica el número de veces que se realiza la eliminación.

a realizar. Además, **d** siempre va seguido de un comando de movimiento que controla el tamaño de la eliminación. En la tabla 12-3 se enumeran algunos ejemplos.

Coloque el cursor sobre la palabra **It** en la primera línea de nuestro texto. Escriba **x** repetidamente hasta que se elimine el resto de la oración. A continuación, escriba **u** repetidamente hasta que se deshaga la eliminación.

**Nota:** Real vi solo admite un único nivel de deshacer. Vim soporta múltiples niveles.

**Tabla 12-3: Comandos de eliminación de texto**

Mandar	Elimina
x	El personaje actual
3 veces	El carácter actual y los dos caracteres siguientes
Dd	La línea actual



Cuadro 12-3 (*continuación* )

Mandar	Elimina
Dw	Desde la ubicación actual del cursor hasta el principio de la siguiente palabra
D\$	Desde la ubicación actual del cursor hasta el final de la línea actual
d0	Desde la ubicación actual del cursor hasta el principio de la línea
d^	Desde la ubicación actual del cursor hasta el primer carácter de la línea que no sea un espacio en blanco
Dg	Desde la línea actual hasta el final del archivo
d20G	Desde la línea actual hasta la línea 20 del archivo

Intentemos la eliminación de nuevo, esta vez usando el comando d. De nuevo, mueva el cursor a la palabra **lt** y escriba **dW** para eliminar la palabra:

---

El rápido zorro marrón saltó sobre el perro perezoso. fue  
genial. Línea 2  
Línea 3  
Línea 4  
Línea 5

---

Escriba **d\$** para eliminar desde la posición del cursor hasta el final de la línea:

---

El rápido zorro marrón saltó sobre el perro perezoso.  
Línea 2  
Línea 3  
Línea 4  
Línea 5

---

Escriba **dG** para eliminar desde la línea actual hasta el final del archivo:

---

~  
~  
~  
~  
~

---

Escriba **u** tres veces para deshacer las eliminaciones.

### *Cortar, copiar y pegar texto*

El comando d no solo elimina texto, sino que también "corta" texto. Cada vez que usamos el comando d, la eliminación se copia en un búfer de pegado (piense en el portapapeles) que luego podemos recuperar con el comando p para pegar el contenido del botón después del cursor o con el comando P para pegar el contenido antes del cursor.

El comando **y** se usa para "tirar" (copiar) texto de la misma manera que el comando **d** se usa para cortar texto. En la tabla 12-4 se enumeran algunos ejemplos de combinación del comando **y** con varios comandos de movimiento.

**Tabla 12-4: Comandos de tirón**

Mandar	Copias
<b>Yy</b>	La línea actual
<b>5yy</b>	La línea actual y las cuatro líneas siguientes
<b>yW</b>	Desde la ubicación actual del cursor hasta el principio de la siguiente palabra
<b>y\$</b>	Desde la ubicación actual del cursor hasta el final de la línea actual
<b>y0</b>	Desde la ubicación actual del cursor hasta el principio de la línea
<b>y^</b>	Desde la ubicación actual del cursor hasta el primer carácter de la línea que no sea un espacio en blanco
<b>yG</b>	Desde la línea actual hasta el final del archivo
<b>y20G</b>	Desde la línea actual hasta la línea 20 del archivo

Intentemos copiar y pegar. Coloque el cursor en la primera línea del texto y escriba **yy** para copiar la línea actual. A continuación, mueva el cursor a la última línea (**G**) y escriba **p** para pegar la línea copiada debajo de la línea actual:

---

```
El rápido zorro marrón saltó sobre el perro perezoso. Fue
genial. Línea 2
Línea 3
Línea 4
Línea 5
El rápido zorro marrón saltó sobre el perro perezoso. Fue genial.
```

---

Al igual que antes, el comando **u** deshará nuestro cambio. Con el cursor aún colocado en la última línea del archivo, escriba **P** para pegar el texto sobre la línea actual:

---

```
El rápido zorro marrón saltó sobre el perro perezoso. Fue
genial. Línea 2
Línea 3
Línea 4
El rápido zorro marrón saltó sobre el perro perezoso. Fue genial.
Línea 5
```

---

Pruebe algunos de los otros comandos **y** de la Tabla 12-4 y conozca el comportamiento de los comandos **p** y **P**. Cuando haya terminado, devuelva el archivo a su estado original.



## *Líneas de unión*

Vi es bastante estricto en su idea de una línea. Normalmente, no es posible mover el cursor al final de una línea y eliminar el carácter de fin de línea para unir una línea con la que está debajo. Debido a esto, vi proporciona un comando específico, J (que no debe confundirse con j, que es para el movimiento del cursor), para unir líneas.

Si colocamos el cursor en la línea 3 y escribimos el comando J, esto es lo que sucede:

---

```
El rápido zorro marrón saltó sobre el perro perezoso. Fue  
genial. Línea 2  
Línea 3 Línea 4  
Línea 5
```

---

## **Buscar y reemplazar**

Vi tiene la capacidad de mover el cursor a ubicaciones en función de las búsquedas. Puede hacerlo en una sola línea o en un archivo completo. También puede realizar reemplazos de texto con o sin confirmación del usuario.

### *Búsqueda dentro de una línea*

El comando f busca una línea y mueve el cursor a la siguiente instancia de un carácter especificado. Por ejemplo, el comando fa movería el cursor a la siguiente aparición del carácter a dentro de la línea actual. Después de realizar una búsqueda de caracteres dentro de una línea, la búsqueda se puede repetir escribiendo un punto y coma.

### *Búsqueda en todo el archivo*

Para mover el cursor a la siguiente aparición de una palabra o frase, se utiliza la com- m. Esto funciona de la misma manera que en el programa menos que cubrimos en el Capítulo 3. Cuando escriba el comando /, aparecerá una barra diagonal en la parte inferior de la pantalla. A continuación, escriba la palabra o frase que desea buscar, seguida de la tecla ENTER. El cursor se moverá a la siguiente ubicación que contenga la cadena de búsqueda. Una búsqueda se puede repetir utilizando la cadena de búsqueda anterior con el comando n. He aquí un ejemplo:

---

```
El rápido zorro marrón saltó sobre el perro perezoso. Fue  
genial. Línea 2  
Línea 3  
Línea 4  
Línea 5
```

---

Coloque el cursor en la primera línea del archivo. Tipo

---

/Línea

seguido de la tecla `ENTER`. El cursor se moverá a la línea 2. A continuación, escriba `n`,

y el cursor se moverá a la línea 3. Repetir el comando `n` moverá el cursor hacia abajo en el archivo hasta que se quede sin coincidencias. Si bien hasta ahora solo hemos usado palabras y frases para nuestros patrones de búsqueda, `vi` permite el uso de *expresiones regulares*, un método poderoso para expresar patrones de texto complejos. Cubriremos las expresiones regulares con cierto detalle en el Capítulo 19.

### Búsqueda y reemplazo global

`Vi` utiliza un comando `ex` para realizar operaciones de búsqueda y reemplazo (denominadas *sustitución* en `Vi`) en un rango de líneas o en todo el archivo. Para cambiar la palabra *Línea* por *línea* para todo el archivo, ingresaríamos el siguiente comando:

---

`:%s/Línea/línea/g`

---

Vamos a dividir este comando en elementos separados y ver qué hace cada uno (ver Tabla 12-5).

**Tabla 12-5: Ejemplo de sintaxis global de búsqueda y reemplazo**

Artículo	Significado
:	El carácter de dos puntos inicia un comando <code>ex</code> .
%	Especifica el rango de líneas de la operación. % es un atajo que significa desde la primera línea hasta la última línea. Alternativamente, se podría haber especificado el rango 1,5 (porque nuestro archivo tiene cinco líneas), o 1,\$, que significa "desde la línea 1 hasta la última línea del archivo". Si se omite el rango de líneas, la operación solo se realiza en la línea actual.
s	Especifica la operación, en este caso, sustitución (buscar y reemplazar).
/Línea/línea/	El patrón de búsqueda y el texto de reemplazo.
g	Esto significa que <i>global</i> , en el sentido de que la sustitución se realiza en cada instancia de la cadena de búsqueda en cada línea. Si <code>g</code> se omite, solo se reemplaza la primera instancia de la cadena de búsqueda en cada línea.

Después de ejecutar nuestro comando de búsqueda y reemplazo, nuestro archivo se ve así:

---

El rápido zorro marrón saltó sobre el perro perezoso. Fue  
genial. línea 2  
Línea 3  
Línea 4  
Línea 5

---

También podemos especificar un comando de sustitución con confirmación del usuario.

Esto se hace agregando una `c` al final del comando. Por ejemplo:

---

```
:%s/línea/línea/gc
```

---

Este comando cambiará nuestro archivo a su forma anterior; sin embargo, antes de cada sustitución, vi se detiene y nos pide que confirmemos la sustitución con este mensaje:

---

```
reemplácelo por Línea (s/n/a/q/l/^E/^Y)?
```

---

Cada uno de los caracteres dentro de los paréntesis es una respuesta posible, como se muestra en la Tabla 12-6.

**Tabla 12-6: Reemplazar claves de confirmación**

Llave	Acción
y	Realice la sustitución.
n	Omita esta instancia del patrón.
a	Realice la sustitución en esta y en todas las instancias posteriores del patrón.
q o ESC	Deja de sustituir.
l	Realice esta sustitución y, a continuación, salga. Abreviatura de <i>último</i> .
CTRL-E, CTRL-Y	Desplácese hacia abajo y desplácese hacia arriba, respectivamente. Útil para Visualización del contexto de la sustitución propuesta.

## Edición de varios archivos

A menudo es útil editar más de un archivo a la vez. Es posible que tenga que realizar cambios en varios archivos o que tenga que copiar contenido de un archivo a otro. Con vi podemos abrir varios archivos para editarlos especificándolos en la línea de comandos:

nosotros `archivo1 archivo2 archivo3...`

Salgamos de nuestra sesión vi existente y creemos un nuevo archivo para editarlo. Tipo

`:wq` para salir de vi, guardando nuestro texto modificado. A continuación, crearemos un archivo adicional en nuestro directorio de inicio con el que podremos jugar. Crearemos el archivo capturando algunos resultados del comando ls:

---

```
[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt
```

---

Editemos nuestro archivo antiguo y el nuevo con vi:

```
[me@linuxbox ~]$ vi foo.txt ls-output.txt
```

Vi se iniciará, y veremos el primer archivo en la pantalla:

---

El rápido zorro marrón saltó sobre el perro perezoso. Fue  
genial. Línea 2  
Línea 3  
Línea 4  
Línea 5

---

### Cambiar entre archivos

Para cambiar de un archivo a otro, use este comando ex:

---

:n

---

Para volver al archivo anterior, utilice:

---

:N

---

Si bien podemos movernos de un archivo a otro, vi aplica una política que nos impide cambiar de archivo si el archivo actual tiene cambios no guardados. Para obligar a vi a cambiar de archivo y abandonar los cambios, agregue un signo de exclamación (!) al comando.

Además del método de conmutación descrito anteriormente, vim (y algunas versiones de vi) proporciona algunos comandos ex que hacen que varios archivos sean más fáciles de administrar. Podemos ver una lista de los archivos que se están editando con el comando :buffers. Al hacerlo, se mostrará una lista de los archivos en la parte inferior de la pantalla:

---

:Búferes  
1 %a "foo.txt" Línea 1  
2 "ls-output.txt" Línea  
0 Presione ENTER o escriba comando para continuar

---

Para cambiar a otro búfer (archivo), escriba :buffer seguido del número del búfer que desea editar. Por ejemplo, para cambiar del búfer 1, que contiene el archivo *foo.txt*, al búfer 2, que contiene el archivo *ls-output.txt*, escribiríamos lo siguiente:

---

:búfer 2

---

y nuestra pantalla ahora muestra el segundo archivo.

### Abrir archivos adicionales para editarlos

También es posible agregar archivos a nuestra sesión de edición actual. El comando ex :e (abreviatura de *editar*) seguido de un nombre de archivo abrirá un archivo adicional. Terminemos nuestra sesión de edición actual y volvamos a la línea de comandos.

Inicie vi de nuevo con un solo archivo:

---

---

```
[me@linuxbox ~]$ vi foo.txt
```

---

Para agregar nuestro segundo archivo, ingrese:

---

**:e ls-output.txt**

---

y debería aparecer en la pantalla. El primer archivo sigue presente, como podemos comprobar:

---

**:Búferes**

1 # "foo.txt"	Línea 1
2 %a "ls-output.txt"	línea
0 Prensa ENTRAR o tipo mandar Continuar	

---

**Nota:** No se puede cambiar a archivos cargados con el comando `:e` utilizando el comando `:n` o `:N`. Para cambiar de archivo, utilice el comando `:buffer` seguido del número de búfer.

### Copiar contenido de un archivo a otro

A menudo, mientras editamos varios archivos, querremos copiar una parte de un archivo en otro archivo que estemos editando. Esto se hace fácilmente usando los comandos habituales de tirar y pegar que usamos anteriormente. Podemos demostrar lo siguiente. Primero, usando nuestros dos archivos, cambie al búfer 1 (*foo.txt*) ingresando

---

**:búfer 1**

---

Esto debería darnos lo siguiente:

El rápido zorro marrón saltó sobre el perro perezoso. Fue  
genial. Línea 2  
Línea 3  
Línea 4  
Línea 5

---

A continuación, mueva el cursor a la primera línea y escriba **yy** para tirar (copiar) la línea.

Cambie al segundo búfer ingresando

---

**:búfer 2**

---

La pantalla ahora contendrá algunos listados de archivos como este (aquí solo se muestra una parte):

Total 343700	
-rwxr-xr-x 1 raíz raíz	31316 2011-12-05 08:58 [
-rwxr-xr-x 1 raíz raíz	8240 2011-12-09 13:39 411toppm
-rwxr-xr-x 1 raíz raíz	111276 31/01/2012 13:36 a2p
-rwxr-xr-x 1 raíz raíz	25368 06/10/2010 20:16 a52dic
-rwxr-xr-x 1 raíz raíz	11532 04/05/2011 17:43 aafire
-rwxr-xr-x 1 raíz raíz	7292 04/05/2011 17:43 aainfo

---

Mueva el cursor a la primera línea y pegue la línea que copiamos del archivo anterior escribiendo el comando p:

---

```
Total 343700
El rápido zorro marrón saltó sobre el perro perezoso. Fue genial.
-rwxr-xr-x 1 raíz raíz          31316 2011-12-05 08:58 [
-rwxr-xr-x 1 raíz raíz          8240 2011-12-09 13:39 411toppm
-rwxr-xr-x 1 raíz raíz         111276 31/01/2012 13:36 a2p
-rwxr-xr-x 1 raíz raíz         25368 06/10/2010 20:16 a52dic
-rwxr-xr-x 1 raíz raíz        11532 04/05/2011 17:43 aafire
-rwxr-xr-x 1 raíz raíz        7292 04/05/2011 17:43 aainfo
```

---

### *Insertar un archivo completo en otro*

También es posible insertar un archivo completo en uno que estemos editando. Para ver esto en acción, terminemos nuestra sesión vi y comenzemos una nueva con un solo archivo:

---

```
[me@linuxbox -]$ vi ls-output.txt
```

---

Volveremos a ver nuestra lista de archivos:

---

```
Total 343700
-rwxr-xr-x 1 raíz raíz          31316 2011-12-05 08:58 [
-rwxr-xr-x 1 raíz raíz          8240 2011-12-09 13:39 411toppm
-rwxr-xr-x 1 raíz raíz         111276 31/01/2012 13:36 a2p
-rwxr-xr-x 1 raíz raíz         25368 06/10/2010 20:16 a52dic
-rwxr-xr-x 1 raíz raíz        11532 04/05/2011 17:43 aafire
-rwxr-xr-x 1 raíz raíz        7292 04/05/2011 17:43 aainfo
```

---

Mueva el cursor a la tercera línea y, a continuación, introduzca el siguiente comando ex:

---

```
:r foo.txt
```

---

El comando :r (abreviatura de *lectura*) inserta el archivo especificado antes de la posición del cursor. Nuestra pantalla ahora debería verse así:

---

```
Total 343700
-rwxr-xr-x 1 raíz raíz          31316 2011-12-05 08:58 [
-rwxr-xr-x 1 raíz root8240 2011-12-09 13:39 411toppm      El
rápido zorro marrón saltó sobre el perro perezoso. Eso fue
genial. Línea 2
Línea 3
Línea 4
Línea 5
-rwxr-xr-x 1 raíz raíz         111276 31/01/2012 13:36 a2p
-rwxr-xr-x 1 raíz raíz         25368 06/10/2010 20:16 a52dic
-rwxr-xr-x 1 raíz raíz        11532 04/05/2011 17:43 aafire
```

---

-rwxr-xr-x 1 raíz raíz

7292 04/05/2011 17:43 aainfo

## Salvando nuestro trabajo

Como todo lo demás en vi, hay varias formas de guardar nuestros archivos editados. Ya hemos cubierto el comando ex :w, pero hay algunos otros que también pueden resultarnos útiles.

En el modo de comando, al escribir ZZ se guardará el archivo actual y se saldrá de vi. Del mismo modo, el comando ex :wq combinará los comandos :w y :q en uno que guardará el archivo y saldrá.

El comando :w también puede especificar un nombre de archivo opcional. Esto actúa como un comando Guardar como. Por ejemplo, si estuviéramos editando *foo.txt* y quisieramos guardar una versión alternativa llamada *foo1.txt*, ingresaríamos lo siguiente:

---

**:w foo1.txt**

---

**Nota:** Si bien esto guarda el archivo con un nuevo nombre, no cambia el nombre del archivo que está editando. A medida que continúes editando, seguirás editando *foo.txt*, no *foo1.txt*.



# 13

## PERSONALIZACIÓN DE LA SOLICITUD

En este capítulo veremos un detalle aparentemente trivial: nuestro aviso de shell. Este examen revelará algunos de los mecanismos internos del shell y del propio programa emulador terminal.

Como tantas cosas en Linux, el prompt de shell es altamente configurable, y aunque lo hemos dado por sentado, el prompt es un dispositivo realmente útil una vez que aprendemos a controlarlo.

### Anatomía de un prompt

Nuestro mensaje predeterminado tiene un aspecto similar al siguiente:

---

```
[me@linuxbox ~]$
```

---

Observe que contiene nuestro nombre de usuario, nuestro nombre de host y nuestro directorio de trabajo actual, pero ¿cómo llegó a ser así? Resulta muy sencillo. El



prompt se define mediante una variable de entorno denominada PS1 (abreviatura *de prompt string 1*). Podemos ver el contenido de PS1 con el comando echo:

```
[me@linuxbox ~]$ echo $PS1  
[\u@\h \W]\$
```

**Nota:** No te preocupes si tus resultados no son exactamente los mismos que los del ejemplo anterior. Cada distribución de Linux define la cadena de solicitud de manera un poco diferente, algunas de manera bastante exótica.

A partir de los resultados, podemos ver que PS1 contiene algunos de los caracteres que vemos en nuestro mensaje, como los corchetes, el signo @ y el signo de dólar, pero el resto son un misterio. Los más astutos reconocerán estos caracteres especiales *como los* que vimos en la Tabla 7-2. Cuadro 13-1 es una lista parcial de los caracteres que el shell trata especialmente en la cadena de solicitud.

Tabla 13-1: Códigos de escape utilizados en las indicaciones de shell

Secuencia	Valor mostrado
\un	Campana ASCII. Esto hace que la computadora emita un pitido cuando se encuentra.
\d	Fecha actual en formato de día, mes, fecha; por ejemplo, "Lunes 26 de mayo"
\h	Nombre de host del equipo local menos el nombre de dominio final
\H	Lleno Nombre de host
\j	Número de trabajos que se ejecutan en la sesión de shell actual
\l	Nombre del dispositivo terminal actual
\n	Un carácter de nueva línea
\r	Un retorno de carro
\s	Nombre del programa shell
\t	Hora actual en formato de 24 horas, horas:minutos:segundos
\T	Hora actual en formato de 12 horas
\ @	Hora actual en 12 horas, AM/PM formato
\Un	Hora actual en formato de 24 horas, horas:minutos
\u	Nombre de usuario del usuario actual
\w	Número de versión del shell
\W	Números de versión y versión del shell
\w	Nombre del directorio de trabajo actual

Cuadro 13-1 (*continuación* )

Secuencia	Valor mostrado
\W	Última parte del nombre del directorio de trabajo actual
\!	Número de historial del comando actual
\#	Número de comandos introducidos durante esta sesión de shell
\\$	Esto muestra un carácter "\$" a menos que tenga privilegios de superusuario. En ese caso, muestra un "#" en su lugar.
\[	Esto señala el inicio de una serie de uno o más caracteres no imprimibles. Se usa para incrustar caracteres de control no imprimibles que manipulan el emulador de terminal de alguna manera, como mover el cursor o cambiar los colores del texto.
\]	Esto señala el final de una secuencia de caracteres no imprimible.

## Probar algunos diseños de indicaciones alternativos

Con esta lista de caracteres especiales, podemos cambiar el mensaje para ver el efecto. Primero, haremos una copia de seguridad de la cadena existente para poder restaurarla más tarde. Para hacer esto, copiaremos la cadena existente en otra variable de shell que creamos nosotros mismos:

```
[me@linuxbox ~]$ ps1_old="$PS1"
```

Creamos una nueva variable llamada `ps1_old` y le asignamos el valor de `PS1`. Podemos verificar que la cadena ha sido copiada mediante el comando `echo`:

```
[me@linuxbox ~]$ echo $ps1_old  
[me@linuxbox ~]$
```

Podemos restaurar el mensaje original en cualquier momento durante nuestra sesión de terminal simplemente invirtiendo el proceso:

```
[me@linuxbox ~]$ PS1="$ps1_vieja"
```

Ahora que estamos listos para continuar, veamos qué sucede si tenemos una cadena de solicitud vacía:

```
[me@linuxbox ~]$ PS1=
```

Si no asignamos nada a la cadena de solicitud, no obtenemos nada. ¡No hay cadena de aviso en absoluto! El mensaje sigue ahí, pero no muestra nada, tal y como le pedimos. Dado que esto es un poco desconcertante a la vista, lo reemplazaremos con un mensaje mínimo:

**PS1="\\$ "**

---

Eso está mejor. Al menos ahora podemos ver lo que estamos haciendo. Observe el espacio final dentro de las comillas dobles. Esto proporciona el espacio entre el signo de dólar y el cursor cuando se muestra el mensaje.

Agreguemos una campana a nuestro mensaje:

---

```
$ PS1="\a\$ "
```

---

Ahora deberíamos escuchar un pitido cada vez que se muestre el mensaje. Esto podría resultar molesto, pero podría ser útil si necesitáramos una notificación cuando se haya ejecutado un comando de ejecución especialmente prolongada.

A continuación, intentemos hacer un mensaje informativo con información sobre el nombre de host y la hora del día:

---

```
$ PS1="\A \h \$ "
17:33 linuxbox $
```

---

Añadir la hora del día a nuestro prompt será útil si necesitamos hacer un seguimiento de cuándo realizamos determinadas tareas. Finalmente, crearemos un nuevo mensaje que sea similar al original:

---

```
17:37 linuxbox $ PS1=<\u@\h \W>\$ "
<me@linuxbox ~>$
```

---

Pruebe las otras secuencias enumeradas en la Tabla 13-1 y vea si puede encontrar una nueva sugerencia brillante.

## Adición de color

La mayoría de los programas emuladores de terminal responden a ciertas secuencias de caracteres no imprimibles para controlar cosas como los atributos de caracteres (como el color, el texto en negrita y el temido texto parpadeante) y la posición del cursor. Cubriremos la posición del cursor en un momento, pero primero veremos el color.

### TERMINAL CONFUSION

En la antigüedad, cuando los terminales estaban conectados a computadoras remotas, había muchas marcas de terminales que competían entre sí y todas funcionaban de manera diferente. Tenían teclados diferentes, y todos tenían diferentes formas de interpretar la información de control. Unix y los sistemas similares a Unix tienen dos subsistemas bastante complejos (llamados `termcap` y `terminfo`) para lidiar con la babel del control terminal. Si busca en los rincones más profundos de la configuración del emulador de terminal, es posible que encuentre una configuración para el tipo de emulación de terminal.

En un esfuerzo por hacer que los terminales hablen algún tipo de lenguaje común, el American National Standards Institute (ANSI) desarrolló un conjunto estándar de secuencias de caracteres para controlar terminales de video. Los usuarios antiguos de DOS recordarán el *archivo*

El color de los caracteres se controla enviando al emulador de terminal un *código de escape ANSI* incrustado en la secuencia de caracteres que se van a mostrar. El código de control no se "imprime" en la pantalla; más bien es interpretado por el terminal como una instrucción. Como vimos en la Tabla 13-1, las secuencias \[ y \] se utilizan para encapsular caracteres no imprimibles. Un código de escape ANSI comienza con un octal 033 (el código generado por la tecla ESC), seguido de un atributo de carácter opcional, seguido de una instrucción. Por ejemplo, el código para establecer el color del texto en texto negro normal (atributo = 0) es \033[0; 30m.

En la tabla 13-2 se enumeran los colores de texto disponibles. Observe que los colores se dividen en dos grupos, diferenciados por la aplicación del atributo de carácter negrita (1), que crea la apariencia de colores "claros".

**Tabla 13-2: Secuencias de escape utilizadas para establecer los colores del texto**

Secuencia	Mensaje de texto Color
\033[0; 30m	Negro
\033[0; 31m	Rojo
\033[0; 32m	Verde
\033[0; 33m	Marrón
\033[0; 34m	Azul
\033[0; 35m	Morado
\033[0; 36m	Cian
\033[0; 37m	Gris claro
\033[1; 30m	Gris oscuro
\033[1; 31m	Rojo claro
\033[1; 32m	Verde claro
\033[1; 33m	Amarillo
\033[1; 34m	Azul claro
\033[1; 35m	Púrpura claro
\033[1; 36m	Cian claro
\033[1; 37m	Blanco

Intentemos hacer un mensaje rojo (visto aquí como gris). Insertaremos el código de escape al principio:

```
<me@linuxbox ~>$ PS1="\[\033[0; 31m\]\u@\h \W\]$ "
<me@linuxbox ~>$
```

Eso funciona, pero ten en cuenta que todo el texto que escribimos después del mensaje también es rojo. Para solucionar esto, agregaremos otro código de escape al final del símbolo del sistema que le dice al emulador de terminal que vuelva al color anterior:

```
<me@linuxbox ~>$ PS1="\[\033[0; 31m\]\u@\h \W\$\[\033[0m\] "
<me@linuxbox ~>$
```

¡Eso es mejor!

También es posible establecer el color de fondo del texto utilizando los códigos enumerados en la Tabla 13-3. Los colores de fondo no admiten el atributo bold.

**Tabla 13-3: Secuencias de escape utilizadas para establecer el color de fondo**

Secuencia	Color de fondo
\033[0; 40m	Negro
\033[0; 41m	Rojo
\033[0; 42m	Verde
\033[0; 43m	Marrón
\033[0; 44m	Azul
\033[0; 45m	Morado
\033[0; 46m	Cian
\033[0; 47m	Gris claro

Podemos crear un prompt con un fondo rojo aplicando un simple cambio al primer código de escape:

```
<me@linuxbox ~>$ PS1="\[\033[0; 41m\]\u@\h \W\$\[\033[0m\] "
<me@linuxbox ~>$
```

¡Pruebe los códigos de color y vea lo que puede crear!

**Nota:** Además de los atributos de caracteres normales (0) y negrita (1), el texto también puede recibir atributos de puntuación inferior (4), parpadeante (5) e inversa (7). En aras del buen gusto, muchos emuladores de terminales se niegan a respetar el atributo parpadeante.

## Mover el cursor

Los códigos de escape se pueden utilizar para posicionar el cursor. Esto se usa comúnmente

para proporcionar un reloj o algún otro tipo de información en una ubicación diferente de la pantalla, como una esquina superior, cada vez que se dibuja el mensaje.

En la tabla 13-4 se enumeran los códigos de escape que colocan el cursor.

**Tabla 13-4: Secuencias de escape del movimiento del cursor**

Escapar Código	Acción
\033[l;Ch	Mover el cursor a la línea <i>l</i> y columna <i>c</i> .
\033[Na	Mueva el cursor hacia arriba <i>n</i> lineas.
\033[Nb	Mueva el cursor hacia abajo <i>n</i> lineas.
\033[Nc	Mover el cursor hacia adelante <i>n</i> Caracteres.
\033[Nd	Mover el cursor hacia atrás <i>n</i> Caracteres.
\033[2J	Borre la pantalla y mueva el cursor a la esquina superior izquierda (línea 0, columna 0).
\033[K	Borrar desde la posición del cursor hasta el final de la línea actual.
\033[s	Almacene la posición actual del cursor.
\033[U	Recupere la posición del cursor almacenada.

Con estos códigos, construiremos un mensaje que dibuje una barra roja en la parte superior de la pantalla que contenga un reloj (representado en texto amarillo) cada vez que se muestre el mensaje. El código para el símbolo del sistema es esta cadena de aspecto formidable:

---

PS1="\[\033[s\033[0; 0H\033[0; 41m\033[K\033[1; 33m\t\033[0m\033[u\]<\u@h \W>\\$ "

---

La tabla 13-5 echa un vistazo a cada parte de la cadena para ver qué hace.

**Tabla 13-5: Desglose de la cadena de solicitud compleja**

Secuencia	Acción
\[	Comienza una secuencia de caracteres no imprimible. El verdadero propósito de esto es permitir que juerga para calcular correctamente el tamaño de la solicitud visible. Sin esto, las funciones de edición de la línea de comandos colocarán incorrectamente el cursor.
\033[s	Almacene la posición del cursor. Esto es necesario para volver a la ubicación del mensaje después de que la barra y el reloj se hayan dibujado en la parte superior de la pantalla. <i>Tenga en cuenta que algunos emuladores de terminal no respetan este código.</i>
\033[0; 0H	Mueva el cursor a la esquina superior izquierda, que es la línea 0, columna 0.
\033[0; 41m	Establece el color de fondo en rojo.

(continuación)

Cuadro 13-5 (*continuación* )

Secuencia	Acción
\033[K	Desactive desde la ubicación actual del cursor (la esquina superior izquierda) hasta el final de la línea. Dado que el color de fondo ahora es rojo, la línea se borra a ese color, creando nuestra barra. Tenga en cuenta que el desplazamiento hasta el final de la línea no cambia la posición del cursor, que permanece en la esquina superior izquierda.
\033[1; 33m	Establezca el color del texto en amarillo.
\t	Muestra la hora actual. Si bien este es un elemento de "impresión", todavía lo incluimos en la parte no imprimible del mensaje, porque no queremos jerga para incluir el reloj al calcular el tamaño real de la solicitud mostrada.
\033[0m	Desactiva el color. Esto afecta tanto al texto como al fondo.
\033[U	Restaure la posición del cursor guardada anteriormente.
\]	Finalice la secuencia de caracteres no imprimibles.
<\u@\h \W>\\$	Cadena de solicitud.

## Guardar el mensaje

Obviamente, no queremos estar escribiendo ese monstruo todo el tiempo, por lo que querremos almacenar nuestro mensaje en algún lugar. Podemos hacer que el prompt sea permanente agregándolo a nuestro archivo *.bashrc*. Para ello, añade estas dos líneas al archivo:

```
PS1="\[\033[s\033[0; 0H\033[0; 41m\033[K\033[1; 33m\t\033[0m\033[u\]<\u@\h \W>\$ "
exportación PS1
```

## Nota final

Lo creas o no, se puede hacer mucho más con indicaciones que involucran funciones de shell y scripts que no hemos cubierto aquí, pero este es un buen comienzo. No a todo el mundo le importará lo suficiente como para cambiar el mensaje, ya que el mensaje predeterminado suele ser satisfactorio. Pero para aquellos de nosotros a los que nos gusta jugar, el caparazón brinda la oportunidad de muchas horas de diversión trivial.

# **PARTE 3**

**TAREAS COMUNES Y  
HERRAMIENTAS  
ESENCIALES**



# 14

## GESTIÓN DE PAQUETES

Si pasamos algún tiempo en la comunidad de Linux, escuchamos muchas opiniones sobre cuál de las muchas distribuciones de Linux es la "mejor". A menudo, estas discusiones se vuelven realmente tontas, centrándose en cosas como la belleza del fondo de escritorio (¡algunas personas no usarán Ubuntu debido a su combinación de colores predeterminada!) y otros asuntos triviales.

El determinante más importante de la calidad de la distribución es el *sistema de embalaje* y la vitalidad de la comunidad de apoyo a la distribución. A medida que pasamos más tiempo con Linux, vemos que su panorama de software es extremadamente dinámico. Las cosas cambian constantemente. La mayoría de las distribuciones de Linux de primer nivel publican nuevas versiones cada seis meses y muchas actualizaciones de programas individuales todos los días. Para mantenernos al día con esta avalancha de software, necesitamos buenas herramientas para la gestión de paquetes.

*La administración de paquetes* es un método de instalación y mantenimiento de software en el sistema. Hoy en día, la mayoría de las personas pueden satisfacer todas sus necesidades de software instalando *paquetes* de su distribuidor de Linux. Esto contrasta con los primeros días de

Linux, cuando uno tenía que descargar y compilar *el código fuente* para



para instalar software. No es que haya nada malo en compilar el código fuente; de hecho, tener acceso al código fuente es la gran maravilla de Linux. Nos da a nosotros (y a todos los demás) la capacidad de examinar y mejorar el sistema. Es solo que trabajar con un paquete precompilado es más rápido y fácil.

En este capítulo, veremos algunas de las herramientas de línea de comandos utilizadas para la gestión de paquetes. Si bien todas las distribuciones principales proporcionan programas gráficos potentes y sofisticados para mantener el sistema, también es importante aprender sobre los programas de línea de comandos. Pueden realizar muchas tareas que son difíciles (o imposibles) de hacer utilizando sus contrapartes gráficas.

## Sistemas de embalaje

Las diferentes distribuciones utilizan diferentes sistemas de empaquetado y, por regla general, un paquete destinado a una distribución no es compatible con otra distribución. La mayoría de las distribuciones caen en uno de los dos campos de tecnologías de empaquetado: el campo de la .deb Debian y el campo .rpm de Red Hat . Hay algunas excepciones importantes, como Gentoo, Slackware y Foresight, pero la mayoría de los demás utilizan uno de los dos sistemas básicos que se muestran en la Tabla 14-1.

**Tabla 14-1: Principales familias de sistemas de embalaje**

Embalaje Sistema	Distribuciones (listado parcial)
Estilo Debian (.Deb)	Debian, Ubuntu, Xandros, Linspire
Estilo Red Hat (.Rpm)	Fedora, CentOS, Red Hat Enterprise Linux, openSUSE, Mandriva, PCLinuxOS

## Cómo funciona un sistema de paquetes

El método de distribución de software que se encuentra en la industria del software propietario generalmente implica comprar una pieza de medio de instalación, como un "disco de instalación", y luego ejecutar un "asistente de instalación" para instalar una nueva aplicación en el sistema.

Linux no funciona de esa manera. Prácticamente todo el software para un sistema Linux se encuentra en Internet. La mayor parte es proporcionada por el proveedor de distribución en forma de archivos de paquete, y el resto está disponible en forma de código fuente, que se puede instalar manualmente. Hablaremos un poco sobre cómo instalar software compilando el código fuente en el Capítulo 23.

### Archivos de paquete

La unidad básica de software en un sistema de empaquetado es el archivo de

paquete. Un *archivo de paquete* es una colección comprimida de archivos que componen el paquete de software. Un paquete puede constar de numerosos programas y archivos de datos que soportan los programas. Además de los archivos que se van a instalar, el archivo del paquete también incluye metadatos sobre el paquete, como una descripción de texto del

paquete y su contenido. Además, muchos paquetes contienen scripts previos y posteriores a la instalación que realizan tareas de configuración antes y después de la instalación del paquete.

Los archivos de paquetes son creados por una persona conocida como *mantenedor de paquetes*, a menudo (pero no siempre) un empleado del proveedor de distribución. El mantenedor del paquete obtiene el software en forma de código fuente del *proveedor original* (el autor del programa), lo compila y crea los metadatos del paquete y los scripts de instalación necesarios. A menudo, el mantenedor del paquete aplicará modificaciones al código fuente original para mejorar la integración del programa con las otras partes de la distribución de Linux.

## **Repositorys**

Si bien algunos proyectos de software optan por realizar su propio empaquetado y distribución, la mayoría de los paquetes hoy en día son creados por los proveedores de distribución y terceros interesados. Los paquetes se ponen a disposición de los usuarios de una distribución en repositorios centrales, que pueden contener muchos miles de paquetes, cada uno especialmente construido y mantenido para la distribución.

Una distribución puede mantener varios repositorios diferentes para diferentes etapas del ciclo de vida del desarrollo de software. Por ejemplo, normalmente habrá un *repositorio de pruebas*, que contiene paquetes que se acaban de construir y que están destinados a ser utilizados por almas valientes que buscan errores antes de que los paquetes se publiquen para su distribución general. Una distribución a menudo tendrá un *repositorio de desarrollo* donde se guardan los paquetes de trabajo en curso destinados a ser incluidos en la próxima versión principal de la distribución.

Una distribución también puede tener repositorios de terceros relacionados. A menudo son necesarios para suministrar software que, por razones legales, como patentes o problemas de antielusión de la gestión de derechos digitales (DRM), no se puede incluir en la distribución. Quizás el caso más conocido es el del soporte de DVD encriptado, que no es legal en los Estados Unidos. Los repositorios de terceros operan en países donde no se aplican las patentes de software ni las leyes contra la elusión. Estos repositorios suelen ser totalmente independientes de la distribución que soportan, y para utilizarlos hay que Conocerlos e incluirlos manualmente en los archivos de configuración del sistema de gestión de paquetes.

## **Dependencias**

Los programas rara vez son independientes, sino que dependen de la presencia de otros componentes de software para realizar su trabajo. Las actividades comunes, como la entrada/salida, por ejemplo, son manejadas por rutinas compartidas por muchos programas. Estas rutinas se almacenan en lo que se denomina *bibliotecas compartidas*, que proporcionan servicios esenciales a más de un programa. Si un paquete requiere un recurso compartido, como una biblioteca compartida, se dice que tiene una

*dependencia*. Todos los sistemas modernos de gestión de paquetes proporcionan algún método de resolución de *dependencias* para garantizar que cuando se instala un paquete, también se instalan todas sus dependencias.

## *Herramientas de paquete de alto y bajo nivel*

Los sistemas de gestión de paquetes suelen consistir en dos tipos de herramientas: herramientas de bajo nivel que se encargan de tareas como la instalación y eliminación de archivos de paquetes, y herramientas de alto nivel que realizan la búsqueda de metadatos y la resolución de dependencias . En este capítulo, veremos las herramientas suministradas con sistemas de estilo Debian (como Ubuntu y muchos otros) y las utilizadas por los productos recientes de Red Hat. Si bien todas las distribuciones de estilo Red Hat se basan en el mismo programa de bajo nivel (rpm), utilizan diferentes herramientas de alto nivel. Para nuestra discusión, cubriremos el programa de alto nivel yum, utilizado por Fedora, Red Hat Enterprise Linux y CentOS. Otras distribuciones de estilo Red Hat proporcionan herramientas de alto nivel con características comparables (consulte la Tabla 14-2).

**Tabla 14-2: Herramientas del sistema de embalaje**

Distribuciones	Herramientas de bajo nivel	Herramientas de alto nivel
Estilo Debian	dpkg	apt-get, aptitud
Fedora, Red Hat Enterprise Linux, CentOS	Rpm	yum

## **Tareas comunes de administración de paquetes**

Se pueden realizar muchas operaciones con las herramientas de gestión de paquetes de línea de comandos. Vamos a ver los más comunes. Tenga en cuenta que las herramientas de bajo nivel también admiten la creación de archivos de paquetes, una actividad fuera del alcance de este libro.

En la siguiente discusión, el término *package\_name* se refiere al nombre real de un paquete, a diferencia de *package\_file*, que es el nombre del archivo que contiene el paquete.

### *Encontrar un paquete en un repositorio*

Mediante el uso de las herramientas de alto nivel para buscar metadatos en el repositorio, se puede localizar un paquete en función de su nombre o descripción (consulte la Tabla 14-3).

**Tabla 14-3: Comandos de búsqueda de paquetes**

Estilo	Comando(s)
Debian	apt-get actualización apt-cache <i>buscar search_string</i>
Rojo Sombbrero	Búsqueda de yum <i>search_string</i>

Ejemplo: Buscar en un repositorio yum el editor de texto emacs en un sistema Red Hat:

---

Búsqueda de yum emacs

---

### ***Instalación de un paquete desde un repositorio***

Las herramientas de alto nivel permiten descargar un paquete de un repositorio e instalarlo con una resolución de dependencia completa (consulte la Tabla 14-4).

**Tabla 14-4: Comandos de instalación de paquetes**

Estilo	Comando(s)
Debian	<code>apt-get actualización</code> <code>apt-get instalar package_name</code>
Rojo Sombrero	<code>yam instalar package_name</code>

Ejemplo: Instale el editor de texto emacs desde un repositorio apt en un sistema de estilo Debian:

---

`actualización de apt-get; apt-get instalar emacs`

---

### ***Instalación de un paquete desde un archivo de paquete***

Si se ha descargado un fichero de paquete de una fuente distinta de un repositorio, puede instalarse directamente (aunque sin resolución de dependencias) utilizando una herramienta de bajo nivel (véase la Tabla 14-5).

**Tabla 14-5: Comandos de instalación de paquetes de bajo nivel**

Estilo	Mandar
Debian	<code>dpkg --install package_file</code>
Rojo Sombrero	<code>Rpm -Yo package_file</code>

Ejemplo: Si el archivo del paquete `emacs-22.1-7.fc7-i386.rpm` se ha descargado desde un sitio que no es un repositorio, instálelo en un sistema Red Hat de esta manera:

---

`rpm -y emacs-22.1-7.fc7-i386.rpm`

---

**Nota:** Dado que esta técnica utiliza el programa rpm de bajo nivel para realizar la instalación, no se realiza ninguna resolución de dependencias. Si rpm descubre una dependencia faltante, rpm se cerrará con un error.

## **Eliminación de un paquete**

Los paquetes se pueden desinstalar utilizando las herramientas de alto o bajo nivel. Las herramientas de alto nivel se muestran en la Tabla 14-6.

**Tabla 14-6: Comandos de eliminación de paquetes**

Estilo	Mandar
Debian	<code>apt-get eliminar package_name</code>
Rojo Sombrero	Borrado de ymm <code>package_name</code>

Ejemplo: Desinstalar el paquete emacs de un sistema de estilo Debian:

---

```
apt-get eliminar emacs
```

---

## **Actualización de paquetes desde un repositorio**

La tarea más común de administración de paquetes es mantener el sistema actualizado con los paquetes más recientes. Las herramientas de alto nivel pueden realizar esta tarea vital en un solo paso (véase la Tabla 14-7).

**Tabla 14-7: Comandos de actualización de paquetes**

Estilo	Comando(s)
Debian	actualización de apt-get; actualización de apt-get
Rojo Sombrero	ñam actualizar

Ejemplo: Aplique las actualizaciones disponibles a los paquetes instalados en un sistema de estilo Debian:

---

```
actualización de apt-get; actualización de apt-get
```

---

## **Actualización de un paquete a partir de un archivo de paquete**

Si se ha descargado una versión actualizada de un paquete desde una fuente que no es un repositorio, se puede instalar, reemplazando la versión anterior (consulte la Tabla 14-8).

**Tabla 14-8: Comandos de actualización de paquetes de bajo nivel**

Estilo	Mandar
Debian	<code>dpkg --install package_file</code>
Rojo Sombrero	<code>Rpm -U package_file</code>

Ejemplo: Actualice una instalación existente de emacs a la versión contenida en el archivo de paquete *emacs-22.1-7.fc7-i386.rpm* en un sistema Red Hat:

---

```
rpm -U emacs-22.1-7.fc7-i386.rpm
```

---

**Nota:** *dpkg* no tiene una opción específica para actualizar un paquete en lugar de instalar uno, ya que *rpm* lo hace.

### ***Listado de paquetes instalados***

Los comandos que se muestran en la Tabla 14-9 se pueden utilizar para mostrar una lista de todos los paquetes instalados en el sistema.

**Tabla 14-9: Comandos de listado de paquetes**

Estilo	Mandar
Debian	<i>dpkg --lista</i>
Rojo Sombrero	<i>Rpm -Qa</i>

### ***Determinar si un paquete está instalado***

Las herramientas de bajo nivel que se muestran en la Tabla 14-10 se pueden utilizar para mostrar si un paquete especificado está instalado.

**Tabla 14-10: Comandos de estado del paquete**

Estilo	Mandar
Debian	<i>dpkg --status package_name</i>
Rojo Sombrero	<i>rpm -q package_name</i>

Ejemplo: Determine si el paquete emacs está instalado en un sistema de estilo Debian:

---

```
dpkg --status emacs
```

---

### ***Visualización de información sobre un paquete instalado***

Si se conoce el nombre de un paquete instalado, se pueden utilizar los comandos que se muestran en la Tabla 14-11 para mostrar una descripción del paquete.

**Tabla 14-11: Comandos de información de paquetes**

Estilo	Mandar
Debian	<i>apt-cache mostrar package_name</i>



Ejemplo: Vea una descripción del paquete emacs en un sistema de estilo Debian:

---

```
apt-cache mostrar emacs
```

---

### **Encontrar qué paquete instaló un archivo**

Para determinar qué paquete es responsable de la instalación de un archivo particular, se pueden utilizar los comandos que se muestran en la Tabla 14-12.

**Tabla 14-12: Comandos de identificación de archivos de paquete**

Estilo	Mandar
Debian	<code>dpkg --buscar file_name</code>
Rojo Sombbrero	<code>rpm -qf file_name</code>

Ejemplo: Vea qué paquete instaló el archivo `/usr/bin/vim` en un sistema Red Hat:

---

```
rpm -qf /usr/bin/vim
```

---

## **Nota final**

En los capítulos que siguen, exploraremos muchos programas que cubren una amplia gama de áreas de aplicación. Si bien la mayoría de estos programas se instalan comúnmente de forma predeterminada, a veces es posible que necesitemos instalar paquetes adicionales. Con nuestro nuevo conocimiento (y apreciación) de la gestión de paquetes, no deberíamos tener problemas para instalar y gestionar los programas que necesitamos.

### **LA INSTALACIÓN DEL SOFTWARE LINUX MYTH**

Las personas que migran desde otras plataformas a veces son víctimas del mito de que el software es de alguna manera difícil de instalar en Linux y que la variedad de esquemas de empaquetado utilizados por las diferentes distribuciones es un obstáculo. Bueno, lo es un obstáculo, pero solo para los proveedores de software privativo que desean distribuir versiones solo binarias de su software secreto.

El ecosistema de software de Linux se basa en la idea del código fuente abierto. Si un desarrollador de programas publica el código fuente de un producto, es probable que una persona asociada con una distribución empaquete el producto y lo incluya en el repositorio. Este método garantiza que el producto esté bien integrado en la distribución y que el usuario tenga la comodidad de comprar software en un solo lugar, en lugar de tener que buscar en el sitio web de cada producto.

Los controladores de dispositivos se manejan de la misma manera, excepto que en su lugar de ser elementos separados en el repositorio de una distribución, se

en Linux. O el kernel soporta un dispositivo o no lo soporta, y el hardware de Linux soporta muchos dispositivos. Muchos más, de hecho, que Windows. Por supuesto, esto no es un consuelo si el dispositivo en particular que necesita no es compatible. Cuando eso sucede, hay que analizar la causa. La falta de asistencia al conductor suele deberse a una de estas tres cosas:

- **El dispositivo es demasiado nuevo.** Dado que muchos proveedores de hardware no apoyan activamente el desarrollo de Linux, corresponde a un miembro de la comunidad de Linux escribir el código del controlador del kernel. Esto lleva tiempo.
- **El dispositivo es demasiado exótico.** No todas las distribuciones incluyen todos los controladores de dispositivo posibles . Cada distribución construye sus propios kernels, y dado que los kernels son muy configurables (que es lo que hace posible ejecutar Linux en todo, desde relojes de pulsera hasta mainframes), la distribución puede haber pasado por alto un dispositivo en particular. Al localizar y descargar el código fuente del controlador, es posible que usted (sí, usted) compile e instale el controlador usted mismo. Este proceso no es demasiado difícil, pero es bastante complicado. Hablaremos sobre la compilación de software en el Capítulo 23.
- **El proveedor de hardware está ocultando algo.** No ha publicado el código fuente de un controlador de Linux, ni ha publicado la documentación técnica para que otra persona cree uno. Esto significa que el proveedor de hardware está tratando de mantener en secreto las interfaces de programación del dispositivo. Dado que no queremos dispositivos secretos en nuestras computadoras, le sugiero que elimine el



# 15

## MEDIOS DE ALMACENAMIENTO

En capítulos anteriores hemos analizado la manipulación de datos a nivel de archivo. En este capítulo, consideraremos los datos a nivel de dispositivo. Linux tiene capacidades asombrosas para manejar dispositivos de almacenamiento, ya sea almacenamiento físico como discos duros, almacenamiento en red o dispositivos de almacenamiento virtual como RAID (matriz redundante de discos independientes) y LVM (administrador de volúmenes lógicos).

Sin embargo, dado que este no es un libro sobre administración de sistemas, no intentaremos cubrir todo este tema en profundidad. Lo que haremos es presentar algunos de los conceptos y comandos de teclado que se utilizan para administrar dispositivos de almacenamiento.

Para llevar a cabo los ejercicios de este capítulo, utilizaremos una memoria USB, un disco CD-RW (para sistemas equipados con una grabadora de CD-ROM) y un disquete (de nuevo, si el sistema está equipado con ello).

Veremos los siguientes comandos:

- `mount`: monta un sistema de archivos.
- `umount`: desmonta un sistema de archivos.

- fdisk: manipulador de tablas de particiones.
- fsck: comprueba y repara un sistema de archivos.
- fdformat: formatee un disquete.
- mkfs: crea un sistema de archivos.
- dd: escriba datos orientados a bloques directamente en un dispositivo.
- genisoimage (mkisofs): cree un archivo de imagen ISO 9660.
- wodim (cdrecord): escribe datos en medios de almacenamiento óptico.
- md5sum: calcule una suma de comprobación MD5.

## Montaje y desmontaje de dispositivos de almacenamiento

Los avances recientes en el escritorio Linux han hecho que la administración de dispositivos de almacenamiento sea extremadamente fácil para los usuarios de escritorio. En su mayor parte, conectamos un dispositivo a nuestro sistema y simplemente funciona. En los viejos tiempos (digamos, 2004), estas cosas tenían que hacerse manualmente. En los sistemas que no son de escritorio (es decir, servidores), este sigue siendo un procedimiento en gran medida manual, ya que los servidores suelen tener necesidades de almacenamiento extremas y requisitos de configuración complejos.

El primer paso para administrar un dispositivo de almacenamiento es conectar el dispositivo al árbol del sistema de archivos. Este proceso, llamado *montaje*, permite que el dispositivo participe con el sistema operativo. Como recordamos del Capítulo 2, los sistemas operativos tipo Unix, como Linux, mantienen un único árbol de sistemas de archivos con dispositivos conectados en varios puntos. Esto contrasta con otros sistemas operativos como MS-DOS y Windows que mantienen árboles separados para cada dispositivo (por ejemplo, *C:\*, *D:\*, etc.).

Un fichero llamado */etc/fstab* enumera los dispositivos (normalmente particiones de disco duro) que se van a montar en el momento del arranque. Aquí hay un ejemplo de archivo */etc/fstab* de un sistema Fedora 7:

ETIQUETA=/12	/	ext3	Defectos	1 1
LABEL=/inicio	/hogar	ext3	Defectos	1 2
ETIQUETA=/arranq	/bota	ext3	Defectos	1 2
ue				
TMPFS	/dev/shm	TMPFS	Defectos	0 0
devpts	/dev/pts	devpts	gid = 5, modo = 620	0 0
sysfs	/sys	sysfs	Defectos	0 0
Proc	/Proc	Proc	Defectos	0 0
LABEL=SWAP-sda3	intercambio	intercambio		0 0
			Defectos	

La mayoría de los sistemas de archivos enumerados en este archivo de ejemplo son virtuales y no son aplicables a nuestra discusión. Para nuestros propósitos, los interesantes son los tres primeros:

---

ETIQUETA=/12	/	ext3	Defectos	1 1
LABEL=/inicio	/hogar	ext3	Defectos	1 2
ETIQUETA=/arra nque	/bota	ext3	Defectos	1 2

---

Estas son las particiones del disco duro. Cada línea del archivo consta de seis campos, como se muestra en la Tabla 15-1.

**Tabla 15-1: Campos /etc/fstab**

Campo	Contenido	Descripción
1	Dispositivo	Tradicionalmente, este campo contiene el nombre real de un archivo de dispositivo asociado con el dispositivo físico, como <code>/dev/hda1</code> (la primera partición del maestro dispositivo en el primer canal IDE). Pero con las computadoras, que tienen muchos dispositivos que están conectados (como las unidades USB), muchos Linux modernos las distribuciones asocian un dispositivo con una etiqueta de texto en lugar de. Esta etiqueta (que se añade al almacenamiento medio cuando está formateado) es leída por el operador. El sistema de conexión cuando el dispositivo está conectado a la sistema. De esa manera, no importa qué archivo de dispositivo sea asignado al dispositivo físico real, aún puede estar correctamente identificado.
2	Punto de montaje	El directorio donde está conectado el dispositivo al Árbol del sistema de archivos
3	Tipo de sistema de archivos	Linux permite montar muchos tipos de sistemas de archivos. La mayoría de los sistemas de archivos nativos de Linux son ext3, pero muchos otros son compatibles, como FAT16 (msdos), FAT32 (vfat), NTFS (ntfs), CD-ROM (iso9660), etc.
4	Opciones	Los sistemas de archivos se pueden montar con varias opciones. Esos son posibles, por ejemplo, montar sistemas de archivos como solo lectura o para evitar que se publiquen programas ejecutados desde ellos (una característica de seguridad útil para medios extraíbles).
5	Frecuencia	Un solo número que especifica si un archivo y cuándo se va a hacer una copia de seguridad del sistema con el comando dump

---

6	Orden	Un solo número que especifica en qué orden el archivo- Los sistemas deben verificarse con el comando fsck
---	-------	--

---

### ***Visualización de una lista de sistemas de archivos montados***

El comando mount se utiliza para montar sistemas de archivos. Al introducir el comando sin argumentos, se mostrará una lista de los sistemas de archivos montados actualmente:

---

```
[me@linuxbox -]$ montaje  
/dev/sda2 activado / tipo  
ext3 (rw) proc activado /proc  
tipo proc (rw) sysfs activado  
/sys tipo sysfs (rw)
```

```
devpts en /dev/pts tipo devpts (rw,gid=5,mode=620)
/dev/sda5 en /home escriba ext3 (rw)
/dev/sda1 en /boot tipo ext3 (rw)
tmpfs en /dev/shm tipo tmpfs (rw)
Ninguno en /proc/sys/fs/binfmt_misc tipo binfmt_misc
(rw) sunrpc en /var/lib/nfs/rpc_pipefs tipo rpc_pipefs
(rw) fusectl en /sys/fs/fuse/connections tipo fusectl
(rw)
/dev/sdd1 en /media/tipo de disco vfat (rw,nosuid,nodev,noatime,
uhelper=hal,uid=500,utf8,shortname=lower)
twin4:/musicbox en /misc/musicbox tipo nfs4 (rw,addr=192.168.1.4)
```

---

El formato de la lista es *dispositivo* en *mount\_point* tipo *filesystem\_type* (*opciones*). Por ejemplo, la primera línea muestra que el dispositivo */dev/sda2* está montado como el sistema de archivos raíz, es de tipo *ext3* y es legible y escribible (la opción *rw*). Este listado también tiene dos entradas interesantes en la parte inferior. La penúltima entrada muestra una tarjeta de memoria SD de 2 gigabytes en un lector de tarjetas montado en */media/disk*, y la última entrada es una unidad de red montada en */miscelánea/caja de música*.

Para nuestro primer experimento, trabajaremos con un CD-ROM. Primero, echemos un vistazo a un sistema antes de insertar un CD-ROM:

```
[me@linuxbox ~]$ montaje
/dev/mapper/VolGroup00-LogVol00 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs en el tipo /sys sysfs (rw)
devpts en /dev/pts tipo devpts (rw,gid=5,mode=620)
/dev/hda1 en /boot tipo ext3 (rw)
tmpfs en /dev/shm tipo tmpfs (rw)
Ninguno en /proc/sys/fs/binfmt_misc tipo binfmt_misc (rw)
sunrpc en /var/lib/nfs/rpc_pipefs tipo rpc_pipefs (rw)
```

---

Este listado es de un sistema CentOS 5 que utiliza LVM para crear su sistema de archivos raíz. Al igual que muchas distribuciones modernas de Linux, este sistema intentará montar automáticamente el CD-ROM después de la inserción. Despues de insertar el disco, vemos lo siguiente:

```
[me@linuxbox ~]$ montaje
/dev/mapper/VolGroup00-LogVol00 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs en el tipo /sys sysfs (rw)
devpts en /dev/pts tipo devpts (rw,gid=5,mode=620)
/dev/hda1 en /boot tipo ext3 (rw)
tmpfs en /dev/shm tipo tmpfs (rw)
Ninguno en /proc/sys/fs/binfmt_misc tipo binfmt_misc (rw)
sunrpc en /var/lib/nfs/rpc_pipefs tipo rpc_pipefs (rw)
/dev/hdc en /media/live-1.0.10-8 tipo iso9660 (ro,noexec,nosuid,uid=500)
```

---

Vemos el mismo listado que antes, con una entrada adicional. Al final de la lista, vemos que el CD-ROM (que es el dispositivo */dev/hdc* en este sistema) ha sido montado en */media/live-1.0.10-8* y es de tipo *iso9660* (un CD-ROM). A los efectos de nuestro experimento, nos interesa el nombre del dispositivo. Cuando realice este experimento usted mismo, lo más probable es que el nombre del dispositivo sea diferente.

**Advertencia:** En los ejemplos que siguen, es de vital importancia que preste mucha atención a los nombres reales de los dispositivos que se utilizan en su sistema y que no utilice los nombres utilizados en este texto.

Además, tenga en cuenta que los CD de audio no son lo mismo que los CD-ROM. Los CD de audio no contienen sistemas de archivos y, por lo tanto, no se pueden montar en el sentido habitual.

Ahora que tenemos el nombre del dispositivo de la unidad de CD-ROM, desmontemos el disco y volvamos a montarlo en otra ubicación en el árbol del sistema de archivos. Para ello, nos convertimos en el superusuario (usando el comando apropiado para nuestro sistema) y desmontamos el disco con el comando umount (fíjate en la ortografía):

---

```
[me@linuxbox ~]$ su -  
Contraseña:  
[root@linuxbox ~]# umount /dev/hdc
```

---

El siguiente paso es crear un nuevo punto de montaje para el disco. Un *punto de montaje* es simplemente un directorio en algún lugar del árbol del sistema de archivos. No tiene nada de especial . Ni siquiera tiene que ser un directorio vacío, aunque si monta un dispositivo en un directorio no vacío, no podrá ver el contenido anterior del directorio hasta que desmonte el dispositivo. Para nuestros propósitos, crearemos un nuevo directorio:

---

```
[root@linuxbox ~]# mkdir /mnt/cdrom
```

---

Finalmente, montamos el CD-ROM en el nuevo punto de montaje. La opción -t se utiliza para especificar el tipo de sistema de archivos:

---

```
[root@linuxbox ~]# mount -t iso9660 /dev/hdc /mnt/cdrom
```

---

A continuación, podemos examinar el contenido del CD-ROM a través del nuevo punto de montaje:

---

```
[root@linuxbox ~]# cd /mnt/cdrom  
[root@linuxbox cdrom]# ls
```

---

Fíjate en lo que ocurre cuando intentamos desmontar el CD-ROM:

---

```
[root@linuxbox cdrom]# umount /dev/hdc  
umount: /mnt/cdrom: el dispositivo está ocupado
```

---

¿A qué se debe esto? No podemos desmontar un dispositivo si el dispositivo está siendo utilizado por alguien o algún proceso. En este caso, cambiamos nuestro directorio de trabajo al punto de montaje del CD-ROM, lo que hace que el dispositivo esté ocupado. Podemos remediar fácilmente el problema cambiando el directorio de trabajo a algo que no sea el punto de montaje:

---

```
[root@linuxbox cdrom]# cd  
[root@linuxbox ~]# umount /dev/hdc
```

---

Ahora el dispositivo se desmonta con éxito.

## ¿POR QUÉ ES IMPORTANTE LA UNIDAD DE BÚFER?

Si observa la salida del comando `free`, que muestra estadísticas sobre el uso de la memoria, verá una estadística llamada *búferes*. Los sistemas informáticos están diseñados para ir lo más rápido posible. Uno de los impedimentos para la velocidad del sistema

es dispositivos lentos. Las impresoras son un buen ejemplo. Incluso la impresora más rápida es extremadamente lenta para los estándares informáticos. De hecho, una computadora sería muy lenta si tuviera que detenerse y esperar a que una impresora termine de imprimir una página. En los primeros días de las PC (antes de la multitarea), esto era un problema real. Si estaba trabajando en una hoja de cálculo o en un documento de texto, el ordenador se detendría y dejaría de estar disponible cada vez que imprimiera. La computadora enviaba los datos a la impresora tan rápido como la impresora podía aceptarlos, pero era muy lento porque las impresoras no imprimen muy rápido. Este problema se resolvió con la llegada del *búfer de impresora*, un dispositivo que contenía algo de memoria RAM, que se encontraba entre la computadora y la impresora. Con el búfer de impresora en su lugar, la computadora enviaría la salida de la impresora al búfer y se almacenaría rápidamente en la RAM rápida para que la computadora pudiera volver a funcionar sin esperar. Mientras tanto, el búfer de la impresora transportaría lentamente los datos a la impresora desde la memoria del búfer a la velocidad a la que la impresora podría aceptarlos.

Esta idea de almacenamiento en búfer se usa ampliamente en las computadoras para hacerlas más rápidas. No permita que la necesidad de leer o escribir datos ocasionalmente hacia o desde dispositivos lentos impida la velocidad del sistema. Los sistemas operativos almacenan los datos que se han leído y se van a escribir en los dispositivos de almacenamiento en la memoria durante el mayor tiempo posible antes de tener que interactuar con el dispositivo más lento. En un sistema Linux, por ejemplo, notará que el sistema parece llenar la memoria cuanto más tiempo se usa. Esto no significa que Linux esté "utilizando" toda la memoria, sino que está aprovechando toda la memoria disponible para hacer todo el esfuerzo que pueda.

Este almacenamiento en búfer permite que la escritura en los dispositivos de almacenamiento se realice muy rápidamente, ya que la escritura en el dispositivo físico se difiere para un momento futuro. Mientras tanto, los datos destinados al dispositivo se acumulan en la memoria. De vez en cuando, el sistema operativo escribirá estos datos en el dispositivo físico.

Desmontar un dispositivo implica escribir todos los datos restantes en el dispositivo para que se pueda quitar de forma segura. Si el dispositivo se retira sin desmontarlo primero, existe la posibilidad de que no se hayan transferido todos los datos destinados al dispositivo. En algunos casos, estos datos pueden incluir actualizaciones vitales del directorio, lo que conducirá a *la corrupción del sistema de archivos*, una de las peores cosas que pueden suceder en una computadora.

### Determinación de los nombres de los dispositivos

A veces es difícil determinar el alcance de un dispositivo. En los viejos tiempos, no era muy difícil. Un dispositivo siempre estaba en el mismo lugar

y no cambiaba. A los sistemas tipo Unix les gusta así. Cuando se desarrolló Unix, "cambiar una unidad de disco" implicaba usar una carretilla elevadora para eliminar un lavado

dispositivo del tamaño de una máquina de la sala de ordenadores. En los últimos años, la configuración típica del hardware de escritorio se ha vuelto bastante dinámica, y Linux ha evolucionado para ser más flexible que sus antepasados.

En los ejemplos anteriores, aprovechamos la capacidad del escritorio Linux moderno para montar "automáticamente" el dispositivo y luego determinar el nombre después del hecho. Pero, ¿qué pasa si estamos administrando un servidor o algún otro entorno donde esto no ocurre? ¿Cómo podemos averiguarlo?

Primero, veamos cómo el sistema nombra los dispositivos. Si listamos el contenido del directorio `/dev` (donde residen todos los dispositivos), podemos ver que hay montones y montones de dispositivos:

```
[me@linuxbox ~]$ ls /dev
```

El contenido de esta lista revela algunos patrones de nomenclatura de dispositivos.

En la Tabla 15-2 se enumeran algunos.

Tabla 15-2: Nombres de dispositivos de almacenamiento de Linux

Patrón	Dispositivo
<code>/dev/fd*</code>	Unidades de disquete
<code>/dev/hd*</code>	Discos IDE (PATA) en sistemas antiguos. Las placas base típicas contienen dos conectores IDE, o <i>Canales</i> , cada uno con un cable con dos puntos de fijación para accionamientos. La primera unidad del cable se denomina <i>maestro</i> dispositivo y el segundo se llama el <i>esclavo</i> dispositivo. Los nombres de los dispositivos se ordenan de tal manera que <code>/dev/hda</code> se refiere al dispositivo maestro en el primer canal, <code>/dev/hdb</code> es el dispositivo esclavo en el primer canal; <code>/dev/hdc</code> , el dispositivo maestro en el segundo canal, y así sucesivamente. Un dígito final indica el número de partición en el dispositivo. Por ejemplo, <code>/dev/hda1</code> hace referencia a la primera partición en el primer disco duro del sistema, mientras que <code>/dev/hda</code> se refiere a toda la unidad.
<code>/dev/lp*</code>	Impresoras
<code>/dev/sd*</code>	Discos SCSI. En los sistemas Linux recientes, el kernel trata todos los dispositivos similares a discos (incluidos los discos duros PATA/SATA, las unidades flash y los dispositivos de almacenamiento masivo USB, como reproductores de música portátiles y cámaras digitales) como discos SCSI. El resto del sistema de nomenclatura es similar al antiguo <code>/dev/hd*</code> esquema de nomenclatura descrito anteriormente.
<code>/dev/sr*</code>	Unidades ópticas (lectores y grabadoras de CD/DVD)

---

Además, a menudo vemos enlaces simbólicos como */dev/cdrom*, */dev/dvd* y */dev/floppy*, que apuntan a los archivos reales del dispositivo, proporcionados para su comodidad.

Si está trabajando en un sistema que no monta automáticamente dispositivos extraíbles, puede utilizar la siguiente técnica para determinar cómo

El dispositivo extraíble se nombra cuando está conectado. Primero, inicie una vista en tiempo real del *archivo /var/log/messages* (es posible que necesite privilegios de superusuario para esto):

---

```
[me@linuxbox ~]$ sudo tail -f /var/log/messages
```

---

Se mostrarán las últimas líneas del archivo y luego se pausarán. A continuación, conecte el dispositivo extraíble. En este ejemplo, usaremos una unidad flash de 16 MB. Casi de inmediato, el kernel notará el dispositivo y lo probará:

```
Jul 23 10:07:53 linuxbox kernel: usb 3-2: nuevo dispositivo USB de alta velocidad  
usando uhci_h cd y dirección 2  
Jul 23 10:07:53 linuxbox kernel: usb 3-2: configuración #1 elegida de 1 elección  
Jul 23 10:07:53 linuxbox kernel: scsi3 : emulación SCSI para desarrolladores de  
almacenamiento masivo USB  
Jul 23 10:07:58 linuxbox kernel: escaneo scsi: resultado de la investigación  
demasiado corto (5), usando 36  
Jul 23 10:07:58 linuxbox kernel: scsi 3:0:0:0: Acceso directo Easy Disk 1.00 PQ:  
0 ANSI: 2  
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] 31263 512-byte hardware secto  
rs (16 MB)  
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Write Protect is off  
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Suponiendo caché de la  
unidad: escribir t hrough  
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] 31263 512-byte hardware secto  
rs (16 MB)  
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Write Protect is off  
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Suponiendo caché de la  
unidad: escribir t hrough  
Jul 23 10:07:59 linuxbox kernel: sdb: sdb1  
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Disco extraíble SCSI adjunto  
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: Adjunto scsi genérico sg3 tipo 0
```

---

Después de que la pantalla se detenga de nuevo, presione CTRL-C para recuperar el mensaje. Las partes interesantes de la salida son las referencias repetidas a [sdb], que coincide con nuestras expectativas de un nombre de dispositivo de disco SCSI. Sabiendo esto, dos líneas se vuelven particularmente esclarecedoras:

---

```
Jul 23 10:07:59 linuxbox kernel: sdb: sdb1  
Jul 23 10:07:59 linuxbox kernel: sd 3:0:0:0: [sdb] Disco extraíble SCSI conectado
```

---

Esto nos indica que el nombre del dispositivo es */dev/sdb* para todo el dispositivo y */dev/sdb1* para la primera partición del dispositivo. Como hemos visto, trabajar con Linux significa un montón de trabajo detectivesco interesante.

**Nota:** El uso de la técnica *tail -f /var/log/messages* es una excelente manera de ver lo que está haciendo el sistema casi en tiempo real.

Con el nombre de nuestro dispositivo en la mano, ahora podemos montar la unidad flash:

---

```
[me@linuxbox ~]$ sudo mkdir /mnt/flash  
[me@linuxbox ~]$ sudo mount /dev/sdb1 /mnt/flash
```

---

[me@linuxbox ~]\$ df

Sistema de archivos	1K-bloques	Usado	Uso Disponible	% Montado en
/dev/sda2	15115452	5186944	9775164	35% /
/dev/sda5	59631908	31777376	24776480	57% /hogar
/dev/sda1	147764	17277	122858	13% /arranque
TMPFS	776808	0	776808	0% /dev/shm
/dev/sdb1	15560	0	15560	0% /mnt/flash

El nombre del dispositivo seguirá siendo el mismo siempre que permanezca conectado físicamente al equipo y el equipo no se reinicie.

## Creación de nuevos sistemas de archivos

Digamos que queremos reformatear la unidad flash con un sistema de archivos nativo de Linux, en lugar del sistema FAT32 que tiene ahora. Esto implica dos pasos: primero, (opcionalmente) crear un nuevo diseño de partición si el existente no es de nuestro agrado, y segundo, crear un nuevo sistema de archivos vacío en la unidad.

**Advertencia:** *En el siguiente ejercicio, vamos a formatear una unidad flash. ¡Use una unidad que no contenga nada que le importe porque se borrará! Nuevamente, asegúrese de especificar el nombre de dispositivo correcto para su sistema, no el que se muestra en el texto. Si no se presta atención a esta advertencia, se podría formatear (es decir, borrar) la unidad incorrecta.*

### Manipulación de particiones con fdisk

El programa fdisk nos permite interactuar directamente con dispositivos similares a discos (como unidades de disco duro y unidades flash) a un nivel muy bajo. Con esta herramienta podemos editar, eliminar y crear particiones en el dispositivo. Para trabajar con nuestra unidad flash, primero debemos desmontarla (si es necesario) y luego invocar el programa fdisk de la siguiente manera:

---

```
[me@linuxbox ~]$ sudo umount /dev/sdb1
[me@linuxbox ~]$ sudo fdisk /dev/sdb
```

---

Tenga en cuenta que debemos especificar el dispositivo en términos de todo el dispositivo, no por el número de partición. Después de que se inicie el programa, veremos el siguiente mensaje:

---

Comando (m de ayuda):

---

Al introducir una m se mostrará el menú del programa:

---

Acción de comando

- a    Alternar una marca de arranque
- b    Editar BSD DiskLabel
- c    Alternar la marca de compatibilidad de DOS
- d    Eliminar una partición
- l    Enumerar los tipos de partición conocidos
- m    Imprimir este menú
- n    Agregar una nueva partición

- o** crear una nueva tabla de particiones de DOS vacía
- p** Imprimir la tabla de particiones
- q** Salir sin guardar los cambios
- s** crear una nueva etiqueta de disco Sun vacía
- t** Cambiar el ID del sistema de una partición
- u** Cambiar unidades de visualización/entrada
- v** Verificación de la tabla de particiones
- w** Escribir tabla en el disco y salir
- x** Funcionalidad adicional (solo expertos)

Comando (m de ayuda):

---

Lo primero que queremos hacer es examinar el diseño de la partición existente.

Para ello, introducimos **p** para imprimir la tabla de particiones del dispositivo:

---

Comando (m de ayuda): **p**

Disco /dev/sdb: 16 MB, 16006656 bytes  
1 cabezas, 31 sectores/pista, 1008 cilindros  
Unidades = cilindros de 31 \* 512 = 15872  
bytes

Arranque del dispositivo	Empeza r	Fin	Bloques	Sistema de identificación
/dev/sdb1	2	1008	15608+	b W95 FAT32

---

En este ejemplo, vemos un dispositivo de 16 MB con una sola partición (1) que usa 1006 de los 1008 cilindros disponibles en el dispositivo. La partición se identifica como una partición FAT32 de Windows 95. Algunos programas usarán este identificador para limitar los tipos de operaciones que se pueden realizar en el disco, pero la mayoría de las veces cambiar el identificador no es crítico. Sin embargo, en aras de la demostración, lo cambiaremos para indicar una partición de Linux. Para hacer esto, primero debemos averiguar qué ID se utiliza para identificar una parte de Linux. En el listado anterior, vemos que el ID **b** se utiliza para especificar la partición existente. Para ver una lista de los tipos de partición disponibles, volvemos al menú del programa. Allí podemos ver la siguiente elección:

---

**l lista Tipos de partición conocidos**

---

Si introducimos **l** en el prompt, se muestra una gran lista de tipos posibles. Entre ellos vemos **b** para nuestro tipo de partición existente y **83** para Linux.

Volviendo al menú, vemos esta opción para cambiar un ID de partición:

---

**t Cambiar el ID del sistema de una partición**

---

Introducimos **t** en el símbolo del sistema e introducimos el nuevo ID:

---

Comando (m de ayuda): **t**  
Partición seleccionada 1

Código hexadecimal (escriba L para listar códigos): **83**  
Se ha cambiado el tipo de sistema de la partición 1 a la 83 (Linux)

---

Esto completa todos los cambios que necesitamos hacer. Hasta este punto, el dispositivo no se ha tocado (todos los cambios se han almacenado en la memoria, no en el dispositivo físico), por lo que escribiremos la tabla de particiones modificada en el dispositivo y saldremos.

Para ello, introducimos w en el prompt:

---

Comando (m de ayuda): w  
¡La tabla de particiones ha sido alterada!

Llamando a ioctl() para volver a leer la tabla de particiones.

ADVERTENCIA: Si ha creado o modificado alguna partición de DOS 6.x, consulte la página de manual de fdisk para obtener información adicional.

Sincronización de discos.  
[me@linuxbox ~]\$

---

Si hubiéramos decidido dejar el dispositivo sin alterar, podríamos haber ingresado q en el prompt, lo que habría salido del programa sin escribir los cambios. Podemos ignorar con seguridad el mensaje de advertencia que suena ominoso.

### *Creación de un nuevo sistema de archivos con mkfs*

Una vez terminada la edición de nuestra partición (por ligera que haya sido), es hora de crear un nuevo sistema de archivos en nuestra unidad flash. Para hacer esto, usaremos mkfs (abreviatura *de make filesystem*), que puede crear sistemas de archivos en una variedad de formatos. Para crear un sistema de archivos ext3 en el dispositivo, usamos la opción -t para especificar el tipo de sistema ext3, seguido del nombre del dispositivo que contiene la partición que deseamos formatear:

---

```
[me@linuxbox ~]$ sudo mkfs -t ext3 /dev/sdb1
mke2fs 1.40.2 (12-Jul-2012)
Etiqueta del
sistema de
archivos = Tipo de
sistema
operativo: Linux
Tamaño de bloque=1024
(log=0) Tamaño del
fragmento=1024 (log=0)
3904 inodos, 15608 bloques
780 bloques (5,00%) reservados para el
superusuario Primer bloque de datos=1
Bloques máximos del sistema de
archivos = 15990784 2 grupos de
bloques
8192 bloques por grupo, 8192 fragmentos por grupo
1952 inodos por grupo
Copias de seguridad de superbloques
almacenadas en bloques: 8193
```

Escritura de tablas de inodos: hecho

Creación de diario (1024 bloques): hecho  
Escritura de superbloques e información contable del sistema de archivos: hecho

Este sistema de archivos se comprobará automáticamente  
cada 34 montajes o 180 días, lo que ocurra primero. Use  
tune2fs -c o -i para anular. [me@linuxbox -]\$\n

---

El programa mostrará una gran cantidad de información cuando ext3 sea el tipo de sistema de archivos elegido . Para volver a formatear el dispositivo a su sistema de archivos FAT32 original, especifique vfat como el tipo de sistema de archivos:

---

```
[me@linuxbox ~]$ sudo mkfs -t vfat /dev/sdb1
```

---

Este proceso de partición y formateo se puede utilizar en cualquier momento en que se agreguen dispositivos de almacenamiento adicionales al sistema. Si bien trabajamos con una pequeña unidad flash, el mismo proceso se puede aplicar a los discos duros internos y otros dispositivos de almacenamiento extraíbles como los discos duros USB.

## Prueba y reparación de sistemas de archivos

En nuestra discusión anterior sobre el *archivo /etc/fstab*, vimos algunos dígitos misteriosos al final de cada línea. Cada vez que el sistema arranca, comprueba rutinariamente la integridad de los sistemas de archivos antes de montarlos. Esto lo hace el programa fsck (abreviatura de *verificación del sistema de archivos*). El último número de cada *entrada fstab* especifica el orden en el que se van a comprobar los dispositivos. En nuestro ejemplo anterior, vemos que primero se verifica el sistema de archivos raíz, seguido de los sistemas de archivos de *inicio* y *arranque*. Los dispositivos con un cero como último dígito no se comprueban de forma rutinaria.

Además de verificar la integridad de los sistemas de archivos, fsck también puede reparar sistemas de archivos corruptos con diversos grados de éxito, dependiendo de la cantidad de daño. En los sistemas de archivos tipo Unix, las partes recuperadas de los archivos se colocan en el *directorio lost+found*, ubicado en la raíz de cada sistema de archivos.

Para comprobar nuestra unidad flash (que debe desmontarse primero), podríamos hacer lo siguiente:

---

```
[me@linuxbox ~]$ sudo fsck /dev/sdb1
fsck 1.40.8 (13-Mar-2012)
e2fsck 1.40.8 (13-Mar-2012)
/dev/sdb1: limpio, archivos 11/3904 , bloques 1661/15608
```

---

En mi experiencia, la corrupción del sistema de archivos es bastante rara a menos que haya un problema de hardware, como una unidad de disco defectuosa. En la mayoría de los sistemas, la corrupción del sistema de archivos detectada en el momento del arranque hará que el sistema se detenga y le indicará que ejecute fsck antes de continuar.

### ¿QUÉ ES LO QUE FS CK?

En la cultura Unix, fsck se usa a menudo en lugar de una palabra popular con la que comparte tres letras. Esto es especialmente apropiado, dado que probablemente pronunciará la palabra antes mencionada si se encuentra en una situación en la que se ve obligado a ejecutar fsck.

## Formateo de disquetes

Para aquellos de nosotros que todavía usamos computadoras lo suficientemente viejas como para estar equipadas con unidades de disquete, también podemos administrar esos dispositivos. La preparación de un disquete en blanco para su uso es un proceso de dos pasos. Primero, realizamos un formateo de bajo nivel en el disco y luego creamos un sistema de archivos. Para llevar a cabo el formateo, usamos el programa `dformat` especificando el nombre del dispositivo de disquete (normalmente `/dev/fd0`):

---

```
[me@linuxbox ~]$ sudo fdformat /dev/fd0
Doble cara, 80 pistas, 18 segundos/pista. Capacidad total 1440
kB. Formateo ... hecho
Verificar... hecho
```

---

A continuación, aplicamos un sistema de archivos FAT al disco con `mkfs`: [me@linuxbox ~]\$ `sudo mkfs -t msdos /dev/fd0`

---

Tenga en cuenta que usamos el tipo de sistema de archivos `msdos` para obtener las tablas de asignación de archivos de estilo más antiguo (y más pequeño). Una vez que se prepara un disco, se puede montar como otros dispositivos.

## Traslado de datos directamente hacia y desde dispositivos

Si bien generalmente pensamos que los datos en nuestras computadoras están organizados en archivos, también es posible pensar en los datos en forma "sin procesar". Si nos fijamos en una unidad de disco, por ejemplo, vemos que consta de un gran número de "bloques" de datos que el sistema operativo ve como directorios y archivos. Si pudiéramos tratar una unidad de disco como una simple colección de bloques de datos, podríamos realizar tareas útiles, como clonar dispositivos.

El programa `dd` realiza esta tarea. Copia bloques de datos de un lugar a otro. Utiliza una sintaxis única (por razones históricas) y se suele utilizar de la siguiente manera:

---

```
dd if=input_file of=output_file [bs=block_size [count=bloques]]
```

---

Digamos que tenemos dos memorias USB del mismo tamaño y queremos para copiar exactamente la primera unidad a la segunda. Si conectamos ambas unidades a la computadora y se asignan a los dispositivos `/dev/sdb` y `/dev/sdc` respectivamente, podríamos copiar todo lo que hay en la primera unidad a la segunda unidad con lo siguiente:

---

```
dd if=/dev/sdb of=/dev/sdc
```

---

Alternativamente, si solo se conectara el primer dispositivo a la computadora, podríamos copiar su contenido a un archivo ordinario para su

posterior restauración o copia:

---

**dd if=/dev/sdb of=flash\_drive.img**

---

**Advertencia:** El comando dd es muy potente. Aunque su nombre deriva de la definición de datos, a veces se le llama destruir disco porque los usuarios a menudo escriben mal las especificaciones if o of. ¡Siempre verifique dos veces sus especificaciones de entrada y salida antes de presionar ENTER!

## Creación de imágenes de CD-ROM

La grabación de un CD-ROM grabable (ya sea un CD-R o un CD-RW) consta de dos pasos: primero, construir un *archivo de imagen ISO* que sea la imagen exacta del sistema de archivos del CD-ROM, y segundo, escribir el archivo de imagen en el medio CD-ROM.

### Creación de una copia de imagen de un CD-ROM

Si queremos hacer una imagen ISO de un CD-ROM existente, podemos usar dd para leer todos los bloques de datos del CD-ROM y copiarlos en un archivo local. Digamos que teníamos un CD de Ubuntu y queríamos hacer un archivo ISO que luego pudiéramos usar para hacer más copias. Después de insertar el CD y determinar el nombre del dispositivo (asumiremos /dev/cdrom), podemos hacer el archivo ISO de la siguiente manera:

---

```
dd if=/dev/cdrom of=ubuntu.iso
```

---

Esta técnica también funciona para los DVD de datos, pero no funcionará para los CD de audio, ya que no utilizan un sistema de archivos para el almacenamiento. En el caso de los CD de audio, consulte el comando cdrdao.

### UN PROGRAMA CON CUALQUIER OTRO NOMBRE...

Si miras tutoriales en línea para crear y grabar medios ópticos como CD-ROM y DVD, con frecuencia encontrarás dos programas llamados mkisofs y cdrecord. Estos programas formaban parte de un paquete popular llamado cdrtools escrito por Jörg Schilling. En el verano de 2006, el Sr. Schilling hizo un cambio de licencia en una parte del paquete cdrtools que, en opinión de muchos en la comunidad Linux, creaba una incompatibilidad de licencia con la GNU GPL. Como resultado, se inició una *bifurcación* del proyecto cdrtools, que ahora incluye programas de reemplazo para cdrecord y mkisofs llamados wodim y genisoimage, respectivamente.

### Creación de una imagen a partir de una colección de archivos

Para crear un archivo de imagen ISO que contenga el contenido de un directorio, utilizamos el programa enisoimage. Para ello, primero creamos un directorio que contiene todos los archivos que deseamos incluir en la imagen y luego ejecutamos el comando genisoimage para crear el archivo de imagen. Por ejemplo, si hubiéramos creado un directorio llamado ~/cd-rom-files y lo hubiéramos llenado con archivos para nuestro CD-ROM, podríamos crear un archivo de imagen llamado cd-rom.iso con el siguiente comando:

---

```
genisoimage -o cd-rom.iso -R -J ~/ficheros-cd-rom
```

---

La opción `-R` agrega metadatos para las *extensiones de Rock Ridge*, que permiten el uso de nombres de archivo largos y permisos de archivo de estilo POSIX. Del mismo modo, la opción `-J` habilita las *extensiones Joliet*, que permiten nombres de archivo largos en Windows.

## Escritura de imágenes de CD-ROM

Una vez que tenemos un archivo de imagen, podemos grabarlo en nuestro medio óptico. La mayoría de los comandos que discutimos a continuación se pueden aplicar tanto a medios de CD-ROM como de DVD grabables.

### *Montaje de una imagen ISO directamente*

Existe un truco que podemos utilizar para montar una imagen ISO mientras aún está en nuestro disco duro y tratarla como si ya estuviera en un soporte óptico. Al agregar la opción de bucle `-o` para montar (junto con el tipo de sistema de archivos `-t iso9660` requerido), podemos montar el archivo de imagen como si fuera un dispositivo y adjuntarlo al árbol del sistema de archivos:

---

```
mkdir /mnt/iso_image  
mount -t iso9660 -o loop image.iso /mnt/iso_image
```

---

En el ejemplo anterior, creamos un punto de montaje denominado `/mnt/iso_image` y, a continuación, montamos el archivo de imagen `image.iso` en ese punto de montaje. Una vez montada la imagen, se puede tratar como si fuera un CD-ROM o DVD real. *Recuerde desmontar la imagen cuando ya no sea necesaria.*

### *Borrar un CD-ROM regrabable*

Los soportes CD-RW regrabables deben borrarse o *borrarse* antes de volver a utilizarse. Para ello, podemos utilizar `wodim`, especificando el nombre del dispositivo para la grabadora de CD y el tipo de borrado a realizar. El programa `wodim` ofrece varios tipos. El más mínimo (y rápido) es el tipo rápido:

---

```
wodim dev=/dev/cdrw blank=rápido
```

---

### *Escribir una imagen*

Para escribir una imagen, volvemos a usar `wodim`, especificando el nombre del dispositivo de escritura de medios ópticos y el nombre del archivo de imagen:

---

```
wodim dev=/dev/cdrw image.iso
```

---

Además del nombre del dispositivo y el archivo de imagen, `wodim` admite un conjunto muy grande de opciones. Dos de los más comunes son `-v` para la salida detallada y `-dao`, que escribe el disco en modo *disco a la vez*. Este modo debe utilizarse si está preparando un disco para su reproducción

comercial. El modo predeterminado para wodim es *track-at-once*, que es útil para grabar pistas de música.

## Crédito Extra

A menudo es útil verificar la integridad de una imagen ISO que hemos descargado. En la mayoría de los casos, un distribuidor de una imagen ISO también suministrará un *archivo de suma de comprobación*. Una suma de comprobación es el resultado de un cálculo matemático exótico que da como resultado un número que representa el contenido del archivo de destino. Si el contenido del archivo cambia incluso en un bit, la suma de comprobación resultante será muy diferente. El método más común de generación de suma de comprobación utiliza el programa md5sum. Cuando se utiliza md5sum, se produce un número hexadecimal único:

---

```
md5sum image.iso  
34e354760f9bb7fbf85c96f6a3f94ece image.iso
```

---

Después de descargar una imagen, debe ejecutar md5sum en ella y comparar los resultados con el valor md5sum proporcionado por el editor.

Además de verificar la integridad de un archivo descargado, podemos usar md5sum para verificar los medios ópticos recién escritos. Para hacer esto, primero calculamos la suma de verificación del archivo de imagen y luego calculamos una suma de verificación para el medio. El truco para verificar el medio es limitar el cálculo a solo la parte del medio óptico que contiene la imagen. Hacemos esto determinando el número de bloques de 2048 bytes que contiene la imagen (los medios ópticos siempre se escriben en bloques de 2048 bytes) y leyendo esa cantidad de bloques del medio. En algunos tipos de medios, esto no es obligatorio. Un CD-R grabado en modo disco a la vez se puede comprobar de la siguiente manera:

---

```
md5sum /dev/cdrom  
34e354760f9bb7fbf85c96f6a3f94ece /dev/cdrom
```

---

Muchos tipos de medios, como los DVD, requieren un cálculo preciso del número de bloques. En el siguiente ejemplo, comprobamos la integridad del archivo de imagen *dvd-image.iso* y del disco en el lector de DVD */dev/dvd*. ¿Puedes averiguar cómo funciona esto?

---

```
md5sum dvd-image.iso; dd if=/dev/dvd bs=2048 count=$(( $(stat -c "%s" dvd-image.iso) / 2048 )) | md5sum
```

---

# 16

## GESTIÓN DE REDES

Cuando se trata de redes, probablemente no haya nada que no se pueda hacer con Linux. Linux se utiliza para construir todo tipo de sistemas y aplicaciones de red , incluidos firewalls, enrutadores, servidores de nombres, cajas NAS (almacenamiento conectado a la red), etc.

Al igual que el tema de las redes es vasto, también lo es el número de comandos que se pueden utilizar para configurarlo y controlarlo. Centraremos nuestra atención en algunos de los más utilizados. Los comandos elegidos para el examen incluyen los que se utilizan para monitorear redes y los que se usan para transferir archivos. Además, vamos a explorar el programa ssh, que se utiliza para realizar inicios de sesión remotos. En este capítulo se tratarán los siguientes temas:

- ping: envíe un ECHO\_REQUEST ICMP a los hosts de red.
- traceroute: imprime el seguimiento de paquetes de ruta a un host de red.
- netstat: imprime conexiones de red, tablas de enrutamiento, estadísticas de interfaz, conexiones de enmascaramiento y membresías de multidifusión.
- ftp: programa de transferencia de archivos por Internet.

- **lftp**: un programa mejorado de transferencia de archivos por Internet.
- **wget**: descargador de red no interactivo.
- **ssh**: cliente SSH de OpenSSH (programa de inicio de sesión remoto).
- **scp**: copia segura (programa de copia remota de archivos).
- **sftp**: programa de transferencia segura de archivos.

Vamos a suponer un poco de experiencia en redes. En esta era de Internet, todos los que usan una computadora necesitan una comprensión básica de los conceptos de redes. Para hacer pleno uso de este capítulo, debe familiarizarse con los siguientes términos:

- Dirección IP (protocolo de Internet)
- Host y nombre de dominio
- URI (identificador uniforme de recursos)

**Nota:** Algunos de los comandos que cubriremos pueden (dependiendo de su distribución) requerir la instalación de paquetes adicionales de los repositorios de su distribución, y algunos pueden requerir privilegios de superusuario para ejecutarse.

## Examen y monitoreo de una red

Incluso si no es el administrador del sistema, a menudo es útil examinar el rendimiento y el funcionamiento de una red.

### **ping: enviar un paquete especial a un host de red**

El comando de red más básico es ping. El comando ping envía un paquete de red específico llamado ICMP ECHO\_REQUEST a un host especificado. La mayoría de los dispositivos de red que reciben este paquete responderán a él, lo que permitirá verificar la conexión a la red.

**Nota:** Es posible configurar la mayoría de los dispositivos de red (incluidos los hosts Linux) para ignorar estos paquetes. Esto generalmente se hace por razones de seguridad, para ocultar parcialmente un host de un posible atacante. También es común que los firewalls se configuren para bloquear el tráfico ICMP.

Por ejemplo, para ver si podemos llegar a <http://www.linuxcommand.org/> (uno de mis sitios favoritos ;-)), podemos usar ping así:

---

```
[me@linuxbox ~]$ ping linuxcommand.org
```

---

Una vez iniciado, ping continúa enviando paquetes en un intervalo especificado (el valor predeterminado es 1 segundo) hasta que se interrumpe:

---

```
[me@linuxbox ~]$ ping linuxcommand.org
PING linuxcommand.org (66.35.250.210) 56(84) bytes de datos.
```

---

```
64 bytes de vhost.sourceforge.net (66.35.250.210): icmp_seq=1 ttl=43 tiempo=10
7 ms
64 bytes de vhost.sourceforge.net (66.35.250.210): icmp_seq=2 ttl=43 tiempo=10
8 ms
64 bytes de vhost.sourceforge.net (66.35.250.210): icmp_seq=3 ttl=43 tiempo=10
6 ms
64 bytes de vhost.sourceforge.net (66.35.250.210): icmp_seq=4 ttl=43 tiempo=10
6 ms
64 bytes de vhost.sourceforge.net (66.35.250.210): icmp_seq=5 ttl=43 tiempo=10
5 ms
64 bytes de vhost.sourceforge.net (66.35.250.210): icmp_seq=6 ttl=43 tiempo=10
7 ms

--- linuxcommand.org estadísticas de ping ---
6 paquetes transmitidos, 6 recibidos, 0% de pérdida de paquetes,
tiempo 6010ms rtt min/avg/max/mdev =
105,647/107,052/108,118/0,824 ms
```

---

Después de que se interrumpe (en este caso después del sexto paquete) presionando CTRL-C, ping imprime estadísticas de rendimiento. Una red que funcione correctamente exhibirá un cero por ciento de pérdida de paquetes. Un ping exitoso indicará que los elementos de la red (sus tarjetas de interfaz, cableado, enrutamiento y puertas de enlace) están en buen estado de funcionamiento en general.

### ***traceroute: rastrea la ruta de un paquete de red***

El programa traceroute (algunos sistemas utilizan el programa tracepath similar en su lugar) muestra una lista de todos los "saltos" que tarda el tráfico de red en llegar desde el sistema local a un host especificado. Por ejemplo, para ver la ruta que se toma para llegar a <http://www.slashdot.org/>, haríamos lo siguiente:

---

```
[me@linuxbox ~]$ traceroute slashdot.org
```

---

El resultado es el siguiente:

```
traceroute a slashdot.org (216.34.181.45), 30 saltos máx., paquetes de 40 bytes
1 ipcop.localdomain (192.168.1.1) 1,066 ms 1,366 ms 1,720 ms
2 * * *
3 ge-4-13-ur01.rockville.md.bad.comcast.net (68.87.130.9) 14.622 ms 14.885
MS 15.169 ms
4 po-30-ur02.rockville.md.bad.comcast.net (68.87.129.154) 17.634 ms 17.626
MS 17.899 MS
5 po-60-ur03.rockville.md.bad.comcast.net (68.87.129.158) 15.992 ms 15.983
MS 16.256 ms
6 po-30-ar01.howardcounty.md.bad.comcast.net (68.87.136.5) 22.835 ms 14.23
3 ms 14.405 ms
7 po-10-ar02.whitemarsh.md.bad.comcast.net (68.87.129.34) 16.154 ms 13.600
MS 18.867 MS
8 te-0-3-0-1-cr01.philadelphia.pa.ibone.comcast.net (68.86.90.77) 21.951 ms
21.073 ms 21.557 ms
9 pos-0-8-0-0-cr01.newyork.ny.ibone.comcast.net (68.86.85.10) 22.917 ms 21
.884 ms 22.126 ms
10 204.70.144.1 (204.70.144.1) 43.110 ms 21.248 ms 21.264 ms
11 cr1-pos-0-7-3-1.newyork.savvis.net (204.70.195.93) 21.857 ms cr2-pos-0-0-
3-1.newyork.savvis.net (204.70.204.238) 19.556 ms cr1-pos-0-7-3-1.newyork.sav
vis.net (204.70.195.93) 19.634 ms
```

```
12 cr2-pos-0-7-3-0.chicago.savvis.net (204.70.192.109) 41,586 ms 42,843 ms  
cr2-tengig-0-0-2-0.chicago.savvis.net (204.70.196.242) 43,115 ms  
13 hr2-tengigabitethernet-12-1.elkgrovech3.savvis.net (204.70.195.122) 44.21  
5 ms 41.833 ms 45.658 ms  
14 csr1-ve241.elkgrovech3.savvis.net (216.64.194.42) 46.840 ms 43.372 ms 4  
7.041 ms  
15 64.27.160.194 (64.27.160.194) 56.137 ms 55.887 ms 52.810 ms  
16 slashdot.org (216.34.181.45) 42.727 ms 42.016 ms 41.437 ms
```

---

En el resultado, podemos ver que conectarse desde nuestro sistema de prueba a <http://www.slashdot.org/> requiere atravesar 16 enrutadores. En el caso de los routers que proporcionan información de identificación, vemos sus nombres de host, direcciones IP y datos de rendimiento, que incluyen tres muestras de tiempo de ida y vuelta desde el sistema local hasta el router. En el caso de los routers que no proporcionan información de identificación (debido a la configuración del router, la congestión de la red, los cortafuegos, etc.), vemos asteriscos como en la línea del salto número dos.

### ***netstat: examinar la configuración de red y las estadísticas***

El programa netstat se utiliza para examinar diversas configuraciones y estadísticas de la red. A través del uso de sus muchas opciones, podemos ver una variedad de características en nuestra configuración de red. Usando la opción **-ie**, podemos examinar las interfaces de red en nuestro sistema:

```
[me@linuxbox ~]$ netstat -ie  
eth0    Conector de enlace:Ethernet HWaddr 00:1d:09:9b:99:67  
        inet addr:192.168.1.2 bcast:192.168.1.255 mascara:255.255.255.0  
        inet6 addr: fe80::21d:9ff:fe9b:9967/64 Alcance:Enlace  
        TRANSMISIÓN ASCENDENTE EN EJECUCIÓN MULTICAST  
        MTU:1500 Métrica:1  
        Paquetes RX:238488 errores:0 descartados:0  
        desbordamientos:0 trama:0 paquetes TX:403217 errores:0  
        descartados:0 saturaciones:0 portador:0 colisiones:0  
        txqueuelen:100  
        Bytes RX:153098921 (146.0 MB) Bytes TX:261035246 (248.9 MB)  
        Memoria:fd0000-fdfe0000  
  
lo     Encap de enlace:Bucle invertido local  
        inet addr:127.0.0.1 Mask:255.0.0.0  
        inet6 addr: ::1/128 Scope:Host  
        BUCLE INVERTIDO ASCENDENTE EN EJECUCIÓN MTU:16436 Métrica:1  
        Paquetes RX:2208 errores:0 descartados:0 saturaciones:0  
        trama:0 paquetes TX:2208 errores:0 descartados:0  
        saturaciones:0 portador:0 colisiones:0 txqueuelen:0  
        Bytes RX:111490 (108.8 KB) Bytes TX:111490 (108.8 KB)
```

---

En el ejemplo anterior, vemos que nuestro sistema de prueba tiene dos interfaces de red. La primera, llamada **eth0**, es la interfaz Ethernet; la segunda, llamada **lo**, es la *interfaz de bucle invertido*, una interfaz virtual que el sistema utiliza para "hablar consigo mismo".

Al realizar diagnósticos de red causales, lo importante que se debe buscar es la presencia de la palabra UP al principio de la cuarta línea de cada interfaz, lo que indica que la interfaz de red está habilitada, y la presencia de una dirección IP válida en el campo **inet addr** de la segunda

línea. Para los sistemas que utilizan el Protocolo de configuración dinámica de host (DHCP), una dirección IP válida en este campo verificará que el DHCP está funcionando.

El uso de la opción `-r` mostrará la tabla de enrutamiento de red del kernel. Esto muestra cómo se configura la red para enviar paquetes de una red a otra:

```
[me@linuxbox -]$ netstat -r
Tabla de enrutamiento IP del kernel
Destino GatewayGenmask          Banderas Ventana MSS IRTT Iface
192.168.1.0 *                  255.255.255.0 U0        00 eth0 por defecto
192.168.1.1  0.0.0.0            UG         0 0           0 eth0
```

En este ejemplo simple, vemos una tabla de enrutamiento típica para una máquina cliente en una red de área local (LAN) detrás de un firewall/enrutador. La primera línea de la lista muestra el destino 192.168.1.0. Las direcciones IP que terminan en cero se refieren a redes en lugar de hosts individuales, por lo que este destino significa cualquier host en la LAN. El siguiente campo, Puerta de enlace, es el nombre o la dirección IP de la puerta de enlace (enrutador) utilizada para ir desde el host actual hasta la red de destino. Un asterisco en este campo indica que no se necesita ninguna puerta de enlace.

La última línea contiene el valor predeterminado de destino. Esto significa cualquier tráfico destinado a una red que no se enumera en la tabla. En nuestro ejemplo, vemos que la puerta de enlace se define como un enrutador con la dirección 192.168.1.1, que presumiblemente sabe qué hacer con el tráfico de destino.

El programa netstat tiene muchas opciones, y hemos visto solo un par. Echa un vistazo a la página del manual de netstat para obtener una lista completa.

## Transporte de archivos a través de una red

¿De qué sirve una red si no sabemos cómo mover archivos a través de ella? Hay muchos programas que mueven datos a través de las redes. Cubriremos dos de ellos ahora y varios más en secciones posteriores.

### *ftp: transfiera archivos con el protocolo de transferencia de archivos*

Uno de los verdaderos programas "clásicos", ftp recibe su nombre del protocolo que utiliza, el Protocolo de Transferencia de Archivos. FTP se usa ampliamente en Internet para descargar archivos. La mayoría de los navegadores web, si no todos, lo admiten y, a menudo, los URI comienzan con el protocolo `ftp://`.

Antes de que existieran los navegadores web, existía el programa ftp. FTP se utiliza para comunicarse con *servidores FTP*, máquinas que contienen archivos que se pueden cargar y descargar a través de una red.

FTP (en su forma original) no es seguro, ya que envía nombres de cuentas y contraseñas en *texto sin cifrar*. Esto significa que no están encriptados y cualquiera que husmee la red puede verlos. Debido a esto, casi todo el FTP realizado a través de Internet lo realizan servidores *FTP anónimos*. Un servidor anónimo permite que cualquier persona inicie sesión con el nombre de inicio de sesión *anónimo* y una contraseña sin sentido.

En el siguiente ejemplo, mostramos una sesión típica con el

programa ftp descargando una imagen ISO de Ubuntu ubicada en el directorio */pub/cd\_images/ Ubuntu-8.04* del servidor de archivos FTP anónimo.

---

```
[me@linuxbox ~]$ ftp fileserver
Conectado a fileserver.localdomain.
220 (vsFTPd 2.0.1)
Nombre (fileserver:me): anónimo
331 Por favor, especifique la
contraseña. Contraseña:
230 Inicio de sesión exitoso.
El tipo de sistema remoto es
UNIX.
Uso del modo binario para transferir
archivos. ftp> cd
pub/cd_images/Ubuntu-8.04 250
Directorio cambiado con éxito. ftp> ls
200 PORT comando exitoso. Considera la posibilidad de
utilizar PASV. 150 Aquí viene la lista del directorio.
-rw-rw-r--          1 500500          733079552 25 de abril 03:53 ubuntu-8.04-
desktop- i386.iso
226 Directorio enviar OK.
ftp> lcd Escritorio
Directorio local ahora /home/me/Desktop
ftp> obtener ubuntu-8.04-desktop-
i386.iso
local: ubuntu-8.04-desktop-i386.iso remoto: ubuntu-8.04-desktop-i386.iso
comando 200 PORT exitoso. Considera la posibilidad de utilizar PASV.
150 Apertura de la conexión de datos en modo BINARIO para ubuntu-8.04-desktop-
i386.iso (733079552 bytes).
226 Envío de archivo OK.
733079552 bytes recibidos en 68,56 segundos (10441,5
kB/s) ftp> adiós
```

---

En la Tabla 16-1 se ofrece una explicación de los comandos introducidos durante esta sesión.

**Tabla 16-1: Ejemplos de comandos ftp interactivos**

Mandar	Significado
FTP fileserverInvoke	el FTP programa y haz que se conecte al servidor FTP <i>Servidor de archivos</i> .
anónimo	Nombre de inicio de sesión. Después de la solicitud de inicio de sesión, aparecerá una solicitud de contraseña. Algunos servidores aceptarán una contraseña en blanco. Otros requerirán una contraseña en
	forma de dirección de correo electrónico. En ese caso, pruebe algo como <i>user@example.com</i> .
cd pub/cd_images/Ubuntu-8.04	Cambiar al directorio del mando a distancia sistema que contiene el archivo deseado. Tenga en cuenta que en la mayoría de los servidores FTP anónimos, los archivos para descarga

pública se encuentran en algún lugar debajo del *directorio pub*.

ls

---

Listar el directorio en el sistema remoto.

Cuadro 16-1 (continuación )

Mandar	Significado
LCD EscritorioCambio el directorio del sistema local a	~/Escritorio. En el ejemplo, el método FTP se invocaba cuando el directorio de trabajo era ~. Este comando cambia el directorio de trabajo a ~/Escritorio.
Obtener ubuntu-8.04-desktop-i386.iso	Dígale al sistema remoto que transfiera el archivo <i>ubuntu-8.04-desktop-i386.iso</i> al sistema local. Dado que el directorio de trabajo en el sistema local se cambió a ~/Desktop, el archivo se descargará allí.
Adiós	Cierre la sesión del servidor remoto y finalice el FTP Sesión del programa. Los comandos renunciar y salida también se puede utilizar.

Al escribir help en el prompt ftp> se mostrará una lista de los comandos soportados. Usando ftp en un servidor donde se han otorgado suficientes permisos, es posible realizar muchas tareas ordinarias de administración de archivos. Es torpe, pero funciona.

### *lftp: un ftp mejor*

ftp no es el único cliente FTP de línea de comandos. De hecho, hay muchos. Uno de los mejores (y más populares) es lftp de Alexander Lukyanov. Funciona de manera muy similar al programa ftp tradicional, pero tiene muchas características convenientes adicionales, que incluyen soporte para múltiples protocolos (incluido HTTP), reinicio automático en descargas fallidas, procesos en segundo plano, finalización de pestañas de nombres de rutas y muchas más.

### *wget: descargador de red no interactivo*

Otro popular programa de línea de comandos para descargar archivos es wget. Es útil para descargar contenido de sitios web y FTP. Se pueden descargar archivos individuales, varios archivos e incluso sitios completos. Para descargar la primera página de <http://www.linuxcommand.org/>, podríamos hacer lo siguiente:

```
[me@linuxbox ~]$ wget http://linuxcommand.org/index.php
--11:02:51-- http://linuxcommand.org/index.php
          => 'index.php'
Resolviendo linuxcommand.org... 66.35.250.210
Conectando a linuxcommand.org|66.35.250.210|:80|: conexo.
```

Solicitud HTTP enviada, a la espera de respuesta... 200  
OK Longitud: no especificada [text/html]

[ <=>

] 3,120 ----- K/s

11:02:51 (161.75 MB/s) - 'index.php' guardado [3120]

---

Las numerosas opciones del programa permiten a wget descargar de forma recursiva, descargar archivos en segundo plano (lo que le permite cerrar la sesión pero continuar con la descarga descendente) y completar la descarga de un archivo parcialmente descargado. Estas características están bien documentadas en su página de manual mejor que la media.

## Comunicación segura con hosts remotos

Durante muchos años, los sistemas operativos tipo Unix han tenido la capacidad de ser administrados de forma remota a través de una red. En los primeros días, antes de la adopción general de Internet, había un par de programas populares que se usaban para iniciar sesión en hosts remotos: los programas rlogin y telnet. Estos programas, sin embargo, sufren de la misma falla fatal que el programa ftp: transmiten todas sus comunicaciones (incluidos los nombres de inicio de sesión y las contraseñas) en texto sin cifrar. Esto los hace totalmente inapropiados para su uso en la era de Internet.

### **ssh: inicie sesión de forma segura en equipos remotos**

Para abordar este problema, se desarrolló un nuevo protocolo llamado SSH (Secure Shell). SSH resuelve los dos problemas básicos de la comunicación segura con un host remoto. En primer lugar, autentica que el host remoto es quien dice ser (evitando así los ataques de intermediario) y, en segundo lugar, cifra todas las comunicaciones entre los hosts locales y remotos.

SSH consta de dos partes. Un servidor SSH se ejecuta en el host remoto, escuchando las conexiones entrantes en el puerto 22, mientras que un cliente SSH se utiliza en el sistema local para comunicarse con el servidor remoto.

La mayoría de las distribuciones de Linux incluyen una implementación de SSH llamada OpenSSH del proyecto BSD. Algunas distribuciones incluyen tanto el paquete de cliente como el de servidor de forma predeterminada (por ejemplo, Red Hat), mientras que otras (como Ubuntu) solo suministran el cliente. Para permitir que un sistema reciba conexiones remotas, debe tener el paquete de servidor OpenSSH instalado, configurado y en ejecución, y (si el sistema está en ejecución o detrás de un firewall) debe permitir conexiones de red entrantes en el puerto TCP 22.

**Nota:** Si no tiene un sistema remoto al que conectarse, pero desea probar estos ejemplos, asegúrese de que el paquete OpenSSH-server esté instalado en su sistema y use localhost como nombre del host remoto. De esa manera, su máquina creará conexiones de red consigo misma.

El programa cliente SSH utilizado para conectarse a servidores SSH remotos se llama, apropiadamente, ssh. Para conectarnos a un host remoto llamado `remote-sys`, usaríamos el programa cliente ssh de la siguiente manera:

---

```
[me@linuxbox ~]$ ssh remote-sys  
No se puede establecer la autenticidad del host 'remote-sys (192.168.1.4).  
La huella digital de la clave RSA es  
41:ed:7a:df:23:19:bf:3c:a5:17:bc:61:b3:7f:d9:bb. ¿Estás seguro de que  
quieres seguir conectándote (sí/no)?
```

---

La primera vez que se intenta la conexión, se muestra un mensaje que indica que no se puede establecer la autenticidad del host remoto. Esto se debe a que el programa cliente nunca antes ha visto este host remoto. Para aceptar las credenciales del host remoto, escriba `sí` cuando se le solicite. Una vez establecida la conexión, se solicita al usuario una contraseña:

---

Advertencia: Se ha añadido permanentemente 'remote-sys,192.168.1.4' (RSA) a la lista de hosts conocidos.  
Contraseña de me@remote-sys:

---

Una vez que la contraseña se ingresa con éxito, recibimos el mensaje del shell del sistema remoto:

---

```
Último inicio de sesión: Tue Aug 30  
13:00:48 2011 [me@remote-sys ~]$
```

---

La sesión de shell remoto continúa hasta que el usuario ingresa al comando de salida en el indicador de shell remoto, cerrando así la conexión remota. En este punto, se reanuda la sesión de shell local y vuelve a aparecer el símbolo del sistema de shell local.

También es posible conectarse a sistemas remotos utilizando un nombre de usuario diferente. Por ejemplo, si el usuario local `me` tenía una cuenta denominada `bob` en un sistema remoto, el usuario `me` podría iniciar sesión en la cuenta `bob` en el sistema remoto de la siguiente manera:

---

```
[me@linuxbox ~]$ ssh bob@remote-sys  
Contraseña de bob@remote-sys:  
Último inicio de sesión: Tue Aug 30  
13:03:21 2011 [bob@remote-sys ~]$
```

---

Como se indicó anteriormente, ssh verifica la autenticidad del host remoto. Si el host remoto no se autentica correctamente, aparece el siguiente mensaje:

---

```
[me@linuxbox ~]$ ssh remoto-sys  
@oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo  
@ ADVERTENCIA: ¡LA IDENTIFICACIÓN DEL HOST REMOTO HA  
CAMBIADO!  
@  
@oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo  
¡ES POSIBLE QUE ALGUIEN ESTÉ HACIENDO ALGO DESAGRADABLE!  
¡Alguien podría estar espiándote ahora mismo (ataque de intermediario)!  
También es posible que se haya cambiado la clave de host RSA.
```

---

La huella digital de la clave RSA enviada por el host remoto es  
41:ed:7a:df:23:19:bf:3c:a5:17:bc:61:b3:7f:d9:bb.  
Póngase en contacto con el administrador del sistema.  
Agregue la clave de host correcta en /home/me/.ssh/known\_hosts para deshacerse de este  
mensaje. Clave ofensiva en /home/me/.ssh/known\_hosts:1  
La clave de host RSA para remote-sys ha cambiado y ha solicitado una  
comprobación estricta.  
Error en la verificación de la clave de host.

---

Este mensaje se debe a una de dos situaciones posibles. En primer lugar, un atacante puede estar intentando un ataque de intermediario. Esto es raro, porque todo el mundo sabe que ssh alerta al usuario de esto. El culpable más probable es que el sistema remoto haya cambiado de alguna manera; por ejemplo, se ha reinstalado su sistema operativo o servidor SSH. Sin embargo, en aras de la seguridad, no se debe descartar de plano la primera posibilidad.

Consulte siempre con el administrador del sistema remoto cuando se produzca este mensaje.

Después de determinar que el mensaje se debe a una causa benigna, es seguro corregir el problema en el lado del cliente. Esto se hace mediante el uso de un editor de texto (vim quizás) para eliminar la clave obsoleta del archivo *~/.ssh/known\_hosts*. En el mensaje de ejemplo anterior, vemos lo siguiente:

---

Clave ofensiva en /home/me/.ssh/known\_hosts:1

---

Esto significa que la línea 1 del *archivo known\_hosts* contiene la clave infractora. Elimine esta línea del archivo y el programa ssh podrá aceptar nuevas credenciales de autenticación del sistema remoto.

Además de abrir una sesión de shell en un sistema remoto, ssh también nos permite ejecutar un solo comando en un sistema remoto. Por ejemplo, podemos ejecutar el comando free en un host remoto llamado *remote-sys* y hacer que los resultados se muestren en el sistema local:

---

```
[me@linuxbox ~]$ ssh remote-sys gratis
Contraseña de me@twin4:
              total   usado   Gratis   compartido   Búferes   Caché
Mem:        775536   507184   268352                   0   110068   154596
-/+ búferes/caché:  242520   533016
Intercambio:                0   1572856
                           1572856
[me@linuxbox ~]$
```

---

Es posible usar esta técnica de formas más interesantes, como este ejemplo en el que realizamos un ls en el sistema remoto y redirigimos la salida a un archivo en el sistema local:

---

```
[me@linuxbox ~]$ ssh remote-sys 'ls *' > dirlist.txt
Contraseña de me@twin4:
[me@linuxbox ~]$
```

---

Fíjate en el uso de las comillas simples. Esto se hace porque no queremos que la expansión del nombre de ruta se realice en la máquina local; más bien, queremos que se realice en el sistema remoto. Del mismo

modo, si hubiéramos querido la salida

redirigido a un archivo en la máquina remota, podríamos haber colocado el operador de redireccionamiento y el nombre del archivo entre comillas simples:

---

```
[me@linuxbox ~]$ ssh remote-sys 'ls * > dirlist.txt'
```

---

## TUNELIZACIÓN CON SSH

Parte de lo que sucede cuando se establece una conexión con un host remoto a través de SSH es que se crea un *túnel cifrado* entre los sistemas local y remoto.

Normalmente, este túnel se utiliza para permitir que los comandos escritos en el sistema local se transmitan de forma segura al sistema remoto y que los resultados se transmitan de forma segura. Además de esta función básica, el protocolo SSH permite que la mayoría de los tipos de tráfico de red se envíen a través del túnel cifrado, creando una especie de *VPN* (red privada virtual) entre los sistemas local y remoto.

Quizás el uso más común de esta función es permitir que se transmita el tráfico del sistema X Window. En un sistema que ejecuta un servidor X (es decir, una máquina que muestra una interfaz gráfica de usuario), es posible iniciar y ejecutar un programa cliente X (una aplicación gráfica) en un sistema remoto y hacer que su visualización aparezca en el sistema local. Es fácil de hacer, aquí hay un ejemplo. Digamos que estamos sentados en un sistema Linux llamado *linuxbox* que está ejecutando un servidor X, y queremos ejecutar el programa *xload* en un sistema remoto llamado *remote-sys* y ver la salida gráfica del programa en nuestro sistema local. Podríamos hacer lo siguiente:

```
[me@linuxbox ~]$ ssh -X remote-sys  
Contraseña de me@remote-sys:  
Último inicio de sesión: Mon Sep 05  
13:23:11 2011 [me@remote-sys ~]$  
xload
```

Una vez que se ejecuta el comando *xload* en el sistema remoto, su ventana aparece en el sistema local. En algunos sistemas, es posible que deba usar la opción *-Y* en lugar de la opción *-X* para hacer esto.

## scp y sftp: transfiere archivos de forma segura

El paquete OpenSSH también incluye dos programas que pueden hacer uso de un túnel cifrado SSH para copiar archivos a través de la red. El primero, *scp* (copia segura) se utiliza de forma muy similar al conocido programa *cp* para copiar archivos. La diferencia más notable es que el nombre de ruta de origen o destino puede ir precedido por el nombre de un host remoto seguido de un carácter de dos puntos. Por ejemplo, si quisieramos copiar un documento llamado *document.txt* de nuestro directorio de inicio en el sistema remoto, *remote-sys*, al directorio de trabajo actual en nuestro sistema local, podríamos hacer esto:

---

```
[me@linuxbox ~]$ scp remote-sys:document.txt .  
Contraseña de me@remote-sys:  
document.txt 100% 55815.5KB/s
```

00:00 [me@linuxbox ~] \$

---

Al igual que con ssh, puede aplicar un nombre de usuario al principio del nombre del host remoto si el nombre de la cuenta de host remoto deseado no coincide con el del sistema local:

---

```
[me@linuxbox ~]$ scp bob@remote-sys:document.txt .
```

---

El segundo programa de copia de archivos SSH es sftp, que, como su nombre lo indica, es un reemplazo seguro para el programa ftp. sftp funciona de manera muy similar al programa FTP original que usamos anteriormente; sin embargo, en lugar de transmitir todo en texto claro, utiliza un túnel cifrado con SSH. SFTP tiene una ventaja importante sobre el FTP convencional en que no requiere que un servidor FTP se ejecute en el host remoto. Solo requiere el servidor SSH.

Esto significa que cualquier máquina remota que pueda conectarse con el cliente SSH también se puede utilizar como un servidor similar a FTP. A continuación, se muestra un ejemplo de sesión:

---

```
[me@linuxbox ~]$ sftp remote-sys
Conectando a remote-sys...
Contraseña de me@remote-sys:
SFTP> LS
ubuntu-8.04-desktop-i386.iso
sftp> lcd de sobremesa
SFTP> obtener ubuntu-8.04-desktop-i386.iso
Obtener /home/me/ubuntu-8.04-desktop-i386.iso para ubuntu-8.04-desktop-i386.iso

/inicio/yo/ubuntu-8.04-desktop-i386.iso 100% 699 MB7,4 MB/s
                                         01:35 sftp>
Adiós
```

---

**Nota:** *El protocolo SFTP es compatible con muchos de los administradores de archivos gráficos que se encuentran en las distribuciones de Linux. Usando Nautilus (GNOME) o Konqueror (KDE), podemos ingresar un URI que comience con sftp:// en la barra de direcciones y operar en archivos almacenados en un sistema remoto que ejecuta un servidor SSH.*

### ¿UN CLIENTE SSH PARA WINDOWS?

Digamos que está sentado en una máquina con Windows, pero necesita iniciar sesión en su servidor Linux y realizar un trabajo real. ¿A qué te dedicas? ¡Obtenga un programa cliente SSH para su máquina Windows, por supuesto! Hay varios de estos. El más popular es probablemente PuTTY de Simon Tatham y su equipo. El programa PuTTY muestra una ventana de terminal y permite a un usuario de Windows abrir una sesión SSH (o telnet) en un host remoto. El programa también proporciona análogos para los programas scp y sftp.

PuTTY está disponible en <http://www.chiark.greenend.org.uk/~sgtatham/putty/>.

# 17

## BÚSQUEDA DE ARCHIVOS

A medida que hemos deambulado por nuestro sistema Linux, una cosa ha quedado muy clara: ¡Un sistema Linux típico tiene muchos archivos! Esto plantea la pregunta: "¿Cómo encontramos las cosas?" Ya sabemos que el Linux

El sistema de archivos está bien organizado de acuerdo con las convenciones que se han transmitido de una generación de sistemas tipo Unix a la siguiente, pero la gran cantidad de archivos puede presentar un problema desalentador.

En este capítulo, veremos dos herramientas que se utilizan para encontrar archivos en un sistema:

- `localizar`: busca archivos por nombre.
- `find`: busca archivos en una jerarquía de directorios.

También veremos un comando que se usa a menudo con comandos de búsqueda de archivos para procesar la lista de archivos resultante:

- `xargs`: compile y ejecute líneas de comandos a partir de una entrada estándar.



Además, introduciremos un par de comandos para ayudarnos en nuestras exploraciones:

- touch: cambia las horas de los archivos.
- stat: muestra el estado del archivo o del sistema de archivos.

## localizar: encuentre archivos de forma sencilla

El programa de localización realiza una búsqueda rápida en la base de datos de nombres de ruta y, a continuación, genera todos los nombres que coinciden con una subcadena determinada. Digamos, por ejemplo, que queremos encontrar todos los programas con nombres que comiencen con *zip*. Dado que estamos buscando programas, podemos suponer que el nombre del directorio que contiene los programas terminaría con *bin/*. Por lo tanto, podríamos intentar usar locate de esta manera para encontrar nuestros archivos:

---

```
[me@linuxbox -]$ localizar contenedor/zip
```

---

locate buscará en su base de datos de nombres de rutas y generará cualquiera que contenga la cadena *bin/zip*:

---

```
/usr/bin/zip  
/usr/bin/zipcloak  
/usr/bin/zipgrep  
/impacto/papelera/zipinfo  
/usr/bin/zipnote  
/usr/bin/zipsplit
```

---

Si el requisito de búsqueda no es tan sencillo, locate se puede combinar con otras herramientas, como grep, para diseñar búsquedas más interesantes:

---

```
[me@linuxbox -]$ localizar zip | grep bin  
/papelera/bunzip2  
/papelera/bzip2  
/bin/bzip2recover  
/papelera/gunzip  
/bin/gzip  
/usr/bin/funzip  
/usr/bin/gpg-zip  
/usr/bin/predescomprimir  
/usr/bin/prezip  
/usr/bin/prezip-bin  
/usr/bin/descomprimir  
/usr/bin/unzipsfx  
/usr/bin/zip  
/usr/bin/zipcloak  
/usr/bin/zipgrep  
/impacto/papelera/zipinfo  
/usr/bin/zipnote  
/usr/bin/zipsplit
```

---

El programa de localización existe desde hace varios años y hay varias variantes diferentes de uso común. Los dos más comunes que se

encuentran en las distribuciones modernas de Linux son slocate y mlocate, aunque generalmente son

Se accede mediante un enlace simbólico llamado `localizar`. Las diferentes versiones de `locate` tienen conjuntos de opciones superpuestos. Algunas versiones incluyen coincidencia de expresiones regulares (que trataremos en el Capítulo 19) y compatibilidad con comodines. Compruebe la página del comando `man` de `locate` para determinar qué versión de `locate` está instalada.

### ¿DE DÓNDE PROVIENE LA BASE DE DATOS DE LOCALIZACIÓN?

Es posible que observe que, en algunas distribuciones, `locate` no funciona justo después de instalar el sistema, pero si vuelve a intentarlo al día siguiente, funciona bien. ¿Qué pasa? La base de datos de localización es creada por otro programa llamado `updatedb`. Por lo general, se ejecuta periódicamente como un *trabajo cron*, es decir, una tarea realizada a intervalos regulares por el demonio cron. La mayoría de los sistemas equipados con `locate` se actualizan una vez al día. Dado que la base de datos no se actualiza continuamente, notará que los archivos muy recientes no aparecen cuando se usa `localizar`. Para superar esto, es posible ejecutar el programa `updatedb` manualmente convirtiéndose en el superusuario y

## find: encuentre archivos de la manera difícil

Mientras que el programa de localización puede encontrar un archivo basándose únicamente en su nombre, el programa de búsqueda busca archivos en un directorio determinado (y sus subdirectorios) en función de una variedad de atributos. Vamos a dedicar mucho tiempo a `find` porque tiene un montón de características interesantes que veremos una y otra vez cuando empiezemos a cubrir los conceptos de programación en capítulos posteriores.

En su uso más simple, `find` recibe uno o más nombres de directorios para buscar. Por ejemplo, puede producir una lista de nuestro directorio de inicio:

---

```
[me@linuxbox ~]$ encontrar ~
```

---

En la mayoría de las cuentas de usuario activas, esto producirá una lista grande. Dado que el

La lista se envía a la salida estándar, podemos canalizar la lista a otros programas. Usemos `wc` para contar el número de archivos:

---

```
[me@linuxbox ~]$ encontrar ~ | wc -l  
47068
```

---

¡Vaya, hemos estado ocupados! Lo bueno de la búsqueda es que se puede utilizar para identificar archivos que cumplen criterios específicos. Lo hace a través de la aplicación (un poco extraña) de *pruebas, acciones y opciones*. Primero veremos las pruebas.

### Pruebas

Digamos que queremos una lista de directorios de nuestra búsqueda. Para ello, podríamos añadir la siguiente prueba:

---

```
[me@linuxbox ~]$ find ~ -type d | wc -l  
1695
```

---

Al agregar el tipo de prueba **d** , se limitó la búsqueda a directorios. Por el contrario, podríamos haber limitado la búsqueda a archivos normales con esta prueba:

---

```
[me@linuxbox ~]$ find ~ -type f | wc -l  
38737
```

---

En la tabla 17-1 se enumeran las pruebas de tipo de archivo comunes admitidas por **find**.

**Tabla 17-1: Buscar tipos de archivos**

Archivo	Tipo	Descripción
b		Bloquear archivo de dispositivo especial
c		Archivo de dispositivo especial de caracteres
d		Directorio
f		Archivo regular
l		Simbólico enlace

También podemos buscar por tamaño de archivo y nombre de archivo agregando algunas pruebas adicionales. Busquemos todos los archivos regulares que coincidan con el patrón de comodines **\*.JPG** y tienen más de 1 megabyte:

---

```
[me@linuxbox ~]$ find ~ -type f -name "*.JPG" -size +1M | wc -l  
840
```

---

En este ejemplo, agregamos la prueba **-name** seguida del patrón de caracteres comodín. Tenga en cuenta que lo incluimos entre comillas para evitar la expansión del nombre de ruta por parte del shell. A continuación, añadimos la prueba **-size** seguida de la cadena **+1M**. El signo más inicial indica que estamos buscando archivos más grandes que el número especificado. Un signo menos a la izquierda cambiaría la cadena para que significase "más pequeño que el número especificado". El uso de ningún signo significa "coincidir exactamente con el valor". La letra **M** final indica que la unidad de medida es megabytes. Los caracteres que se muestran en la Tabla 17-2 se pueden utilizar para especificar unidades.

**Tabla 17-2: Buscar unidades de tamaño**

Carácter	Unidad
b	Bloques de 512 bytes (el valor predeterminado si no se especifica ninguna unidad)
c	Bytes
w	Palabras de 2 bytes
k	Kilobytes (unidades de 1024 bytes)

M	Megabytes (unidades de 1.048.576 bytes)
G	Gigabytes (unidades de 1.073.741.824 bytes)

find admite un gran número de pruebas diferentes. La Tabla 17-3 proporciona un resumen de los más comunes. Tenga en cuenta que en los casos en que se requiere un argumento numérico, se puede aplicar la misma notación + y - mencionada anteriormente.

**Tabla 17-3: Pruebas de búsqueda**

Prueba	Descripción
-cmin <i>nCocidencia</i>	archivos o directorios cuyo contenido o atributos se modificaron por última vez exactamente <i>n</i> hace minutos. Para especificar menos de <i>n</i> hace unos minutos, use - <i>n</i> ; para especificar más de <i>n</i> hace unos minutos, use + <i>n</i> .
-más nuevo <i>archivo</i>	Hacer coincidir archivos o directorios cuyo contenido o atributos fueron última modificación más reciente que las de <i>archivo</i> .
-ctime <i>nCocidencia</i>	archivos o directorios cuyo contenido o atributos (es decir, permisos) fueron modificados por última vez <i>n</i> *Hace 24 horas.
-vacío	Hacer coincidir archivos y directorios vacíos.
-grupo <i>nameCocide con el archivo o los directorios que pertenecen al nombre del grupo . nombre</i>	puede expresarse como un nombre de grupo o como un ID de grupo numérico.
-iname <i>patrón</i>	Al igual que el -nombre prueba, pero no distingue entre mayúsculas y minúsculas.
-inum <i>nCocidencia</i>	Archivos con número de inodo <i>n</i> . Esto es útil para encontrar todos los enlaces duros a un inodo en particular.
-mmin <i>n</i>	Hacer coincidir archivos o directorios cuyo contenido se modificó <i>n</i> hace minutos.
-mtime <i>nCocidencia</i>	archivos o directorios cuyo contenido solo fue el último modificado <i>n</i> *Hace 24 horas.
-nombre <i>patrón</i>	Hacer coincidir archivos y directorios con el comodín especificado <i>patrón</i> .
-Nuevos <i>archivo</i>	Hacer coincidir archivos y directorios cuyo contenido se modificó más recientemente que el especificado <i>archivo</i> . Esto es muy útil cuando se escriben scripts de shell que realizan copias de seguridad de archivos. Cada vez que realice una copia de seguridad, actualice un archivo (como un registro) y, a continuación, use encontrar para determinar qué archivos han cambiado desde la última actualización.
-Nouser	Coincidir con archivos y directorios que no pertenezcan a un usuario válido. Esto se puede utilizar para encontrar archivos que pertenecen a

cuentas eliminadas o para detectar la actividad de los atacantes.

- nogrupo      Coincidir con archivos y directorios que no pertenecen a un archivo válido grupo.
- 

*(continuación)*

Cuadro 17-3 (continuación )

Prueba	Descripción
-permanente <i>modeMatch</i>	archivos o directorios que tienen permisos establecidos en el <i>modo</i> . <i>modo</i> puede expresarse mediante notación octal o simbólica.
-samenombre de fichero	Similar a la prueba -inum. Coincide con los archivos que comparten el mismo número de inodo que el nombre de archivo.
-tamaño <i>n</i>	Hacer coincidir archivos de tamaño <i>n</i> .
-tipo <i>c</i>	Hacer coincidir archivos de tipo <i>c</i> .
-usuario <i>nameMatch</i>	archivos o directorios que pertenecen a <i>nombre</i> . <i>nombre</i> puede expresarse mediante un nombre de usuario o un ID de usuario numérico.

Esta no es una lista completa. La página del manual de búsqueda tiene todos los detalles.

### Operadores

Incluso con todas las pruebas que proporciona find , es posible que necesitemos una mejor manera de describir las *relaciones lógicas* entre las pruebas. Por ejemplo, ¿qué pasaría si necesitáramos determinar si todos los archivos y subdirectorios de un directorio tenían permisos seguros? Buscaríamos todos los archivos con permisos que no sean 0600 y los directorios con permisos que no sean 0700. Afortunadamente, find proporciona una manera de combinar pruebas mediante *operadores lógicos* para crear relaciones lógicas más complejas. Para expresar la prueba antes mencionada, podríamos hacer lo siguiente:

```
[me@linuxbox ~]$ find ~ \(`-type f -not -perm 0600 \|` -or \(`-type d -not -perm 0700 \|`)
```

¡Vaya! Eso sí que se ve raro. ¿Qué es todo esto? En realidad, los operadores no son tan complicados una vez que se llegan a conocerlos (véase la Tabla 17-4).

Tabla 17-4: Búsqueda de operadores lógicos

Operador	Descripción
-y	Coincide si las pruebas en ambos lados del operador son verdaderas. Puede acortarse a -un. Tenga en cuenta que cuando no hay ningún operador presente, -y está implícito de forma predeterminada.
-o	Coincide si una prueba en cualquiera de los lados del operador es verdadera. Puede acortarse a -o.
-no	Coincide si la prueba que sigue al operador es falsa.

Puede acortarse a -!.

---

Cuadro 17-4 (continuación )

Operador	Descripción
( )	<p>Agrupa pruebas y operadores para formar expresiones más grandes. Se utiliza para controlar la prioridad de las evaluaciones lógicas. De forma predeterminada, encontrar Evalúa de izquierda a derecha. A menudo es necesario invalidar el orden de evaluación predeterminado para obtener el resultado deseado. Incluso si no es necesario, a veces es útil incluir los caracteres de agrupación para mejorar la legibilidad del comando.</p> <p>Tenga en cuenta que, dado que los caracteres entre paréntesis tienen un significado especial para el shell, deben entrecambiarse cuando se usen en la línea de comandos para permitir que se pasen como argumentos para encontrar. Por lo general, el carácter de barra invertida se usa para escaparlos.</p>

Con esta lista de operadores en la mano, deconstruyamos nuestro comando find. Cuando se ve desde el nivel más alto, vemos que nuestras pruebas están organizadas como dos agrupaciones separadas por un operador -or:

(expresión 1) -o (expresión 2)

Esto tiene sentido, ya que estamos buscando archivos con un cierto conjunto de permisos y directorios con un conjunto diferente. Si buscamos tanto archivos como directorios, ¿por qué usamos -or en lugar de -and? Debido a que a medida que find escanea los archivos y directorios, cada uno se evalúa para ver si coincide con las pruebas especificadas. Queremos saber si se trata de un archivo con malos permisos *o* de un directorio con malos permisos. No pueden ser las dos cosas al mismo tiempo. Entonces, si expandimos las expresiones agrupadas, podemos verlo de esta manera:

(fichero con perms incorrectos) -o (directorio con perms incorrectos)

Nuestro próximo desafío es cómo probar los "permisos incorrectos". ¿Cómo lo hacemos? En realidad, no lo hacemos. Lo que probaremos es "permisos no buenos", ya que sabemos qué son los "permisos buenos". En el caso de los archivos, definimos *bueno* como 0600; para los directorios, 0700. La expresión que probará los archivos en busca de permisos "no buenos" es:

-type f -y -not -perms 0600

y la expresión para directorios es:

-tipo d -y -no -perms 0700

Como se indica en la tabla 17-4, el operador -and se puede quitar de forma segura, ya que está implícito de forma predeterminada. Entonces, si volvemos a juntar todo esto, obtenemos nuestro comando final:

find ~ (-type f -not -perms 0600) -or (-type d -not -perms 0700)

Sin embargo, dado que los paréntesis tienen un significado especial

para el shell, debemos escaparlos para evitar que el shell intente interpretarlos. Preceder a cada uno con un carácter de barra invertida hace el truco.

Hay otra característica de los operadores lógicos que es importante comprender. Digamos que tenemos dos expresiones separadas por un operador lógico:

*expr1* -operador *expr2*

En todos los casos, *siempre se realizará expr1*; sin embargo, el operador determinará si *se realiza expr2*. La tabla 17-5 muestra cómo funciona.

**Tabla 17-5: Lógica de búsqueda Y/O**

Resultados de <i>expr1</i>	Operador	<i>expr2</i> es...
Verdadero	-y	Siempre realizado
Falso	-y	Nunca se realizó
Verdadero	-o bien	Nunca se realizó
Falso	-o bien	Siempre realizado

¿Por qué sucede esto? Se hace para mejorar el rendimiento. Tomemos -y, por ejemplo. Sabemos que la expresión *expr1* -y *expr2* no puede ser verdadera si el resultado de *expr1* es falso, por lo que no tiene sentido realizar *expr2*. Del mismo modo, si tenemos la expresión *expr1* -o *expr2* y el resultado de *expr1* es verdadero, no tiene sentido realizar *expr2*, pues ya sabemos que la expresión *expr1* -o *expr2* es verdadera.

Bien, esto ayuda a que las cosas vayan más rápido. ¿Por qué es importante? Porque podemos confiar en este comportamiento para controlar cómo se realizan las acciones, como pronto veremos.

## Acciones

¡Vamos a hacer un poco de trabajo! Tener una lista de resultados de nuestro comando `find` es útil, pero lo que realmente queremos hacer es actuar sobre los elementos de la lista. Afortunadamente, `find` permite realizar acciones en función de los resultados de la búsqueda.

### Acciones predefinidas

Hay un conjunto de acciones predefinidas y varias formas de aplicar acciones definidas por el usuario. En primer lugar, echemos un vistazo a algunas de las acciones predefinidas en la Tabla 17-6.

**Tabla 17-6: Acciones de búsqueda predefinidas**

Acción	Descripción
-borrar	Elimine el archivo que coincide actualmente.
-ls	Realice el equivalente de <code>ls -dils</code> en el archivo coincidente. La salida se envía a la salida estándar.
-Impresión	Genere el nombre de ruta completo del archivo

coincidente al estándar salida. Esta es la acción predeterminada si no se especifica ninguna otra acción.

---

#### Cuadro 17-6 (continuación )

Acción	Descripción
-renunciar	Salga una vez que se haya hecho una coincidencia.

Al igual que con las pruebas, hay muchas más acciones. Consulte la página del comando man Find para obtener todos los detalles.

En nuestro primer ejemplo, hicimos lo siguiente:

---

#### encontrar~

---

Este comando produjo una lista de todos los archivos y subdirectorios contenidos en nuestro directorio de inicio. Generó una lista porque la acción -print está implícita si no se especifica ninguna otra acción. Por lo tanto, nuestro mandato también podría expresarse como

---

#### Buscar ~ -Imprimir

---

Podemos usar find para eliminar archivos que cumplan con ciertos criterios. Por ejemplo, para eliminar archivos que tienen la extensión de archivo .BAK (que a menudo se usa para diseñar archivos de copia de seguridad), podríamos usar este comando:

---

**find ~ -type f -name '\*.BAK' -borrar**

---

En este ejemplo, se buscan nombres de archivo en cada archivo en el directorio principal del usuario (y sus subdirectorios) que terminen en .BAK. Cuando se encuentran, se eliminan.

**Advertencia:** *No hace falta decir que debe tener mucho cuidado al usar*

*Siempre pruebe el comando primero sustituyendo la acción -print por -delete para confirmar los resultados de la búsqueda.*

Antes de continuar, echemos otro vistazo a cómo afectan los operadores lógicos a las acciones. Considere el siguiente comando:

---

**find ~ -type f -name '\*.BAK' -imprimir**

---

Como hemos visto, este comando buscará todos los archivos regulares (-type f) cuyo nombre termine en .BAK (-nombre '\*.BAK') y generará el nombre de ruta relativo de cada archivo coincidente a la salida estándar (-print). Sin embargo, la razón por la que el comando funciona de la manera en que lo hace está determinada por las relaciones lógicas entre cada una de las pruebas y acciones. Recuerde que, de forma predeterminada, existe una relación -and implícita entre cada prueba y acción. También podríamos expresar el comando de esta manera para que las relaciones lógicas sean más fáciles de ver:

---

**Encuentre ~ -type f -and -name '\*.BAK' -y -impresión**

---

Con nuestro comando completamente expresado, echemos un vistazo a

la Tabla 17-7 para ver cómo los operadores lógicos afectan su ejecución.

Tabla 17-7: Efecto de los operadores lógicos

Prueba/Acción	Se realiza cuando...
-Impresión	-tipo f y -nombre '*.BAK' son verdadero.
- Nombre *.Buck.	-tipo f es cierto.
-tipo f	Siempre se realiza, ya que es la primera prueba/acción de un -y la relación.

Dado que la relación lógica entre las pruebas y las acciones determina cuáles de ellas se realizan, podemos ver que el orden de las pruebas y acciones es importante. Por ejemplo, si tuviéramos que reordenar las pruebas y acciones para que la acción `-print` fuera la primera, el comando se comportaría de manera muy diferente:

---

**Encuentre ~ -print -and -type f -and -name '\*.BAK'**

---

Esta versión del comando imprimirá cada archivo (la acción `-print` siempre se evalúa como true) y, a continuación, probará el tipo de archivo y la extensión de archivo especificada.

### Acciones definidas por el usuario

Además de las acciones predefinidas, también podemos invocar comandos arbitrarios. La forma tradicional de hacer esto es con la acción `-exec`, así:

`-exec comando {} ;`

donde *command* es el nombre de un comando, {} es una representación simbólica del nombre de la ruta actual, y el punto y coma es un delimitador necesario que indica el final del comando. A continuación, se muestra un ejemplo del uso de `-exec` para actuar como la acción `-delete` descrita anteriormente:

---

`-exec rm '{}';`

---

De nuevo, dado que los caracteres de llave y punto y coma tienen un significado especial para el shell, deben estar entre comillas o con escape.

También es posible ejecutar una acción definida por el usuario de forma interactiva. Al utilizar la acción `-ok` en lugar de `-exec`, se pregunta al usuario antes de la ejecución de cada comando especificado:

---

```
find ~ -type f -name 'foo*' -ok ls -l '{}' ';' 
< ls ... /inicio/me/bin/foo > ? y
-rwxr-xr-x 1 yo      m 224 2011-10-29 18:44 /inicio/me/bin/foo
< ls ... /home/me/foo.txt > ? y
-rw-r--r-- 1 me      me  0 2012-09-19 12:53 /inicio/yo/foo.txt
```

---

En este ejemplo, buscamos archivos con nombres que comiencen con la cadena *foo* y ejecutamos el comando `ls -l` cada vez que se encuentra uno. El uso de la acción `-ok` pregunta al usuario antes de que se ejecute el comando `ls`.

## Mejora de la eficiencia

Cuando se utiliza la acción `-exec`, inicia una nueva instancia del comando especificado cada vez que se encuentra un archivo coincidente. Hay ocasiones en las que es posible que prefiramos combinar todos los resultados de la búsqueda e iniciar una sola instancia del comando. Por ejemplo, en lugar de ejecutar los comandos de esta manera,

```
ls -l fichero1  
ls -l fichero2
```

Es posible que prefiramos ejecutarlos de esta manera:

```
ls -l fichero1 fichero2
```

Aquí hacemos que el comando se ejecute solo una vez en lugar de varias veces. Hay dos formas de hacer esto: la forma tradicional, usando el comando externo `xargs`, y la forma alternativa, usando una nueva característica en `find itself`. Primero hablaremos de la forma alternativa.

Al cambiar el carácter de punto y coma final a un signo más, activamos la capacidad de `find` para combinar los resultados de la búsqueda en una lista de argumentos para una sola ejecución del comando deseado. Volviendo a nuestro ejemplo,

---

```
find ~ -type f -name 'foo*' -exec ls -l '{}' '+'  
-rwxr-xr-x 1 yo      m 224 2011-10-29 18:44 /inicio/me/bin/foo  
-rw-r--r-- 1 me      me   0 2012-09-19 12:53 /inicio/yo/foo.txt
```

---

ejecutará `ls` cada vez que se encuentre un archivo coincidente. Al cambiar el com- mand a

---

```
find ~ -type f -name 'foo*' -exec ls -l '{}' +  
-rwxr-xr-x 1 yo      m 224 2011-10-29 18:44 /inicio/me/bin/foo  
-rw-r--r-- 1 me      me   0 2012-09-19 12:53 /inicio/yo/foo.txt
```

---

Obtenemos los mismos resultados, pero el sistema tiene que ejecutar el comando `ls` solo una vez.

También podemos usar el comando `xargs` para obtener el mismo resultado. `xargs` acepta la entrada de la entrada estándar y la convierte en una lista de argumentos para un comando especificado. Con nuestro ejemplo, lo usaríamos así:

---

```
find ~ -type f -name 'foo*' -print | xargs ls -l  
-rwxr-xr-x 1 yo      m 224 2011-10-29 18:44 /inicio/me/bin/foo  
-rw-r--r-- 1 me      me   0 2012-09-19 12:53 /inicio/yo/foo.txt
```

---

Aquí vemos la salida del comando `find` canalizado a `xargs`, que, a su vez, construye una lista de argumentos para el comando `ls` y luego lo ejecuta.

**Nota:** Si bien el número de argumentos que se pueden colocar en una línea de comandos es bastante grande, no es ilimitado. Es posible crear comandos que sean demasiado largos para que el shell los acepte. Cuando una línea de comandos excede la longitud máxima admitida por el sistema, `xargs` ejecuta el comando especificado con el número máximo de argumentos posible y luego repite este proceso hasta que se agota la entrada estándar. Para ver el tamaño máximo de la línea de comandos, ejecute `xargs` con la opción `--show-limits`.

## TRATAR CON LOS NOMBRES DE ARCHIVO FU NNY

Los sistemas tipo Unix permiten espacios incrustados (¡e incluso nuevas líneas!) en los nombres de archivo. Esto causa problemas para programas como xargs que construyen listas de argumentos para otros programas. Un espacio incrustado se tratará como un delimitador, y el comando resultante interpretará cada palabra separada por espacios como un argumento separado. Para superar esto, find y xarg permiten el uso opcional de un carácter nulo como separador de argumentos. Un carácter nulo se define en ASCII como el carácter representado por el número cero (a diferencia, por ejemplo, del carácter de espacio, que se define en ASCII como el carácter representado por el número 32). El comando find proporciona la acción -print0, que produce una salida separada por null, y el comando xargs tiene la opción --null, que acepta entradas separadas por null. He aquí un ejemplo:

```
find ~ -iname "*.jpg" -print0 | xargs --null ls -l
```

Con esta técnica, podemos asegurarnos de que todos los archivos, incluso

### *Un regreso al patio de recreo*

Es hora de darle un uso (casi) práctico a la búsqueda. Primero, vamos a crear un terreno de juego con muchos subdirectorios y archivos:

---

```
[me@linuxbox ~]$ mkdir -p playground/dir-{00{1..9},0{10..99},100}  
[me@linuxbox ~]$ touch playground/dir-{00{1..9},0{10..99},100}/file-{A..Z}
```

---

¡Maravíllate con el poder de la línea de comandos! Con estas dos líneas, creamos un directorio playground que contiene 100 subdirectorios, cada uno de los cuales contiene 26 archivos vacíos. ¡Pruébalo con la GUI!

El método que empleamos para llevar a cabo esta magia involucró un comando familiar (mkdir), una expansión de caparazón exótica (llaves) y un nuevo comando, el tacto. Al combinar mkdir con la opción -p (que hace que mkdir cree los directorios principales de las rutas especificadas) con la expansión de llaves, pudimos crear 100 directorios.

El comando táctil se utiliza generalmente para establecer o actualizar los tiempos de modificación de los archivos. Sin embargo, si un argumento filename es el de un archivo inexistente, se crea un archivo vacío.

En nuestro patio de juegos, creamos 100 instancias de un archivo llamado *archivo-A*. Vamos a encontrarlos:

---

```
[me@linuxbox ~]$ find playground -type f -name 'file-A'
```

---

Tenga en cuenta que, a diferencia de ls, find no produce resultados en orden ordenado. Su orden está determinado por el diseño del dispositivo de almacenamiento. Podemos confirmar que realmente tenemos 100 instancias del archivo de esta manera:

---

```
[me@linuxbox ~]$ find playground -type f -name 'file-A' | wc -l  
100
```

---

A continuación, veamos cómo encontrar archivos en función de sus tiempos de modificación. Esto será útil a la hora de crear copias de seguridad u organizar archivos en orden cronológico. Para ello, primero crearemos un fichero de referencia con el que compararemos el tiempo de modificación:

---

```
[me@linuxbox ~]$ toque playground/marca de tiempo
```

---

Esto crea un archivo vacío llamado `timestamp` y establece su hora de modificación en la hora actual. Podemos verificar esto usando otro comando útil, `stat`, que es una especie de versión mejorada de `ls`. El comando `stat` revela todo lo que el sistema entiende sobre un archivo y sus atributos:

---

```
[me@linuxbox ~]$ stat playground/marca de tiempo
  Archivo: 'playground/timestamp'
  Tamaño: 0      Bloques: 0      Bloque de E/S: 4096 vacío regular
archivo Dispositivo: 803H/2051D Inodo: 14265061 Enlaces: 1
  Acceso: (0644/-rw-r--r--)Uid: ( 1001/ me)   Gid: ( 1001/ yo)
  Acceso: 2012-10-08 15:15:39.000000000 -0400
Modificar: 2012-10-08 15:15:39.000000000 -0400
Cambio: 2012-10-08 15:15:39.000000000 -0400
```

---

Si volvemos a tocar el fichero y luego lo examinamos con `stat`, veremos que las horas del fichero se han actualizado:

---

```
[me@linuxbox ~]$ toque playground/marca de tiempo
[me@linuxbox ~]$ stat playground/marca de tiempo
  Archivo: 'playground/timestamp'
  Tamaño: 0      Bloques: 0      Bloque de E/S: 4096 vacío regular
archivo Dispositivo: 803H/2051D Inodo: 14265061 Enlaces: 1
  Acceso: (0644/-rw-r--r--)Uid: ( 1001/ me)   Gid: ( 1001/ yo)
  Acceso: 2012-10-08 15:23:33.000000000 -0400
Modificar: 2012-10-08 15:23:33.000000000 -0400
Cambio: 2012-10-08 15:23:33.000000000 -0400
```

---

A continuación, usemos `find` para actualizar algunos de nuestros archivos de `playground`:

---

```
[me@linuxbox ~]$ find playground -type f -name 'file-B' -exec touch '{}' ';'
```

---

De este modo, se actualizan todos los archivos del `playground` que se denominan *archivo-B*. A continuación, usaremos `find` para identificar los archivos actualizados comparando todos los archivos con la marca de tiempo del archivo de referencia:

---

```
[me@linuxbox ~]$ find playground -type f -newer playground/timestamp
```

---

Los resultados contienen las 100 instancias del *archivo B*. Dado que realizamos un toque en todos los archivos en el patio de recreo que se llaman *archivo-B* después de que Marca de tiempo actualizada, ahora son "más nuevos" que la marca de tiempo y, por lo tanto, se pueden identificar con la prueba `-newer`.

Por último, volvamos a la prueba de permisos incorrectos que realizamos anteriormente y aplicémosla a `playground`:

```
[me@linuxbox ~]$ encontrar playground \(` -type f -not -perm 0600 \|) -or \(` -type d  
-no -perm 0700 \|)
```

---

Este comando enumera los 100 directorios y 2.600 archivos de *playground* (así como la *marca de tiempo* y el propio *playground*, para un total de 2.702) porque ninguno de ellos cumple con nuestra definición de "buenos permisos". Con nuestro conocimiento de operadores y acciones, podemos agregar acciones a este comando para aplicar nuevos permisos a los archivos y directorios en nuestro patio de recreo:

```
[me@linuxbox ~]$ find playground \(` -type f -not -perm 0600 -exec chmod 0600 '{}` ;` `) -or \(` -type d -not -perm 0700 -exec chmod 0700 '{}` ;` `)
```

En el día a día, puede que nos resulte más fácil emitir dos comandos, uno para los directorios y otro para los archivos, en lugar de este gran comando compuesto, pero es bueno saber que podemos hacerlo de esta manera. El punto importante aquí es comprender cómo los operadores y las acciones se pueden usar juntos para realizar tareas útiles.

## Opciones

Por último, tenemos las opciones. Las opciones se utilizan para controlar el ámbito de una búsqueda de búsqueda. Pueden incluirse con otras pruebas y acciones al crear expresiones de búsqueda. En la tabla 17-8 se enumeran las opciones más utilizadas.

Tabla 17-8: Buscar opciones

Opción	Descripción
-profundidad	Directo encontrar para procesar los archivos de un directorio antes que el propio directorio. Esta opción se aplica automáticamente cuando el - borrar se especifica la acción.
-Profundidad máxima <i>Niveles</i>	Establezca el número máximo de niveles que encontrar descenderá en un árbol de directorios al realizar pruebas y acciones.
-mindepth <i>Niveles</i>	Establezca el número mínimo de niveles que encontrar en un árbol de directorios antes de aplicar pruebas y acciones.
-montar	Directo encontrar no para atravesar directorios que están montados en otros sistemas de archivos.
-noleaf	Directo encontrar no para optimizar su búsqueda basándose en la suposición de que está buscando un sistema de archivos similar a Unix. Esto es necesario cuando se escanean sistemas de archivos y CD-ROM de DOS/Windows.

# 18

## ARCHIVADO Y COPIA DE SEGURIDAD

Una de las principales tareas del administrador de un sistema informático es mantener seguros los datos del sistema. Una forma de hacerlo es realizando copias de seguridad oportunas de los archivos del sistema. Incluso si no es un administrador de sistemas, a menudo es útil hacer copias de cosas y mover grandes colecciones de archivos de un lugar a otro y de un dispositivo a otro.

En este capítulo, veremos varios programas comunes que se utilizan para administrar colecciones de archivos. Existen los programas de compresión de archivos:

- `gzip`: comprime o expande archivos.
- `bzip2`: un compresor de archivos de clasificación de bloques. Los programas de archivo:
  - `tar`: utilidad de archivado de cintas.

- zip: empaqueta y comprime archivos.

y el programa de sincronización de archivos:

- rsync: sincronización remota de archivos y directorios.

## Compresión de archivos

A lo largo de la historia de la informática, ha habido una lucha por obtener la mayor cantidad de datos en el espacio más pequeño disponible, ya sea memoria, dispositivos de almacenamiento o ancho de banda de red. Muchos de los servicios de datos que hoy damos por sentados, como los reproductores de música portátiles, la televisión de alta definición o la Internet de banda ancha, deben su existencia a técnicas eficaces *de compresión de datos*.

La compresión de datos es el proceso de eliminar la *redundancia* de los datos. Consideremos un ejemplo imaginario. Digamos que tenemos un archivo de imagen completamente negro con las dimensiones de 100 píxeles por 100 píxeles. En términos de almacenamiento de datos (suponiendo 24 bits, o 3 bytes por píxel), la imagen ocupará 30.000 bytes de almacenamiento:  $100 \times 100 \times 3 = 30.000$ .

Una imagen de un solo color contiene datos totalmente redundantes. Si fuéramos inteligentes, podríamos codificar los datos de tal manera que simplemente describiríamos el hecho de que tenemos un bloque de 10.000 píxeles negros. Entonces, en lugar de almacenar un bloque de datos que contenga 30.000 ceros (el negro generalmente se representa en los archivos de imagen como cero), podríamos comprimir los datos en el número 30.000, seguido de un cero para representar nuestros datos. Este esquema de compresión de datos, llamado *codificación de longitud de ejecución*, es una de las técnicas de compresión más rudimentarias. Las técnicas actuales son mucho más avanzadas y complejas, pero el objetivo básico sigue siendo el mismo: deshacerse de los datos redundantes.

*Los algoritmos de compresión* (las técnicas matemáticas utilizadas para llevar a cabo la compresión) se dividen en dos categorías generales, *sin pérdidas* y *con pérdidas*. La compresión sin pérdidas conserva todos los datos contenidos en el original. Esto significa que cuando se restaura un archivo a partir de una versión comprimida, el archivo restaurado es exactamente el mismo que la versión original sin comprimir. La compresión con pérdida, por otro lado, elimina los datos a medida que se realiza la compresión, para permitir que se aplique más compresión. Cuando se restaura un archivo con pérdida, no coincide con la versión original; más bien, es una aproximación cercana. Ejemplos de compresión con pérdida son JPEG (para imágenes) y MP3 (para música). En nuestra discusión, nos ocuparemos exclusivamente de la compresión sin pérdidas, ya que la mayoría de los datos de los ordenadores no pueden tolerar ninguna pérdida de datos.

### **gzip: comprimir o expandir archivos**

El programa gzip se utiliza para comprimir uno o más archivos. Cuando se

ejecuta, reemplaza el archivo original con una versión comprimida del original. El programa gunzip correspondiente se utiliza para restaurar archivos comprimidos a su forma original, sin comprimir. He aquí un ejemplo:

---

```
[me@linuxbox ~]$ ls -l /etc > foo.txt
[me@linuxbox ~]$ ls -l foo.*
```

```

-Rw-r--r-- 1 yo      me   15738 2012-10-14 07:15 foo.txt
[me@linuxbox ~]$ gzip foo.txt
[me@linuxbox ~]$ ls -l foo.*
-Rw-r--r-- 1 yo      me    3230 2012-10-14 07:15 foo.txt.gz
[me@linuxbox ~]$ Gunzip foo.txt
[me@linuxbox ~]$ ls -l foo.*
-rw-r--r-- 1 me      me   15738 14/10/2012 07:15 foo.txt

```

---

En este ejemplo, creamos un archivo de texto llamado *foo.txt* a partir de una lista de directorios. A continuación, ejecutamos *gzip*, que reemplaza el archivo original con una versión comprimida llamada *foo.txt.gz*. En la lista de directorios de *foo.\**, vemos que el archivo original ha sido reemplazado por la versión comprimida y que la versión comprimida tiene aproximadamente una quinta parte del tamaño del original. También podemos ver que el archivo comprimido tiene los mismos permisos y marca de tiempo que el original.

A continuación, ejecutamos el programa *gunzip* para descomprimir el archivo. Despues, podemos ver que la versión comprimida del archivo ha sido reemplazada por la original, de nuevo con los permisos y la marca de tiempo conservados.

*gzip* tiene muchas opciones. En la Tabla 18-1 se enumeran algunos.

**Tabla 18-1: Opciones de gzip**

Opción	Descripción
-c	Escriba la salida en la salida estándar y conserve los archivos originales. También puede ser especificado con --stdout y --a-stdout.
-d	Descomprimir. Esto provoca <i>gzip</i> para actuar como <i>Gunzip</i> . También se puede especificar con --descomprimir o --descomprimir.
-f	Forzar la compresión incluso si se trata de una versión comprimida del archivo original ya existe. También se puede especificar con --fuerza.
-h	Mostrar información de uso. También se puede especificar con --Ayuda.
-l	Enumere las estadísticas de compresión de cada archivo comprimido. También se puede especificar con --lista.
-r	Si uno o más argumentos de la línea de comandos son directorios, comprima de forma recursiva los archivos contenidos en ellos. También se puede especificar con --recursivo.
-t	Probar la integridad de un archivo comprimido. También se puede especificar con --prueba.
-v	Muestra mensajes detallados mientras se comprime. También se puede especificar con --verboso.
-número	Establezca la cantidad de compresión. <i>número</i> es un número

entero en el intervalo de 1 (más rápido, menor compresión) a 9 (más lento, mayor compresión). Los valores 1 y 9 también pueden expresarse como: --rápido y --mejor respectivamente. El valor predeterminado es 6.

Veamos de nuevo nuestro ejemplo anterior:

```
[me@linuxbox ~]$ gzip foo.txt  
[me@linuxbox ~]$ gzip -tv foo.txt.gz  
foo.txt.gz:          De acuerdo  
[me@linuxbox ~]$ gzip -d foo.txt.gz
```

Aquí, reemplazamos el archivo *foo.txt* con una versión comprimida llamada *foo.txt.gz*. A continuación, probamos la integridad de la versión comprimida, utilizando el método

Opciones *-t* y *-v*. Finalmente, descomprimimos el archivo a su forma original. *gzip* también se puede utilizar de formas interesantes a través de la entrada y salida estándar:

```
[me@linuxbox ~]$ ls -l /etc | gzip > foo.txt.gz
```

Este comando crea una versión comprimida de una lista de directorios. El programa *gunzip*, que descomprime los archivos *gzip*, asume que el archivo

Los nombres terminan en la *.gz* de extensión, por lo que no es necesario especificarla, siempre y cuando el nombre especificado no esté en conflicto con un archivo sin comprimir existente:

```
[me@linuxbox ~]$ gunzip foo.txt
```

Si nuestro objetivo fuera solo ver el contenido de un archivo de texto comprimido, podríamos hacer lo siguiente:

```
[me@linuxbox ~]$ gunzip -c foo.txt | menos
```

Alternativamente, un programa suministrado con *gzip*, llamado *zcat*, es equivalente a *gunzip* con la opción *-c*. Se puede usar como el comando *cat* en *gzip* - archivos comprimidos:

```
[me@linuxbox ~]$ zcat foo.txt.gz | menos
```

**Nota:** También hay un programa *zless*. Realiza la misma función que la tubería anterior.

### ***bzip2: mayor compresión a costa de la velocidad***

El programa *bzip2*, de Julian Seward, es similar a *gzip* pero utiliza un algoritmo de compresión diferente, que logra niveles más altos de compresión a costa de la velocidad de compresión. En la mayoría de los aspectos, funciona de la misma manera que *gzip*. Un archivo comprimido con *bzip2* se denota con la extensión *.bz2*:

```
[me@linuxbox ~]$ ls -l /etc > foo.txt  
[me@linuxbox ~]$ ls -l foo.txt  
-Rw-r--r-- 1 yo      me      15738 17/10/2012 13:51  
foo.txt [me@linuxbox ~]$ bzip2 foo.txt  
[me@linuxbox ~]$ ls -l foo.txt.bz2  
-Rw-r--r-- 1 yo      me      2792 17/10/2012 13:51 foo.txt.bz2  
[me@linuxbox ~]$ bunzip2 foo.txt.bz2
```

Como podemos ver, bzip2 se puede usar de la misma manera que gzip. Todas las opciones (excepto `-r`) que discutimos para gzip también son compatibles con bzip2. Nótese, sin embargo, que la opción de nivel de compresión (`-number`) tiene un significado algo diferente al de bzip2. bzip2 viene con bunzip2 y bzcat para descomprimir archivos.

Bzip2 también viene con el programa bzip2recover, que intentará recuperar archivos `.bz2` dañados.

### **NO SEAS COMPRESIVO COMPULSIVO**

De vez en cuando veo a personas que intentan comprimir un archivo que ya ha sido comprimido con un algoritmo de compresión efectivo, haciendo algo como esto:

```
$ gzip picture.jpg
```

No lo hagas. ¡Probablemente solo estés perdiendo tiempo y espacio! Si aplica compresión a un archivo que ya está comprimido, en realidad terminará con un archivo más grande. Esto se debe a que todas las técnicas de compresión implican una sobrecarga que se agrega al archivo para describir la compresión. Si intenta comprimir un archivo que ya no contiene información redundante, la compresión no generará ningún ahorro para compensar la sobrecarga adicional.

## **Archivado de archivos**

Una tarea común de administración de archivos que se utiliza junto con la compresión es *el archivado*. El archivado es el proceso de recopilar muchos archivos y agruparlos en un solo archivo grande. El archivado a menudo se realiza como parte de las copias de seguridad del sistema. También se utiliza cuando los datos antiguos se mueven de un sistema a algún tipo de almacenamiento a largo plazo.

### ***tar: utilidad de archivado de cintas***

En el mundo del software, similar a Unix, el programa tar es la herramienta clásica para archivar archivos. Su nombre, abreviatura de *archivo de cinta*, revela sus raíces como herramienta para hacer cintas de copia de seguridad. Si bien todavía se usa para esa tarea tradicional, es igualmente hábil en otros dispositivos de almacenamiento. A menudo vemos nombres de archivo que terminan con la extensión `.tar` o `.tgz`, que indican un archivo tar "simple" y un archivo comprimido con gzip, respectivamente. Un archivo tar puede consistir en un grupo de archivos separados, una o más jerarquías de directorios o una combinación de ambos. La sintaxis del comando funciona de la siguiente manera:

```
tar mode[opciones] nombre de la ruta...
```

donde *mode* es uno de los modos de funcionamiento que se muestran en la Tabla 18-2 (aquí solo se muestra una lista parcial; consulte la página del manual tar para obtener una lista completa).

**Tabla 18-2: Modos de alquitrán**

Modo	Descripción
c	Cree un archivo a partir de una lista de archivos y/o directorios.
x	Extraer un archivo.
r	Anexe los nombres de ruta especificados al final de un archivo.
t	Enumerar el contenido de un archivo.

tar usa una forma un poco extraña de expresar opciones, por lo que necesitaremos algunos ejemplos para mostrar cómo funciona. Primero, recreemos nuestro patio de recreo del capítulo anterior:

```
[me@linuxbox ~]$ mkdir -p playground/dir-{00{1..9},0{10..99},100}  
[me@linuxbox ~]$ touch playground/dir-{00{1..9},0{10..99},100}/file-{A..Z}
```

A continuación, vamos a crear un archivo tar de todo el patio de recreo:

```
[me@linuxbox ~]$ tar cf playground.tar patio de recreo
```

Este comando crea un archivo tar llamado *playground.tar*, que contiene toda la jerarquía de directorios del patio de recreo. Podemos ver que el modo y la opción f, que se usa para especificar el nombre del archivo tar, pueden estar unidos y no requieren un guión inicial. Tenga en cuenta, sin embargo, que el modo siempre debe especificarse primero, antes de cualquier otra opción.

Para listar el contenido del archivo, podemos hacer lo siguiente:

```
[me@linuxbox ~]$ tar tf playground.tar
```

Para una lista más detallada, podemos agregar la opción v (verbose):

```
[me@linuxbox ~]$ tar tvf playground.tar
```

Ahora, extraigamos el patio de recreo en una nueva ubicación. Para ello, crearemos un nuevo directorio llamado *foo*, cambiando el directorio y extrayendo el archivo tar:

```
[me@linuxbox ~]$ mkdir foo  
[me@linuxbox ~]$ CD foo  
[me@linuxbox foo]$ tar xf .. /playground.tar  
[me@linuxbox foo]$ ls  
patio de recreo
```

Si examinamos el contenido de *~/foo/playground*, vemos que el archivo se instaló correctamente, creando una reproducción precisa de los archivos originales. Sin embargo, hay una advertencia: a menos que esté operando como el superusuario, los archivos y directorios extraídos de los archivos asumen la propiedad del usuario que realiza la restauración, en lugar del propietario original.

Otro comportamiento interesante de tar es la forma en que maneja los nombres de ruta en los archivos. El valor predeterminado para los nombres de ruta es relativo, en lugar de absoluto. tar hace esto simplemente eliminando cualquier barra diagonal inicial del nombre de la ruta al crear el archivo. Para demostrarlo, volveremos a crear nuestro archivo, esta vez especificando un nombre de ruta absoluto:

---

```
[me@linuxbox foo]$ cd  
[me@linuxbox ~]$ tar cf playground2.tar ~/playground
```

---

Recuerde, *~/playground* se expandirá a */home/me/playground* cuando presionemos la tecla ENTER , por lo que obtendremos un nombre de ruta absoluto para nuestra demostración. A continuación, extraeremos el archivo como antes y observaremos qué sucede:

---

```
[me@linuxbox ~]$ CD foo  
[me@linuxbox foo]$ tar xf .. /playground2.tar  
[me@linuxbox foo]$ ls  
hogar  parque infantil  
[me@linuxbox foo]$ ls hogar  
me  
[me@linuxbox foo]$ ls inicio/yo  
patio de recreo
```

---

Aquí podemos ver que cuando extrajimos nuestro segundo archivo, volvió a crear el directorio *home/me/playground* en relación con nuestro directorio de trabajo actual,

*~/foo*, no relativo al directorio raíz, como habría sido el caso con un nombre de ruta absoluto. Esto puede parecer una forma extraña de que funcione, pero en realidad es más útil de esta manera, ya que nos permite extraer archivos a cualquier ubicación en lugar de vernos obligados a extraerlos a sus ubicaciones originales.

Repetir el ejercicio con la inclusión de la opción detallada (v) dará una imagen más clara de lo que está sucediendo.

Consideremos un ejemplo hipotético, pero práctico, de alquitrán en acción. Imaginemos que queremos copiar el directorio de inicio y su contenido de un sistema a otro y tenemos un disco duro USB grande que podemos usar para la transferencia. En nuestro moderno sistema Linux, la unidad se monta "automáticamente" en el directorio */media*. Imaginemos también que el disco tiene un nombre de volumen de *BigDisk* cuando lo conectamos. Para hacer el archivo tar, podemos hacer lo siguiente:

---

```
[me@linuxbox ~]$ sudo tar cf /media/BigDisk/home.tar /home
```

---

Una vez escrito el archivo tar, desmontamos la unidad y la conectamos a la segunda computadora. De nuevo, se monta en */media/BigDisk*. Para extraer el archivo, hacemos lo siguiente:

---

```
[me@linuxbox2 ~]$ cd /  
[me@linuxbox2 /]$ sudo tar xf /media/BigDisk/home.tar
```

---

Lo que es importante ver aquí es que primero debemos cambiar el directorio a / para que la extracción sea relativa al directorio raíz, ya que

todos los nombres de ruta dentro del archivo son relativos.

Al extraer un archivo, es posible limitar lo que se extrae. Por ejemplo, si quisiéramos extraer un solo archivo de un archivo, se podría hacer de la siguiente manera:

---

#### Archivo tar XF.tar *nombre de la ruta*

---

Al agregar el nombre de la *ruta final* al comando, nos aseguramos de que tar restaure solo el archivo especificado. Se pueden especificar varios nombres de ruta. Tenga en cuenta que el nombre de ruta debe ser el nombre de ruta relativo completo y exacto almacenado en el archivo. Al especificar nombres de ruta, normalmente no se admiten caracteres comodín; sin embargo, la versión GNU de tar (que es la versión que se encuentra con mayor frecuencia en las distribuciones de Linux) los admite con la opción `--wildcards`. A continuación, se muestra un ejemplo utilizando nuestro *archivo playground.tar* anterior:

---

```
[me@linuxbox ~]$ CD foo  
[me@linuxbox foo]$ tar xf .. /playground2.tar --wildcards 'home/me/playground/  
dir-*/*file-A'
```

---

Este comando extraerá solo los archivos que coincidan con el nombre de ruta especificado, incluido el comodín `dir-*`.

tar se usa a menudo junto con find para producir archivos. En este ejemplo, usaremos find para producir un conjunto de archivos para incluir en un archivo:

---

```
[me@linuxbox ~]$ find playground -name 'file-A' -exec tar rf playground.tar '{  
3}' '+'
```

---

Aquí usamos find para hacer coincidir todos los archivos en el *playground* llamado *file-A* y luego, usando la acción `-exec`, invitamos tar en el modo de adición (`r`) para agregar los archivos coincidentes al *playground.tar* de archivo.

El uso de tar con find es una buena manera de crear *copias de seguridad incrementales* de un árbol directo o de un sistema completo. Al usar buscar para hacer coincidir archivos más recientes que un archivo de marca de tiempo, podríamos crear un archivo que contenga solo archivos más nuevos que el último archivo, suponiendo que el archivo de marca de tiempo se actualiza justo después de crear cada archivo.

tar también puede hacer uso de entrada y salida estándar. He aquí un ejemplo comprensible:

---

```
[me@linuxbox foo]$ cd  
[me@linuxbox ~]$ find playground -name 'file-A' | tar cf - --files-from=- | gzip  
> patio de juegos.tgz
```

---

En este ejemplo, usamos el programa find para producir una lista de archivos coincidentes y los canalizamos a tar. Si se especifica el nombre de archivo `,` se considera que significa entrada o salida estándar, según sea necesario. (Por cierto, esta convención de usar `-` para representar la entrada/salida estándar también es utilizado por varios otros programas). La opción `--files-from` (que también puede especificarse como `-T`) hace que tar lea su lista de nombres de ruta de un archivo en lugar de la línea de

comandos. Por último, el archivo producido por tar se canaliza a gzip para crear el archivo comprimido *playground.tgz*. La extensión *.tgz* es la extensión convencional que se da a los archivos tar comprimidos con gzip. La extensión *.tar.gz* también se usa a veces.

Mientras que nosotros usábamos el programa gzip externamente para producir nuestro archivo comprimido, las versiones modernas de GNU tar soportan la compresión gzip y bzip2 directamente con el uso de las opciones z y j, respectivamente. Usando nuestro ejemplo anterior como base, podemos simplificarlo de esta manera:

---

```
[me@linuxbox ~]$ find playground -name 'file-A' | tar czf playground.tgz -T -
```

---

Si hubiéramos querido crear un archivo comprimido con bzip2 en su lugar, podríamos haber hecho esto:

---

```
[me@linuxbox ~]$ find playground -name 'file-A' | tar cjf playground.tbz -T -
```

---

Simplemente cambiando la opción de compresión de z a j (y cambiando la extensión del archivo de salida a .tbz para indicar un archivo comprimido bzip2), habilitamos la compresión bzip2.

Otro uso interesante de la entrada y salida estándar con el controlador tar consiste en la transferencia de archivos entre sistemas a través de una red. Imaginemos que tenemos dos máquinas ejecutando un sistema tipo Unix equipado con tar y ssh. En tal escenario, podríamos transferir un directorio de un sistema remoto (llamado *remote-sys* para este ejemplo) a nuestro sistema local:

---

```
[me@linuxbox ~]$ mkdir remote-stuff  
[me@linuxbox ~]$ cd remote-stuff  
[me@linuxbox remote-stuff]$ ssh remote-sys 'tar cf - Documentos' | tar xf -  
Contraseña de me@remote-sys:  
[me@linuxbox cosas remotas]$ ls  
Documentos
```

---

Aquí pudimos copiar un directorio llamado *Documents* del sistema remoto *remote-sys* a un directorio dentro del directorio llamado *remote-stuff* en el sistema local. ¿Cómo lo hicimos? Primero, lanzamos el programa tar en el sistema remoto usando ssh. Recordará que ssh nos permite ejecutar un programa de forma remota en una computadora en red y "ver" los resultados en el sistema local: la salida estándar producida en el sistema remoto se envía al sistema local para su visualización. Podemos aprovechar esto haciendo que tar cree un archivo (el modo c) y lo envíe a la salida estándar, en lugar de a un archivo (la opción f con el argumento dash), transportando así el archivo a través del túnel cifrado proporcionado por ssh al sistema local. En el sistema local, ejecutamos tar y hacemos que expanda un archivo (el modo x) suministrado desde la entrada estándar (de nuevo, la opción f con el argumento dash).

### ***zip: empaquetar y comprimir archivos***

El programa zip es tanto una herramienta de compresión como un archivador. El formato de archivo utilizado por el programa es familiar para los usuarios de Windows, ya que lee y escribe *archivos.zip*. En Linux, sin embargo, gzip es el programa de compresión predominante, con bzip2 en segundo lugar. Los usuarios de Linux utilizan principalmente zip para intercambiar archivos con sistemas Windows, en lugar de realizar la

compresión y el archivado.

En su uso más básico, zip se invoca de la siguiente manera:

**opciones zip archivo zip...**

Por ejemplo, para hacer un archivo zip de nuestro patio de recreo, haríamos lo siguiente:

---

```
[me@linuxbox ~]$ zip -r playground.zip patio de recreo
```

---

A menos que incluyamos la opción `-r` para la recursividad, solo se almacena el directorio `playground` (pero ninguno de sus contenidos). Aunque la adición de la extensión `.zip` es automática, incluiremos la extensión del archivo para mayor claridad.

Durante la creación del archivo zip, zip normalmente mostrará una serie de mensajes como este:

---

```
adding: playground/dir-020/file-Z (almacenado 0%)
añadiendo: playground/dir-020/file-Y
(almacenado 0%) añadiendo: playground/dir-020/file-X (almacenado 0%) añadiendo:
playground/dir-087/ (almacenado 0%)
añadiendo: playground/dir-087/file-S
(almacenado 0%)
```

---

Estos mensajes muestran el estado de cada archivo agregado al archivo. zip agregará archivos al archivo utilizando uno de dos métodos de almacenamiento: O bien "almacenará" un archivo sin compresión, como se muestra aquí, o bien "desinflará" el archivo, lo que realiza la compresión. El valor numérico que se muestra después del método de almacenamiento indica la cantidad de compresión lograda. Dado que nuestro patio de recreo contiene solo archivos vacíos, no se realiza ninguna compresión en su contenido.

Extraer el contenido de un archivo zip es sencillo cuando se utiliza el método Descomprimir programa:

---

```
[me@linuxbox ~]$ CD foo
[me@linuxbox foo]$ descomprimir .. /playground.zip
```

---

Una cosa a tener en cuenta sobre zip (a diferencia de tar) es que si se especifica un archivo existente, se actualiza en lugar de reemplazarse. Esto significa que se conserva el archivo existente, pero se agregan nuevos archivos y se reemplazan los archivos correspondientes.

Los archivos pueden ser listados y extraídos selectivamente de un archivo zip especificándolos para descomprimir:

---

```
[me@linuxbox ~]$ descomprimir -l playground.zip playground/dir-087/file-Z
Archivo: ./playground.zip Longitud
```

Fecha	Hora	Nombre
-----	-----	-----
0	10-05-12 09:25	patio de juegos/dir-087/file-Z
-----	-----	-----
0		1 archivo

```
[me@linuxbox ~]$ CD foo
```

```
[me@linuxbox foo]$ descomprimir .. /playground.zip parque infantil/dir-087/file-Z
Archivo:... /playground.zip
```

reemplace playground/dir-087/file-Z? [y]es, [n]o, [A]ll, [N]one, [r]ename: y  
extracción: playground/dir-087/file-Z

---

El uso de la opción `-l` hace que `unzip` simplemente enumere el contenido del archivo sin extraer el archivo. Si no se especifica ningún archivo, `unzip` enumerará todos los archivos del archivo. Se puede agregar la opción `-v` para aumentar el nivel de detalle de la lista. Tenga en cuenta que cuando la extracción de archivos entra en conflicto con un archivo existente, se le pregunta al usuario antes de que se reemplace el archivo.

Al igual que `tar`, `zip` puede hacer uso de entrada y salida estándar, aunque su implementación es algo menos útil. Es posible canalizar una lista de nombres de archivos para comprimir a través de la opción `-@`:

---

```
[me@linuxbox foo]$ cd  
[me@linuxbox ~]$ find playground -name "file-A" | zip -@ file-A.zip
```

---

Aquí usamos `find` para generar una lista de archivos que coinciden con el nombre de prueba "file-A" y luego canalizamos la lista a `zip`, que crea el archivo `file-A.zip` que contiene los archivos seleccionados.

`zip` también admite la escritura de su salida en una salida estándar, pero su uso es limitado porque muy pocos programas pueden hacer uso de la salida. Desafortunadamente, el programa de descompresión no acepta entradas estándar. Esto evita que `zip` y `unzip` se usen juntos para realizar copias de archivos de red como `tar`.

Sin embargo, `zip` puede aceptar entradas estándar, por lo que se puede utilizar para comprimir la salida de otros programas:

---

```
[me@linuxbox ~]$ ls -l /etc/ | zip ls-etc.zip -  
Añadiendo: - (Desinflado 80%)
```

---

En este ejemplo, canalizamos la salida de `ls` a `zip`. Al igual que `tar`, `zip` interpreta el guión final como "usar entrada estándar para el archivo de entrada".

El programa `unzip` permite que su salida se envíe a la salida estándar cuando se especifica la opción `-p` (para tubería):

---

```
[me@linuxbox ~]$ descomprimir -p ls-etc.zip | menos
```

---

Tocamos algunas de las cosas básicas que puede hacer comprimir y descomprimir. Ambos tienen muchas opciones que se suman a su flexibilidad, aunque algunas son plataformas específicas de otros sistemas. Las páginas del manual para `zip` y `unzip` son bastante buenas y contienen ejemplos útiles.

## Sincronización de archivos y directorios

Una estrategia común para mantener una copia de seguridad de un sistema consiste en mantener uno o más directorios sincronizados con otro directorio (o directrices) ubicado en el sistema local (generalmente un dispositivo de almacenamiento extraíble de algún tipo) o en un sistema remoto.

Podríamos, por ejemplo, tener una copia local de un sitio web en desarrollo y sincronizarla de vez en cuando con la copia "en vivo" en un servidor web remoto.

## ***rsync: sincronización remota de archivos y directorios***

En el mundo tipo Unix, la herramienta preferida para esta tarea es **rsync**. Este programa puede sincronizar directorios locales y remotos mediante el uso del *protocolo de actualización remota rsync*, que permite a **rsync** detectar rápidamente las diferencias entre dos directorios y realizar la cantidad mínima de copia necesaria para sincronizarlos. Esto hace que **rsync** sea muy rápido y económico de usar, en comparación con otros tipos de programas de copia.

**rsync** se invoca de la siguiente manera:

**Destino de origen de las opciones de rsync**

donde el *origen* y el *destino* son cada uno de los siguientes:

- Un archivo o directorio local
- Un archivo o directorio remoto en forma de *[user@Jhost:path]*
- Un servidor rsync remoto especificado con un URI de *rsync://[user@Jhost[:port]/path]*

Tenga en cuenta que el origen o el destino deben ser un archivo local. No se admite la copia de remoto a remoto.

Probemos **rsync** en algunos archivos locales. Primero, limpiemos nuestro *directorio foo*:

---

```
[me@linuxbox ~]$ rm -rf foo/*
```

---

A continuación, sincronizaremos el directorio *playground* con una copia correspondiente en *foo*:

---

```
[me@linuxbox ~]$ rsync -av playground foo
```

---

Hemos incluido la opción **-a** (para archivar, provoca la recursividad y la conservación de los atributos del archivo) y la opción **-v** (salida detallada) para hacer

Un *espejo* del directorio de *playground* dentro de *FOO*. Mientras se ejecuta el comando, veremos una lista de los archivos y directorios que se están copiando. Al final, veremos un mensaje de resumen como este, indicando la cantidad de copias realizadas:

---

```
Enviado 135759 bytes recibidos: 57870 bytes: 387258,00 bytes/seg,
el tamaño total es 3230, la velocidad es 0,02
```

---

Si volvemos a ejecutar el comando, veremos un resultado diferente:

---

```
[me@linuxbox ~]$ rsync -av playground foo
lista de archivos de construcción ... hecho
```

---

```
enviado: 22635 bytes recibidos, 20 bytes 45310.00
bytes/seg, el tamaño total es 3230, la aceleración es
0.14
```

---

Observe que no había ninguna lista de archivos. Esto se debe a que **rsync** detectó que no había diferencias entre *~/playground* y *~/foo/playground*, y por lo tanto no necesitó copiar nada. Si modificamos un fichero en *playground* y

volvemos a ejecutar rsync, vemos que rsync ha detectado el cambio y ha copiado solo el fichero actualizado.

---

```
[me@linuxbox ~]$ touch playground/dir-099/file-Z
[me@linuxbox ~]$ rsync -av playground foo
building file list ... hecho
patio de juegos/dir-099/file-Z
enviado: 22685 bytes recibidos, 42 bytes, 45454.00
bytes/seg, el tamaño total es 3230, la velocidad es
0.14
```

---

Como ejemplo práctico, consideremos el disco duro externo imaginario que usamos anteriormente con tar. Si conectamos la unidad a nuestro sistema y, una vez más, está montada en */media/BigDisk*, podemos realizar una copia de seguridad útil del sistema creando primero un directorio llamado */backup* en la unidad externa y luego usando rsync para copiar las cosas más importantes de nuestro sistema a la unidad externa:

---

```
[me@linuxbox ~]$ mkdir /media/BigDisk/copia de seguridad
[me@linuxbox ~]$ sudo rsync -av --delete /etc /home /usr/local /media/BigDisk/
backup
```

---

En este ejemplo, copiamos los directorios */etc*, */home* y */usr/local* de nuestro sistema a nuestro dispositivo de almacenamiento imaginario. Incluimos la opción *--delete* para eliminar archivos que pueden haber existido en el dispositivo de copia de seguridad que ya no existían en el dispositivo de origen (esto es irrelevante la primera vez que hacemos una copia de seguridad, pero será útil en copias posteriores). Repetir el procedimiento de conectar la unidad externa y ejecutar este comando rsync sería una forma útil (aunque no ideal) de mantener una copia de seguridad de un sistema pequeño. Por supuesto, un alias también sería útil aquí. Podríamos crear un alias y agregarlo a nuestro archivo *.bashrc* para proporcionar esta característica:

---

```
alias backup='sudo rsync -av --delete /etc /home /usr/local /media/BigDisk/bac
kup'
```

---

Ahora todo lo que tenemos que hacer es conectar nuestra unidad externa y ejecutar la copia de seguridad comando para hacer el trabajo.

### ***Uso de rsync a través de una red***

Una de las verdaderas bellezas de rsync es que se puede usar para copiar archivos a través de una red. Después de todo, la *r* en rsync significa *remoto*. La copia remota se puede realizar de dos maneras.

La primera forma es con otro sistema que tenga rsync instalado, junto con un programa de shell remoto como ssh. Supongamos que tenemos otro sistema en nuestra red local con mucho espacio disponible en el disco duro y queremos realizar nuestra operación de copia de seguridad utilizando el sistema remoto en lugar de una unidad externa. Suponiendo que ya tenía un directorio llamado */backup* donde podríamos entregar nuestros archivos, podríamos hacer esto:

---

```
[me@linuxbox ~]$ sudo rsync -av --delete --rsh=ssh /etc /home /usr/local remote-
sys:/backup
```

---

Hemos realizado dos cambios en nuestro comando para facilitar la copia de red. Primero, añadimos la opción `--rsh=ssh`, que indica a `rsync` que use el programa `ssh` como su shell remoto. De esta manera, pudimos utilizar un túnel cifrado con SSH para transferir de forma segura los datos del sistema local al host remoto. En segundo lugar, especificamos el host remoto prefijando su nombre (en este caso, el host remoto se denomina *remote-sys*) al nombre de la ruta de destino.

La segunda forma en que se puede usar `rsync` para sincronizar archivos a través de una red es mediante el uso de un *servidor rysnc*. `rsync` se puede configurar para que se ejecute como un demonio y escuche las solicitudes entrantes de sincronización. Esto se hace a menudo para permitir la duplicación de un sistema remoto. Por ejemplo, Red Hat Software mantiene un gran repositorio de paquetes de software en desarrollo para su distribución Fedora. Es útil que los evaluadores de software reflejen esta colección durante la fase de prueba del ciclo de lanzamiento de la distribución. Dado que los archivos del repositorio cambian con frecuencia (a menudo más de una vez al día), es conveniente mantener una réplica local mediante sincronización periódica, en lugar de mediante la copia masiva del repositorio. Uno de estos repositorios se mantiene en Georgia Tech; podríamos duplicarlo usando nuestra copia local de `rsync` y el servidor `rsync` de Georgia Tech de la siguiente manera:

---

```
[me@linuxbox -]$ mkdir fedora-devel  
[me@linuxbox -]$ rsync -av -delete rsync://rsync.gtlb.gatech.edu/fedora-  
linux-core/development/i386/os fedora-devel
```

---

En este ejemplo, usamos el URI del servidor `rsync` remoto, que consta de un protocolo (`rsync://`), seguido del nombre de host remoto (`rsync.gtlb.gatech.edu`), seguido del nombre de la ruta del repositorio.

# 19

## EXPRESIONES REGULARES

En los próximos capítulos, vamos a ver las herramientas utilizadas para manipular el texto. Como hemos visto, los datos de texto juegan un papel importante en todos los sistemas tipo Unix, como Linux. Pero antes de que podamos apreciar completamente todas las características que ofrecen estas herramientas, tenemos que examinar una tecnología que se asocia con frecuencia con los usos más sofisticados de estas herramientas: las expresiones regulares.

A medida que hemos navegado por las muchas características y facilidades que ofrece la línea de com- m, hemos encontrado algunas características y comandos de shell verdaderamente arcanos, como la expansión y las citas de shell, los atajos de teclado y el historial de comandos, sin mencionar el editor vi. Las expresiones regulares continúan esta "tradición" y pueden ser (posiblemente) la característica más arcana de todas ellas. Esto no quiere decir que el tiempo que se necesita para aprender sobre ellos no valga la pena. Todo lo contrario. Una buena comprensión nos permitirá realizar hazañas asombrosas, aunque su valor total puede no ser evidente de inmediato.



## ¿Qué son las expresiones regulares?

En pocas palabras, *las expresiones regulares* son notaciones simbólicas que se utilizan para identificar patrones en el texto. De alguna manera, se asemejan al método comodín del shell para hacer coincidir los nombres de archivos y rutas, pero a una escala mucho mayor. Las expresiones regulares son soportadas por muchas herramientas de línea de comandos y por la mayoría de los lenguajes de programación para facilitar la solución de problemas de manipulación de texto. Sin embargo, para confundir aún más las cosas, no todas las expresiones regulares son iguales; Varían ligeramente de una herramienta a otra y de un lenguaje de programación a otro. Para nuestra discusión, nos limitaremos a las expresiones regulares como se describe en el estándar POSIX (que cubrirá la mayoría de las herramientas de línea de comandos), a diferencia de muchos lenguajes de programación (sobre todo Perl), que usan conjuntos de notaciones un poco más grandes y ricos.

## grep: búsqueda a través de texto

El programa principal que usaremos para trabajar con expresiones regulares es nuestro viejo amigo, grep. El nombre *grep* en realidad se deriva de la frase *impresión de expresiones regulares globales*, por lo que podemos ver que grep tiene algo que ver con las expresiones regulares. En esencia, grep busca en los archivos de texto la aparición de una expresión regular especificada y genera cualquier línea que contenga una coincidencia con la salida estándar.

Hasta ahora, hemos usado grep con cadenas fijas, así:

---

```
[me@linuxbox ~]$ ls /usr/bin | grep zip
```

---

Esto listará todos los archivos en el directorio */usr/bin* cuyos nombres contienen la subcadena *zip*.

El programa grep acepta opciones y argumentos de la siguiente manera:

```
grep [opciones] regex [archivo...]
```

donde *regex* es una expresión regular.

En la tabla 19-1 se enumeran las opciones grep más utilizadas.

Tabla 19-1: Opciones grep

Opción	Descripción
-I	Ignorar mayúsculas y minúsculas. No distinga entre mayúsculas y minúsculas Caracteres. También se puede especificar --ignorar-caso.
-v	Invertir coincidencia. Normalmente Grep Imprime líneas que contienen una coincidencia. Esta opción hace que Grep para imprimir todas las líneas que no contengan una coincidencia. También se puede especificar --invertir-coincidencia.

-c

Imprima el número de coincidencias (o no coincidencias si el -v también se especifica) en lugar de las propias líneas.  
También se puede especificar --contar.

---

Cuadro 19-1 (*continuación* )

Opción	Descripción
-l	Imprima el nombre de cada archivo que contenga una coincidencia en lugar de las propias líneas. También se puede especificar --archivos-con-coincidencias.
-L	Al igual que el -l , pero imprime solo los nombres de los archivos que no contienen coincidencias. También se puede especificar --archivos-sin-coincidencia.
-n	Prefije a cada línea coincidente el número de la línea dentro del archivo. También se puede especificar --número-línea.
-h	En el caso de las búsquedas de varios archivos, suprime la salida de los nombres de archivo. También se puede especificar --nombre-de-archivo.

Para explorar más a fondo grep, vamos a crear algunos archivos de texto para buscar:

```
[me@linuxbox ~]$ ls /bin > dirlist-bin.txt  
[me@linuxbox ~]$ ls /usr/bin > dirlist/usr-bin.txt  
[me@linuxbox ~]$ ls /sbin > dirlist-sbin.txt  
[me@linuxbox ~]$ ls /usr/sbin > dirlist/usr-sbin.txt  
[me@linuxbox ~]$ ls dirlist*.txt  
dirlist-bin.txt  dirlist-sbin.txt  dirlist/usr-sbin.txt  
dirlist/usr-bin.txt
```

Podemos realizar una simple búsqueda de nuestra lista de archivos de la siguiente manera:

```
[me@linuxbox ~]$ grep bzip dirlist*.txt  
dirlist-bin.txt:bzip2 dirlist-  
bin.txt:bzip2recover
```

En este ejemplo, grep busca en todos los archivos enumerados la cadena bzip y encuentra dos coincidencias, ambas en el archivo *dirlist-bin.txt*. Si estuviéramos interesados solo en los archivos que contienen coincidencias en lugar de las coincidencias en sí, podríamos especificar la opción -l:

```
[me@linuxbox ~]$ grep -l bzip dirlist*.txt  
dirlist-bin.txt
```

Por el contrario, si quisieramos ver una lista de solo los archivos que no contienen una coincidencia, podríamos hacer lo siguiente:

```
[me@linuxbox ~]$ grep -L bzip dirlist*.txt  
dirlist-sbin.txt  
dirlist/usr-bin.txt  
dirlist/usr-sbin.txt
```

## **Metacaracteres y literales**

Aunque no lo parezca, nuestras búsquedas grep han estado utilizando expresiones regulares todo el tiempo, aunque muy simples. La expresión regular bzip es

Se entiende que sólo se producirá una coincidencia si la línea del archivo contiene al menos cuatro caracteres y que en algún lugar de la línea los caracteres *b*, *z*, *i* y *p* se encuentran en ese orden, sin ningún otro carácter intermedio. Los caracteres de la cadena bzip son todos *caracteres literales*, en el sentido de que coinciden consigo mismos. Además de los literales, las expresiones regulares también pueden incluir *metacaracteres*, que se utilizan para especificar coincidencias más complejas. Los metacaracteres de las expresiones regulares constan de lo siguiente:

^ \$ . [ ] { } - ? \* + ( ) | \

Todos los demás caracteres se consideran literales, aunque el carácter de barra invertida se utiliza en algunos casos para crear *metasecuencias*, además de permitir que los metacaracteres se escapen y se traten como literales en lugar de interpretarse como metacaracteres.

**Nota:** *Como podemos ver, muchos de los metacaracteres de expresión regular también son caracteres que tienen significado para el shell cuando se realiza la expansión.*

*Cuando pasamos expresiones regulares que contienen metacaracteres en la línea de comandos, es vital que estén entre comillas para evitar que el shell intente expandirlas.*

## El carácter cualquiera

El primer metacarácter que veremos es el carácter de punto o punto, que se utiliza para coincidir con cualquier carácter. Si lo incluimos en una expresión regular, coincidirá con cualquier carácter en esa posición de carácter. He aquí un ejemplo:

---

```
[me@linuxbox ~]$ grep -h '.zip' dirlist*.txt
bunzip2
bzip2
bzip2recover
gunzip
gzip
funzip
gpg-zip
preunzip
prezip
prezip-bin
descompri
mir
descompri
mir sfx
```

---

Buscamos cualquier línea en nuestros archivos que coincida con la expresión regular *.zip*. Hay un par de cosas interesantes a tener en cuenta sobre los resultados. Observe que no se encontró el programa zip. Esto se debe a que la inclusión del metacarácter de punto en nuestra expresión regular aumentó la longitud de la coincidencia requerida a cuatro caracteres; Debido a que el nombre *zip* contiene solo tres, no coincide. Además, si algún archivo de nuestras listas hubiera contenido el *.zip* de extensión de archivo, habrían coincidido, porque el carácter de punto en la extensión del archivo también se trata como "cualquier carácter".

## Ancajes

Los caracteres de símbolo de intercalación (^) y signo de dólar (\$) se tratan como *anclas* en las expresiones regulares. Esto significa que hacen que la coincidencia ocurra solo si la expresión regular se encuentra al principio de la línea (^) o al final de la línea (\$).

```
[me@linuxbox ~]$ grep -h '^zip' dirlist*.txt
zip
zipcloak
zipgrep
zipinfo
zipnote
zipsplit
[me@linuxbox ~]$ grep -h 'zip$' dirlist*.txt
G.I.P.
de
segunda
mano
[me@linuxbox ~]$ grep -h '^zip$' dirlist*.txt
cremalleria
```

Aquí buscamos en la lista de archivos la cadena zip ubicada al principio de la línea, al final de la línea y en una línea donde está tanto al principio como al final de la línea (es decir, sola en la línea). Tenga en cuenta que la expresión regular ^\$ (un principio y un final sin nada en el medio) coincidirá con líneas en blanco.

### UN CRUCIGRAMA HE LPER

A mi esposa le encantan los crucigramas y, a veces, me pide ayuda con una pregunta en particular. Algo así como: "¿Qué es una palabra de cinco letras cuya tercera letra es *j* y la última letra es *r* que significa...?" Este tipo de preguntas me hicieron pensar.

¿Sabías que tu sistema Linux contiene un diccionario? Lo hace. Echa un vistazo en el directorio */usr/share/dict* y puede que encuentres uno o varios. Los archivos de diccionario que se encuentran allí son solo largas listas de palabras, una por línea, ordenadas alfabéticamente. En mi sistema, el archivo de *palabras* contiene poco más de 98.500 palabras. Para encontrar posibles respuestas a la pregunta del crucigrama anterior, podríamos hacer lo siguiente:

```
[me@linuxbox ~]$ grep -i '^.. j.r$' /usr/compartir/dictar/palabras
Mayor
Mayor
```

Usando esta expresión regular, podemos encontrar todas las palabras en nuestro archivo de diccionario que tienen cinco letras y tienen una *j* en la tercera posición y una *r* en la última posición.

## Expresiones de corchetes y clases de caracteres

Además de hacer coincidir cualquier carácter en una posición dada en nuestra expresión regular, también podemos hacer coincidir un solo carácter de un conjunto específico de caracteres mediante el uso de *expresiones entre paréntesis*. Con las expresiones entre corchetes, podemos especificar un conjunto de caracteres (incluidos los caracteres que, de otro modo, se interpretarían como metacaracteres) para que coincidan. En este ejemplo, usando un conjunto de dos caracteres, hacemos coincidir cualquier línea que contenga la cadena bzip o gzip:

---

```
[me@linuxbox ~]$ grep -h '[bg]zip' dirlist*.txt
bzip2
bzip2recuper
ar gzip
```

---

Un conjunto puede contener cualquier número de caracteres, y los metacaracteres pierden su significado especial cuando se colocan entre paréntesis. Sin embargo, hay dos casos en los que los metacaracteres se utilizan dentro de las expresiones entre corchetes y tienen significados diferentes. El primero es el símbolo de intercalación (^), que se utiliza para indicar negación; El segundo es el guion (-), que se utiliza para indicar un rango de caracteres.

### Negación

Si el primer carácter de una expresión entre corchetes es un símbolo de intercalación (^), los caracteres restantes se consideran un conjunto de caracteres que no deben estar presentes en la posición de carácter especificada. Para ello, modificamos nuestro ejemplo anterior:

---

```
[me@linuxbox ~]$ grep -h '[^bg]zip' dirlist*.txt
La 2ª
preprod
ucción
del G-
2000
prezip-bin
descompri
mir
descompri
mir sfx
```

---

Con la negación activada, obtenemos una lista de archivos que contienen la cadena zip precedida por cualquier carácter excepto *b* o *g*. Observe que no se encontró el archivo *zip*. Un conjunto de caracteres negado todavía requiere un carácter en la posición dada, pero el carácter no debe ser miembro del conjunto negado.

El carácter de intercalación invoca la negación solo si es el primer carácter dentro de una expresión entre corchetes; de lo contrario, pierde su significado especial y se convierte en un personaje ordinario en el conjunto.

### *Rangos de caracteres tradicionales*

Si quisieramos construir una expresión regular que encontrara todos los archivos de nuestras listas cuyo nombre comience con una letra mayúscula, podríamos hacer lo siguiente:

---

```
[me@linuxbox -]$ grep -h '^[ABCDEFGHIJKLMNOPQRSTUVWXYZ]' dirlist*.txt
```

---

Solo es cuestión de poner las 26 letras mayúsculas en una expresión entre paréntesis.

Pero la idea de toda esa escritura es profundamente inquietante, por lo que hay otra manera:

---

```
[me@linuxbox ~]$ grep -h '^[A-Z]' dirlist*.txt
MAKEDEV
Panel de
control GET
POST
E DE
CABE
ZA X
X11
Xorg
MAKEFLOPPIES
NetworkManager
NetworkManagerDispatcher
```

---

Al usar un rango de 3 caracteres, podemos abreviar las 26 letras. Cualquier rango de caracteres se puede expresar de esta manera, incluidos varios rangos como esta expresión, que coincide con todos los nombres de archivo que comienzan con letras y números:

---

```
[me@linuxbox ~]$ grep -h '^-[A-Za-z0-9]' dirlist*.txt
```

---

En los rangos de caracteres, vemos que el carácter de guión se trata de manera especial, entonces, ¿cómo incluimos realmente un carácter de guión en una expresión de corchetes? Convirtiéndolo en el primer carácter de la expresión. Considerar

---

```
[me@linuxbox ~]$ grep -h '[A-Z]' dirlist*.txt
```

---

Esto coincidirá con todos los nombres de archivo que contengan una letra mayúscula. Esto, por otro lado,

---

```
[me@linuxbox ~]$ grep -h '[-AZ]' dirlist*.txt
```

---

coincidirá con todos los nombres de archivo que contengan un guión, una A mayúscula o una Z mayúscula.

### **Clases de caracteres POSIX**

Los rangos de caracteres tradicionales son una forma fácil de entender y eficaz de manejar el problema de especificar rápidamente conjuntos de caracteres. Desgraciadamente, no siempre funcionan. Si bien hasta ahora no hemos encontrado ningún problema con el uso de grep , es posible que tengamos problemas al usar otros programas.

En el Capítulo 4, analizamos cómo se utilizan los comodines para realizar la expansión de nombres de ruta. En esa discusión, dijimos que los rangos de caracteres podrían usarse de una manera casi idéntica a la forma en que se usan en las expresiones regulares, pero aquí está el problema:

---

```
[me@linuxbox ~]$ ls /usr/sbin/[ABCDEFGHIJKLMNOPQRSTUVWXYZ]*
/usr/sbin/MAKEFLOPPIES
```

---

`/usr/sbin/NetworkManagerDispatcher`  
`/usr/sbin/Administrador de red`

---

(Dependiendo de la distribución de Linux, obtendremos una lista diferente de archivos, posiblemente una lista vacía. Este ejemplo es de Ubuntu.) Este comando produce el resultado esperado: una lista de solo los archivos cuyos nombres comienzan con una letra mayúscula. Pero con este comando obtenemos un resultado completamente diferente (solo se muestra una lista parcial de los resultados):

---

```
[me@linuxbox ~]$ ls /usr/sbin/[A-Z]*  
/usr/sbin/biosdecode  
/usr/sbin/chat  
/usr/sbin/chgpasswd  
/usr/sbin/chpasswd  
/usr/sbin/chroot  
/usr/sbin/cleanup-info  
/usr/sbin/quejarse  
/usr/sbin/demonio-de-log-kit de consola
```

---

¿Por qué? Es una historia larga, pero aquí está la versión corta.

Cuando Unix se desarrolló por primera vez, solo conocía los caracteres ASCII, y esta característica refleja ese hecho. En ASCII, los primeros 32 caracteres (números del 0 al 31) son códigos de control (como tabulaciones, retrocesos y retornos de transporte). Los siguientes 32 (32–63) contienen caracteres imprimibles, incluidos la mayoría de los caracteres de puntuación y los números del cero al nueve. Los 32 siguientes (números 64-95) contienen las letras mayúsculas y algunos símbolos de puntuación más. Los 31 finales (números 96-127) contienen letras minúsculas y aún más símbolos de puntuación. Sobre la base de esta disposición, los sistemas que utilizaban ASCII utilizaban un *orden de intercalación* similar al siguiente:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

Esto difiere del orden adecuado del diccionario, que es así:

aAbBcCdDeEfFgGhHiljJkKllMmNnOoPpqrRsStTuUvVwWxXyYzZ

A medida que la popularidad de Unix se extendió más allá de los Estados Unidos, creció la necesidad de admitir caracteres que no se encontraban en el inglés de los Estados Unidos. La tabla ASCII se amplió para usar un total de 8 bits, agregando números de caracteres 128-255, que se adaptaban a muchos más idiomas. Para respaldar esta capacidad, los estándares POSIX introdujeron un concepto llamado *configuración regional*, que podía ajustarse para seleccionar el conjunto de caracteres necesario para una ubicación en particular. Podemos ver la configuración del idioma de nuestro sistema usando este comando:

---

```
[me@linuxbox ~]$ echo $LANG  
en_US.UTF-8
```

---

Con esta configuración, las aplicaciones compatibles con POSIX usarán un orden de intercalación de diccionario en lugar de un orden ASCII. Esto explica el comportamiento de los comandos anteriores. Un rango de caracteres de [A-Z], cuando se interpreta en el orden del

diccionario, incluye todos los caracteres alfabéticos excepto la a minúscula, de ahí nuestros resultados.

Para solucionar parcialmente este problema, el estándar POSIX incluye una serie de clases de caracteres, que proporcionan rangos útiles de caracteres. Se describen en la Tabla 19-2.

Tabla 19-2: Clases de caracteres POSIX

Carácter Clase	Descripción
[:alnum:]	Los caracteres alfanuméricos; en ASCII, equivalente a [A-Za-z0-9]
[:palabra:]	Lo mismo que [:alnum:], con la adición del carácter de subrayado (_)
[:alfa:]	Los caracteres alfabéticos; en ASCII, equivalente a [A-Za-z]
[:en blanco:]	Incluye los caracteres de espacio y tabulación
[:cntrl:]	Los códigos de control ASCII; incluye los caracteres ASCII 0 a través de 31 y 127
[:d igit:]	Los números del 0 al 9
[:gráfico:]	Los caracteres visibles; en ASCII, incluye los caracteres del 33 al 126
[:inferior:]	Las letras minúsculas
[:p UNCT:]	Los caracteres de puntuación; en ASCII, equivalente a [-!"#\$%&'(*+,./; <=>?@[\\"\\]_`{ }~]
[:p rint:]	Los caracteres imprimibles; Todos los personajes de [:gráfico:] más el carácter de espacio
[:espacio:]	Los caracteres de espacio en blanco, incluidos el espacio, la tabulación, el carro retorno, nueva línea, tabulación vertical y avance de formularios; en ASCII, equivalente a [ \t\r\n\v\f]
[:superior:]	Los caracteres en mayúsculas
[:xdígito:]	Caracteres utilizados para expresar números hexadecimales; en ASCII, equivalente a [0-9A-Fa-f]

Incluso con las clases de caracteres, todavía no hay una forma conveniente de expresar rangos parciales, como [A-M].

Usando clases de caracteres, podemos repetir nuestra lista de directorios y ver un resultado mejorado.

```
[me@linuxbox ~]$ ls /usr/sbin/[[:upper:]]*
/usr/sbin/MAKEFLOPPIES
/usr/sbin/NetworkManagerDispatcher
/usr/sbin/Administrador de red
```

Recuérdese, sin embargo, que este no es un ejemplo de una expresión regular; más bien, es el shell el que realiza la expansión del nombre de ruta. Lo mostramos aquí porque las clases de caracteres POSIX se pueden usar para ambos.

## VOLVER AL ORDEN DE INTERCALACIÓN TRADICIONAL

Puede optar por que el sistema utilice el orden de intercalación tradicional (ASCII) cambiando el valor de la variable de entorno `LANG`. Como vimos en la sección anterior, la variable `LANG` contiene el nombre del idioma y el conjunto de caracteres utilizados en su configuración regional. Este valor se determinó originalmente cuando seleccionó un idioma de instalación a medida que se instalaba Linux.

Para ver la configuración regional, utilice el comando `locale`:

```
[me@linuxbox ~]$ locale
LANG=en_US.UTF-8
LC_CTYPE="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_COLLATE="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_MESSAGES="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8" LC_ALL=
```

Para cambiar la configuración regional para usar los comportamientos tradicionales de Unix, establezca el parámetro `LANG` variable a POSIX:

```
[me@linuxbox ~]$ export LANG=POSIX
```

Tenga en cuenta que este cambio convierte el sistema para usar el inglés de EE. UU. (más específicamente, ASCII) para su conjunto de caracteres, así que asegúrese de que esto es realmente lo que desea.

Puedes hacer que este cambio sea permanente añadiendo esta línea a tu archivo `.bashrc`:

```
export LANG=POSIX
```

## POSIX Expresiones regulares básicas frente a expresiones regulares extendidas

Justo cuando pensábamos que esto no podía ser más confuso, descubrimos que POSIX también divide las implementaciones de expresiones regulares en dos tipos: *expresiones regulares básicas (BRE)* y *expresiones regulares extendidas (ERE)*. Las características que hemos cubierto hasta ahora son compatibles con cualquier aplicación que sea compatible con POSIX e implemente BRE.

Nuestro programa `grep` es uno de esos programas.

¿Cuál es la diferencia entre BRE y ERE? Es una cuestión de metacaracteres. Con BRE, se reconocen los siguientes metacaracteres: ^ \$ . [ ] \* Todos los demás caracteres se consideran literales. Con ERE, se añaden los siguientes metacaracteres (y sus funciones asociadas): ( ) { } ? + |

Sin embargo (y esta es la parte divertida), los caracteres () {} se tratan como metacaracteres en BRE *si* se escapan con una barra invertida, mientras que con ERE, preceder a cualquier metacarácter con una barra invertida hace que se trate como un literal.

Dado que las características que vamos a discutir a continuación son parte de ERE, vamos a necesitar usar un grep diferente. Tradicionalmente, esto ha sido realizado por el programa egrep, pero la versión GNU de grep también soporta expresiones regulares extendidas cuando se utiliza la opción -E.

## POSIX

Durante la década de 1980, Unix se convirtió en un sistema operativo comercial muy popular, pero en 1988, el mundo Unix estaba en crisis. Muchos fabricantes de computadoras habían licenciado el código fuente de Unix de sus creadores, AT&T, y estaban suministrando diversas versiones del sistema operativo con sus sistemas. Sin embargo, en sus esfuerzos por crear una diferenciación del producto, cada fabricante agregó cambios y extensiones patentados. Esto comenzó a limitar la compatibilidad del software. Como siempre ocurre con los proveedores propietarios, cada uno intentaba jugar un juego ganador de "bloqueo" con sus clientes. Esta época oscura en la historia de Unix se conoce hoy como *la balcanización*.

Entra en el IEEE (Instituto de Ingenieros Eléctricos y Electrónicos). A mediados de la década de 1980, el IEEE comenzó a desarrollar un conjunto de estándares que definirían cómo funcionarían los sistemas Unix (y similares a Unix). Estos estándares, formalmente conocidos como IEEE 1003, definen las *interfaces de programación de aplicaciones (API)*, el shell y las utilidades que se encuentran en un sistema estándar tipo Unix. El nombre *POSIX*, que significa *Interfaz de Sistema Operativo Portátil* (con la X añadida al final para mayor agilidad), fue sugerido por Richard Stallman (sí, ese Richard Stallman) y fue adoptado por el IEEE.

## Alternancia

La primera de las características de las expresiones regulares extendidas que discutiremos se llama *alternancia*, que es la facilidad que permite que se produzca una coincidencia entre un conjunto de expresiones. Al igual que una expresión entre corchetes permite que un solo carácter coincida con un conjunto de caracteres especificados, la alternancia permite coincidencias con un conjunto de cadenas u otras expresiones regulares.

Para demostrarlo, usaremos grep junto con echo. Primero, intentemos una coincidencia de cadena simple:

```
[me@linuxbox ~]$ echo "AAA" | grep AAA
AAA
[me@linuxbox ~]$ echo "BBB" | grep AAA
[me@linuxbox ~]$
```

Un ejemplo bastante sencillo, en el que canalizamos la salida de echo a grep y vemos los resultados. Cuando se produce una coincidencia, la vemos impresa; Cuando no se produce ninguna coincidencia, no vemos

resultados.

Ahora agregaremos una alternancia, representada por el metacarácter de barra vertical vertical:

```
[me@linuxbox ~]$ echo "AAA" | grep -E 'AAA|BBB'  
AAA  
[me@linuxbox ~]$ echo "BBB" | grep -E 'AAA|BBB'  
BBB  
[me@linuxbox ~]$ echo "CCC" | grep -E 'AAA|BBB'  
[me@linuxbox ~]$
```

Aquí vemos la expresión regular 'AAA|BBB', que significa "coincidir con la cadena AAA o la cadena BBB". Nótese que, dado que se trata de una característica extendida, añadimos la opción -E a grep (aunque podríamos haber utilizado el programa egrep en su lugar), y encerramos la expresión regular entre comillas para evitar que el shell interprete el metacarácter de la tubería vertical como un operador de tubería. La alternancia no se limita a dos opciones:

```
[me@linuxbox ~]$ echo "AAA" | grep -E 'AAA|BBB|CCC'  
AAA
```

Para combinar la alternancia con otros elementos de expresión regular, podemos usar () para separar la alternancia:

```
[me@linuxbox ~]$ grep -Eh '^^(bz|gz|zip)' dirlist*.txt
```

Esta expresión coincidirá con los nombres de archivo de nuestras listas que comiencen con bz, gz o zip. Si dejamos los paréntesis, el significado de esta expresión regular cambia para coincidir con cualquier nombre de archivo que comience con bz o contenga gz o contenga zip:

```
[me@linuxbox ~]$ grep -Eh '^bz|gz|zip' dirlist*.txt
```

## Cuantificadores

Las expresiones regulares extendidas admiten varias formas de especificar el número de veces que coincide un elemento.

### *?: coincidir con un elemento cero veces o una vez*

Este cuantificador significa, en efecto, "Hacer que el elemento anterior sea opcional". Supongamos que queremos comprobar la validez de un número de teléfono y consideramos que un número de teléfono es válido si coincide con cualquiera de estas dos formas, (nnn) nnn-nnnn o nnn nnn-nnnn, donde n es un número. Podríamos construir una expresión regular como esta:

```
^\(? [0-9] [0-9] [0-9]\)? [0-9] [0-9] [0-9]-[0-9][0-9][0-9]$
```

En esta expresión, seguimos los caracteres entre paréntesis con signos de interrogación para indicar que deben coincidir cero o una vez. De nuevo, dado que los paréntesis son normalmente metacaracteres (en ERE), los precedemos con barras invertidas para que se traten como literales.

Vamos a probarlo:

```
[me@linuxbox ~]$ echo "(555) 123-4567" | grep -E '^\\(\\d{3}\\) \\d{3}-\\d{4}$'  
(555) 123-4567  
[me@linuxbox ~]$ echo "555 123-4567" | grep -E '^\\d{3} \\d{3}-\\d{4}$'  
555 123-4567  
[me@linuxbox ~]$ echo "AAA 123-4567" | grep -E '^\\d{3} \\d{3}-\\d{4}$'  
[me@linuxbox ~]$
```

Aquí vemos que la expresión coincide con ambas formas del número del teléfono, pero no coincide con una que contenga caracteres no numéricos.

\*: coincidir con un elemento cero o más veces

Al igual que el  $\text{?}$  metacarácter, el  $*$  se usa para denotar un elemento opcional; sin embargo, a diferencia del  $\text{?}$ , el elemento puede aparecer cualquier número de veces, no solo una vez. Digamos que queremos ver si una cadena es una oración; es decir, comienza con una letra mayúscula, luego contiene cualquier número de letras mayúsculas y minúsculas y espacios, y termina con un punto. Para coincidir con esta definición (muy cruda) de una oración, podríamos usar una expresión regular como esta:

**[[:superior:]]**[[:superior:]]**[:inferior: ]\***\.

La expresión consta de tres elementos: una expresión entre corchetes que contiene la clase de caracteres [:upper:], una expresión entre corchetes que contiene las clases de caracteres [:upper:] y [:lower:] y un espacio, y un punto escapado con una barra invertida. El segundo elemento se deja ir con un metacarácter \* de modo que después de la letra mayúscula inicial en nuestra oración, cualquier número de letras mayúsculas y minúsculas y espacios pueden seguirla y aún coincidir:

```
[me@linuxbox ~]$ echo "Esto funciona." | grep -E '[[:upper:]][[:upper:]][[:lower:]]*\.'
Esto funciona.
[me@linuxbox ~]$ echo "Esto funciona." | grep -E '[[:upper:]][[:upper:]][[:lower:]]*\.'
Esto funciona.
[me@linuxbox ~]$ echo "esto no" | grep -E '[[:upper:]][[:upper:]][[:lower:]]*\.'
[me@linuxbox ~]$
```

La expresión coincide con las dos primeras pruebas, pero no con la tercera, ya que carece del carácter en mayúsculas inicial y el punto final necesarios.

+–Hacer coincidir un elemento una o más veces

El metacarácter + funciona de forma muy similar a \*, excepto que requiere al menos una instancia del elemento anterior para provocar una

coincidencia. Esta es una expresión regular que solo coincidirá con líneas que constan de grupos de uno o más caracteres alfabéticos separados por espacios simples:

$^([[:alfa:]]+ ?)+\$$

Vamos a probarlo:

```
[me@linuxbox ~]$ echo "Esto que" | grep -E '^([[:alpha:]]+ ?)+$'  
Esto que  
[me@linuxbox ~]$ echo "a b c" | grep -E '^([[:alpha:]]+ ?)+$'  
a b c  
[me@linuxbox ~]$ echo "a b 9" | grep -E '^([[:alpha:]]+ ?)+$'  
[me@linuxbox ~]$ echo "abc d" | grep -E '^([[:alpha:]]+ ?)+$'  
[me@linuxbox ~]$
```

Vemos que esta expresión no coincide con la línea "a b 9", porque contiene un carácter no alfabético; Tampoco coincide con "abc d", porque más de un carácter de espacio separa los caracteres *c* y *d*.

### *{ }:* coincide con un elemento un número específico de veces

Los metacaracteres { y } se utilizan para expresar el número mínimo y máximo de coincidencias necesarias. Pueden especificarse de cuatro maneras posibles, como se muestra en la Tabla 19-3.

Tabla 19-3: Especificación del número de coincidencias

Especificador	Significado
{n}	Coincide con el elemento anterior si se produce exactamente <i>n</i> veces.
{n,m}	Coincide con el elemento anterior si se produce al menos <i>n</i> veces, pero no más de <i>m</i> veces.
{n,}	Coincide con el elemento anterior si se produce <i>n</i> o más veces.
{,m}	Coincide con el elemento anterior si no se produce más de <i>m</i> veces.

Volviendo a nuestro ejemplo anterior con los números de teléfono, podemos usar este método de especificar repeticiones para simplificar nuestra expresión regular original de

```
^\(? [0-9] [0-9] [0-9]\)? [0-9] [0-9] [0-9]-[0-9][0-9][0-9][0-9]$
```

Para

```
^\(? [0-9] {3}\)? [0-9] {3}-[0-9]{4}$
```

Vamos a probarlo:

```
[me@linuxbox ~]$ echo "(555) 123-4567" | grep -E '^\(?[ 0-9]{3}\)? [0-9]{3}-[0-9]{4}$'  
(555) 123-4567  
[me@linuxbox ~]$ echo "555 123-4567" | grep -E '^\(?[ 0-9]{3}\)? [0-9]{3}-[0-9]{4}$'  
555 123-4567  
[me@linuxbox ~]$ echo "5555 123-4567" | grep -E '^\(?[ 0-9]{3}\)? [0-9]{3}-[0-9]{4}$'  
[me@linuxbox ~]$
```

Como podemos ver, nuestra expresión revisada puede validar con éxito números con y sin paréntesis, mientras rechaza aquellos números que no tienen el formato adecuado.

## Poner a trabajar las expresiones regulares

Echemos un vistazo a algunos de los comandos que ya conocemos y veamos cómo se pueden usar con expresiones regulares.

### Validación de una lista de teléfonos con grep

En nuestro ejemplo anterior, observamos números de teléfono individuales y verificamos que tuvieran el formato adecuado. Un escenario más realista sería revisar una lista de números, así que hagamos una lista. Lo haremos recitando un conjuro mágico en la línea de comandos. Será mágico porque no hemos cubierto la mayoría de los comandos involucrados, pero no te preocupes, llegaremos allí en capítulos futuros. Aquí está el conjuro:

---

```
[me@linuxbox ~]$ for i in {1..10}; do echo "(${RANDOM:0:3}) ${RANDOM:0:3}-$${ALEATORIO:0:4}" >> phonelist.txt; hecho
```

---

Este comando producirá un archivo llamado *phonelist.txt* que contiene 10 números de teléfono. Cada vez que se repite el comando, se agregan otros 10 números a la lista. También podemos cambiar el valor 10 cerca del comienzo del comando para producir más o menos números de teléfono. Sin embargo, si examinamos el contenido del archivo, vemos que tenemos un problema:

---

```
[me@linuxbox ~]$ gato phonelist.txt
(232) 298-2265
(624) 381-1078
(540) 126-1980
(874) 163-2885
(286) 254-2860
(292) 108-518
(129) 44-1379
(458) 273-1642
(686) 299-8268
(198) 307-2440
```

---

Algunos de los números están mal formados, lo cual es perfecto para nuestros propósitos porque usaremos grep para validarlos.

Un método útil de validación sería escanear el archivo en busca de números no válidos y mostrar la lista resultante.

---

```
[me@linuxbox ~]$ grep -ev '^\\([0-9]{3}\\) [0-9]{3}-[0-9]{4}$' phonelist.txt
(292) 108-518
(129) 44-1379
[me@linuxbox ~]$
```

---

Aquí usamos la opción *-v* para producir una coincidencia inversa, de modo que solo generaremos las líneas de la lista que no coincidan con la expresión especificada.

La propia expresión incluye los metacaracteres de anclaje en cada extremo para asegurarse de que el número no tenga caracteres adicionales en ninguno de los extremos. Esta expresión también requiere que los paréntesis estén presentes en un número válido, a diferencia de nuestro ejemplo anterior de número de teléfono.

### ***Encontrar nombres de archivo feos con find***

El comando `find` admite una prueba basada en una expresión regular. Hay una consideración importante que se debe tener en cuenta al usar expresiones regulares en `find` frente a `grep`. Mientras que `grep` imprimirá una línea cuando la línea *contenga* una cadena que coincide con una expresión, `find` requiere que el nombre de la ruta *coincida exactamente con* la expresión regular. En el siguiente ejemplo, usaremos `find` con una expresión regular para encontrar cada nombre de ruta que contenga cualquier carácter que no sea miembro del siguiente conjunto:

```
[_-./0-9a-zA-Z]
```

Un análisis de este tipo revelaría nombres de ruta que contienen espacios incrustados y otros caracteres potencialmente ofensivos:

---

```
[me@linuxbox ~]$ encontrar . -regex '.*[^-_./0-9a-zA-Z].*'
```

---

Debido al requisito de una coincidencia exacta de todo el nombre de la ruta, usamos `.*` en ambos extremos de la expresión para que coincida con cero o más instancias de cualquier carácter. En el centro de la expresión, usamos una expresión de corchetes negados que contiene nuestro conjunto de caracteres de nombre de ruta aceptables.

### ***Búsqueda de archivos con locate***

El programa `locate` admite expresiones regulares básicas (la opción `--regexp`) y extendidas (la opción `--regex`). Con él, podemos realizar muchas de las mismas operaciones que realizamos anteriormente con nuestros archivos *de lista de directorios*:

---

```
[me@linuxbox ~]$ locate --regex 'bin/(bz|gz|zip)'  
/bin/bzcat  
/bin/bzcmp  
/bin/bzdiff  
/bin/bzegrep  
/bin/bzexe  
/bin/bzfgrep  
/bin/bzgrep  
/papelera/bzip2  
/bin/bzip2recover  
/bin/bzless  
/bin/bzmore  
/bin/gzexe  
/bin/gzip  
/usr/bin/zip  
/usr/bin/zipcloak  
/usr/bin/zipgrep  
/impacto/papelera/zipinfo
```

`/usr/bin/zipnote`  
`/usr/bin/zipsplit`

---

Con la alternancia, realizamos una búsqueda de nombres de ruta que contengan *bin/bz*, *bin/gz* o */bin/zip*.

### Búsqueda de texto con less y vim

less y vim comparten el mismo método de búsqueda de texto. Al presionar la tecla / seguida de una expresión regular, se realizará una búsqueda. Usamos menos para ver nuestro archivo *phonelist.txt*:

---

```
[me@linuxbox ~]$ menos phonelist.txt
```

---

A continuación, buscamos nuestra expresión de validación:

---

```
(232) 298-2265
(624) 381-1078
(540) 126-1980
(874) 163-2885
(286) 254-2860
(292) 108-518
(129) 44-1379
(458) 273-1642
(686) 299-8268
(198) 307-2440
~
~
~
/^([0-9]{3}\) [0-9]{3}-[0-9]{4}$
```

---

less resaltará las cadenas que coincidan, dejando las inválidas fáciles de detectar:

---

```
(232) 298-2265
(624) 381-1078
(540) 126-1980
(874) 163-2885
(286) 254-2860
(292) 108-518
(129) 44-1379
(458) 273-1642
(686) 299-8268
(198) 307-2440
~
~
~
(FEN
D)
```

---

Vim, por otro lado, admite expresiones regulares básicas, por lo que nuestra expresión de búsqueda se vería así:

```
/([0-9]\{3\}) [0-9]\{3\}-[0-9]\{4\}
```

Podemos ver que la expresión es casi la misma; Sin embargo, muchos de los caracteres que se consideran metacaracteres en las expresiones extendidas se consideran literales en las expresiones básicas. Se tratan como metacaracteres

solo cuando se escapa con una barra invertida. Dependiendo de la configuración particular de vim en nuestro sistema, se resaltará la coincidencia. Si no es así, pruebe el comando de modo de comando :hlsearch para activar el resaltado de búsqueda.

**Nota:** *Dependiendo de su distribución, vim puede o no admitir el resaltado de búsqueda de texto. Ubuntu, en particular, proporciona una versión muy reducida de vim de forma predeterminada. En estos sistemas, es posible que desee utilizar su administrador de paquetes para instalar una versión más completa de vim.*

## Nota final

En este capítulo, hemos visto algunos de los muchos usos de las expresiones regulares.

Podemos encontrar aún más si usamos expresiones regulares para buscar aplicaciones adicionales que las usen. Podemos hacerlo buscando en las páginas del manual:

---

```
[me@linuxbox ~]$ cd /usr/share/man/man1
[me@linuxbox man1]$ zgrep -El 'regex | expresión regular' *.gz
```

---

El programa zgrep proporciona una interfaz para grep, lo que le permite leer archivos comprimidos. En nuestro ejemplo, buscamos los archivos comprimidos de la página del manual de la Sección 1 en su ubicación habitual. El resultado de este comando es una lista de archivos que contienen la cadena `regex` o `expresión regular`. Como vemos, las expresiones regulares aparecen en muchos programas.

Hay una característica que se encuentra en las expresiones regulares básicas que no cubrimos. Llamadas *referencias inversas*, esta característica se discutirá en el próximo capítulo.

# 20

## PROCESAMIENTO DE TEXTOS

Todos los sistemas operativos tipo Unix dependen en gran medida de los archivos de texto para varios tipos de almacenamiento de datos. Por lo tanto, tiene sentido que haya muchas herramientas para manipular el texto. En este capítulo, veremos los programas que se utilizan

para "cortar y trocear" el texto. En el próximo capítulo, examinaremos más sobre el procesamiento de textos, centrándonos en los programas que se utilizan para formatear textos para impresión y otros tipos de consumo humano.

En este capítulo volveremos a visitar a algunos viejos amigos y nos presentaremos a algunos nuevos:

- cat: concatene archivos e imprímalos en la salida estándar.
- sort: ordena las líneas de los archivos de texto.
- uniq: informe u omita líneas repetidas.
- cortar: elimina secciones de cada línea de archivos.
- pegar: combina líneas de archivos.

- **join**: unir líneas de dos archivos en un campo común.
- **comm**: compara dos archivos ordenados línea por línea.

- diff: compara archivos línea por línea.
- patch: aplique un archivo diff a un original.
- tr: traduce o elimina caracteres.
- sed: editor de secuencias para filtrar y transformar texto.
- aspell: corrector ortográfico interactivo.

## Aplicaciones del texto

Hasta ahora, hemos aprendido sobre un par de editores de texto (nano y vim), hemos mirado un montón de archivos de configuración y hemos sido testigos de la salida de docenas de comandos, todos en texto. Pero, ¿para qué más se utiliza el texto? Resulta que hay muchas cosas.

### Documentos

Muchas personas escriben documentos utilizando formatos de texto sin formato. Si bien es fácil ver cómo un pequeño archivo de texto puede ser útil para tomar notas simples, también es posible escribir documentos grandes en formato de texto. Un enfoque popular es escribir un documento grande en formato de texto y luego usar un *lenguaje de marcado* para describir el formato del documento terminado. Muchos artículos científicos se escriben utilizando este método, ya que los sistemas de procesamiento de texto basados en Unix se encuentran entre los primeros sistemas que admitieron el diseño tipográfico avanzado que necesitaban los escritores en disciplinas técnicas.

### Páginas Web

El tipo de documento electrónico más popular del mundo es probablemente la página web. Las páginas web son documentos de texto que utilizan *HTML* (*Hypertext Markup Language*) o *XML* (*Extensible Markup Language*) como lenguaje de marcado para describir el formato visual del documento.

### Correo electrónico

El correo electrónico es un medio intrínsecamente basado en texto. Incluso los archivos adjuntos que no son de texto se convierten en una representación de texto para su transmisión. Podemos ver esto por nosotros mismos descargando un mensaje de correo electrónico y luego viéndolo en menos. Veremos que el mensaje comienza con un *encabezado* que describe el origen del mensaje y el procesamiento que recibió durante su recorrido, seguido del *cuerpo* del mensaje con su contenido.

### Salida de la impresora

En sistemas tipo Unix, la salida destinada a una impresora se envía como texto plano o, si la página contiene gráficos, se convierte en un lenguaje de descripción de páginas en formato de texto conocido como *PostScript*, que

luego se envía a un programa que genera los puntos gráficos que se van a imprimir.

## Código fuente del programa

Muchos de los programas de línea de comandos que se encuentran en sistemas similares a Unix fueron creados para apoyar la administración de sistemas y el desarrollo de software, y los programas de procesamiento de texto no son una excepción. Muchos de ellos están diseñados para resolver problemas de desarrollo de software. La razón por la que el procesamiento de texto es importante para los desarrolladores de software es que todo el software comienza como texto. *El código fuente*, la parte del programa que realmente escribe el programador, está siempre en formato de texto.

## Volver a visitar a algunos viejos amigos

En el Capítulo 6, aprendimos sobre algunos comandos que pueden aceptar entradas estándar además de los argumentos de la línea de comandos. Solo los mencionamos brevemente entonces, pero ahora analizaremos más de cerca cómo se pueden usar para realizar el procesamiento de texto.

### *cat: concatenar archivos e imprimir en una salida estándar*

El programa para gatos tiene una serie de opciones interesantes. Muchos de ellos se utilizan para visualizar mejor el contenido del texto. Un ejemplo es la opción **-A**, que se utiliza para mostrar caracteres no imprimibles en el texto. Hay ocasiones en las que queremos saber si los caracteres de control están incrustados en nuestro texto visible. Los más comunes son los caracteres de tabulación (en lugar de los espacios) y los retornos de acarreo, a menudo presentes como caracteres de fin de línea en archivos de texto de estilo MS-DOS. Otra situación común es un archivo que contiene líneas de texto con espacios finales.

Vamos a crear un archivo de prueba usando **cat** como un procesador de textos primitivo. Para hacer esto, simplemente ingresaremos el comando **cat** (junto con especificar un archivo para la salida redirigida) y escribiremos nuestro texto, seguido de **ENTER** para terminar correctamente la línea, luego **CTRL-D** para indicar a **cat** que hemos llegado al final del archivo. En este ejemplo, introducimos un carácter de tabulación inicial y seguimos la línea con algunos espacios finales:

---

```
[me@linuxbox ~]$ gato > foo.txt
      El rápido zorro marrón saltó sobre el perro perezoso.
[me@linuxbox ~]$
```

---

A continuación, usaremos **cat** con la opción **-A** para mostrar el texto:

---

```
[me@linuxbox ~]$ gato -A foo.txt
^ELe rápido zorro marrón saltó sobre el perro perezoso. $
[me@linuxbox ~]$
```

---

Como podemos ver en los resultados, el carácter de tabulación en nuestro texto está representado por **^I**. Esta notación común significa "CTRL-I", que, como resultado, es lo mismo que un carácter de tabulación. También vemos que aparece un **\$** en el extremo verdadero de la línea, lo que indica que nuestro texto contiene espacios finales.

## TEXTO MS-DOS VS. TEXTO UNIX

Una de las razones por las que es posible que desee usar cat para buscar caracteres no imprimibles en el texto es para detectar retornos de carro ocultos. ¿De dónde vienen los retornos de carro ocultos? ¡DOS y Windows! Unix y DOS no definen el final de una línea de la misma manera en los archivos de texto. Unix termina una línea con un carácter de salto de línea (ASCII 10), mientras que MS-DOS y sus derivados utilizan el retorno de carro de secuencia (ASCII 13) y el salto de línea para terminar cada línea de texto.

Hay varias formas de convertir archivos de formato DOS a Unix. En muchos sistemas Linux, los programas llamados dos2unix y unix2dos pueden convertir archivos de texto desde y hacia el formato DOS. Sin embargo, si no tienes dos2unix en tu sistema, no te preocupes. El proceso de conversión de texto de formato DOS a Unix es muy sencillo; simplemente implica la eliminación de los retornos de carro que infractores. Eso se logra fácilmente

cat también tiene opciones que se utilizan para modificar texto. Los dos más prominentes son -n, que numera las líneas, y -s, que suprime la salida de varias líneas en blanco. Podemos demostrar así:

```
[me@linuxbox ~]$ gato > foo.txt  
El rápido zorro marrón
```

```
saltó sobre el perro perezoso.  
[me@linuxbox ~]$ gato -ns foo.txt  
    1 El zorro marrón rápido  
    2  
    3 saltó sobre el perro perezoso.  
[me@linuxbox ~]$
```

En este ejemplo, creamos una nueva versión de nuestro archivo de *prueba foo.txt*, que contiene dos líneas de texto separadas por dos líneas en blanco. Después del procesamiento por cat con las opciones -ns, se elimina la línea en blanco adicional y se numeran las líneas restantes. Si bien esto no es un gran proceso para realizar en el texto, es un proceso.

### *sort: ordenar líneas de archivos de texto*

El programa de ordenación ordena el contenido de la entrada estándar, o uno o más archivos especificados en la línea de comandos, y envía los resultados a la salida estándar. Usando la misma técnica que usamos con cat, podemos demostrar el procesamiento de la entrada estándar directamente desde el teclado.

```
[me@linuxbox ~]$ ordenar > foo.txt  
c  
b  
a  
[me@linuxbox ~]$ gato foo.txt  
a  
b  
c
```

Después de ingresar el comando, escribimos las letras c, b y a, seguidas una vez más por CTRL-D para indicar el final del archivo. A continuación, vemos el archivo resultante y vemos que las líneas aparecen ahora en orden ordenado.

Dado que sort puede aceptar varios archivos en la línea de comandos como argumentos, es posible *fusionar* varios archivos en un solo conjunto ordenado. Por ejemplo, si tuviéramos tres archivos de texto y quisieramos combinarlos en un solo archivo ordenado, podríamos hacer algo como esto:

```
Sort file1.txt file2.txt file3.txt > final_sorted_list.txt sort tiene  
varias opciones interesantes. La Tabla 20-1 muestra una lista  
parcial.
```

Tabla 20-1: Opciones comunes de clasificación

Opción	Opción larga	Descripción
-b	--ignorar-los-espacios-en-blanco-principales	De forma predeterminada, la ordenación se realiza en el todo la línea, empezando por el primer carácter en la línea. Esta opción hace que ordenar para ignorar los espacios iniciales en las líneas y calcula la clasificación en función de la primera carácter que no sea un espacio en blanco en la línea.
-f	--ignorar-caso	Hace que la clasificación no distinga entre mayúsculas y minúsculas.
-n	--ordenar-numérico	Realiza la ordenación en función de los valores numéricos Evaluación de una cadena. Usando esto permite que la clasificación se realice en Valores numéricos en lugar de alfabéticos valores.
-r	--Marcha atrás	Ordenar en orden inverso. Los resultados están a la vista descendente en lugar de ascender orden.
-k	--key=campo1[,campo2]	Ordenar en función de un campo clave ubicado del campo1 al campo2 en lugar del campo toda la línea.
-m	--fusionar	Trate cada argumento como el nombre de un Archivo preclasificado. Fusionar varios archivos en

		Un único resultado ordenado sin cualquier clasificación adicional.
<b>-o</b>	<b>--salida=archivo</b>	Envíe la salida ordenada a <i>un archivo</i> en lugar de a salida estándar.
<b>-t</b>	<b>--separador-de-campo=char</b>	Defina el carácter separador de campos. Por Predeterminado, los campos están separados por espacios o pestañas.

Aunque la mayoría de las opciones anteriores se explican por sí mismas, algunas no lo son. En primer lugar, echemos un vistazo a la opción `-n`, que se utiliza para la ordenación numérica. Con esta opción, es posible ordenar los valores en función de valores numéricos. Podemos demostrarlo ordenando los resultados del comando `du` para determinar los mayores usuarios de espacio en disco. Normalmente, el comando `du` enumera los resultados de un resumen en orden de nombre de ruta:

---

```
[me@linuxbox ~]$ du -s /usr/share/* | head
252          /usr/share/aclocal
96           /usr/share/acpi-support
8            /usr/acciones/ADAS
196          /usr/compartir/alkarate
344          /usr/compartir/alsa
8            /usr/share/alsa-base
12488        /usr/compartir/anti
8            /usr/compartir/apmd
21440        /usr/share/app-install
48           /usr/share/application-registro
```

---

En este ejemplo, canalizamos los resultados a `head` para limitar los resultados a las primeras 10 líneas. Podemos producir una lista ordenada numéricamente para mostrar los 10 mayores consumidores de espacio de esta manera:

---

```
[me@linuxbox ~]$ du -s /usr/share/* | sort -nr | head
509940        /usr/share/locale-langpack
242660        /usr/compartir/doc
197560        /usr/compartir/fuentes
179144        /usr/share/gnome
146764        /usr/compartir/mispellografía
144304        /usr/compartir/GIMP
135880        /usr/compartir/dictar
76508         /usr/compartir/iconos
68072         /usr/compartir/aplicaciones
62844         /usr/compartir/foomatic
```

---

Mediante el uso de las opciones `-nr`, producimos una ordenación numérica inversa, en la que los valores más grandes aparecen primero en los resultados. Esta ordenación funciona porque los valores numéricos aparecen al principio de cada línea. Pero, ¿qué pasa si queremos ordenar una lista en función de algún valor que se encuentre dentro de la línea? Por ejemplo, el resultado de `ls -l` tiene el siguiente aspecto:

---

```
[me@linuxbox ~]$ ls -l /usr/bin | cabeza
Total 152948
-rwxr-xr-x 1 raíz    raíz      34824 2012-04-04 02:42 [ 
-rwxr-xr-x 1 raíz    raíz      101556 27/11/2011 06:08 a2p
-rwxr-xr-x 1 raíz    raíz      13036 27/02/2012 08:22 aconectar
-rwxr-xr-x 1 raíz    raíz      10552 15/08/2011 10:34 acpi
-rwxr-xr-x 1 raíz    raíz      3800 14/04/2012 03:51 acpi_fakekey
-rwxr-xr-x 1 raíz    raíz      7536 19/04/2012 00:19 acpi_listen
-rwxr-xr-x 1 raíz    raíz      3576 2012-04-29 07:57 addpart
-rwxr-xr-x 1 raíz    raíz      20808 2012-01-03 18:02 addr2line
-rwxr-xr-x 1 raíz    raíz      489704 09/10/2012 17:02 adept_batch
```

---

Ignorando, por el momento, que `ls` puede ordenar sus resultados por

tamaño, podríamos usar `sort` para ordenar esta lista por tamaño de archivo también.

---

```
[me@linuxbox ~]$ ls -l /usr/bin | sort -nr -k 5 | head
-rwxr-xr-x 1 raíz  raíz   8234216 2012-04-07 17:42 Paisaje de tinta
-rwxr-xr-x 1 raíz  raíz   8222692 2012-04-07 17:42 vista de tinta
-rwxr-xr-x 1 raíz  raíz   3746508 2012-03-07 23:45 GIMP-2.4
-rwxr-xr-x 1 raíz  raíz   3654020 2012-08-26 16:16 Quanta
-rwxr-xr-x 1 raíz  raíz   2928760 2012-09-10 14:31 gdbtui
-rwxr-xr-x 1 raíz  raíz   2928756 2012-09-10 14:31 Gdb
-rwxr-xr-x 1 raíz  raíz   2602236 2012-10-10 12:56 red
-rwxr-xr-x 1 raíz  raíz   2304684 2012-10-10 12:56 RPConder
-rwxr-xr-x 1 raíz  raíz   2241832 2012-04-04 05:56 aptitud
-rwxr-xr-x 1 raíz  raíz   2202476 2012-10-10 12:56 SMBCLACS
```

---

Muchos usos de `sort` implican el procesamiento de *datos tabulares*, como los resultados del comando `ls` anterior. Si aplicamos la terminología de la base de datos a la tabla anterior, diríamos que cada fila es un *registro* y que cada registro consta de varios campos, como los atributos del archivo, el recuento de enlaces, el nombre del archivo, el tamaño del archivo, etc. `sort` es capaz de procesar campos individuales. En términos de bases de datos, Podemos especificar uno o más *campos clave* para usar como *claves de ordenación*. En el ejemplo anterior, especificamos las opciones `n` y `r` para realizar una ordenación numérica inversa y especificamos `-k 5` para hacer que la ordenación utilice el quinto campo como clave para la ordenación.

La opción `k` es muy interesante y tiene muchas características, pero primero tenemos que hablar de cómo `sort` define los campos. Consideremos un archivo de texto muy simple que consta de una sola línea que contiene el nombre del autor:

---

Guillermo Shotts

---

De forma predeterminada, ordenar ve esta línea como si tuviera dos campos. El primer campo contiene los caracteres `Guillermo` y el segundo campo contiene los caracteres `Shotts`, lo que significa que los caracteres de espacio en blanco (espacios y tabulaciones) se utilizan como delimitadores entre campos y que los delimitadores se incluyen en el campo cuando se realiza la ordenación.

Mirando de nuevo una línea de nuestra salida `ls`, podemos ver que una línea contiene ocho campos y que el quinto campo es el tamaño del archivo:

---

```
-rwxr-xr-x 1 raíz  raíz   8234216 07/04/2012 17:42 inkscape
```

---

Para nuestra próxima serie de experimentos, consideremos el siguiente archivo que contiene la historia de tres distribuciones populares de Linux lanzadas entre 2006 y 2008. Cada línea del archivo tiene tres campos: el nombre de la distribución, el número de versión y la fecha de lanzamiento en formato MM/DD/AAAA:

---

SUSE	10.2	12/07/2006
Fedora	10	11/25/2008
SUSE	11.0	06/19/2008
Ubuntu	8.04	04/24/2008
Fedora	8	11/08/2007
SUSE	10.3	10/04/2007
Ubuntu	6.10	10/26/2006

---

Fedora	7	05/31/2007
Ubuntu	7.10	10/18/2007
Ubuntu	7.04	04/19/2007
SUSE	10.1	05/11/2006
Fedora	6	10/24/2006

---

Fedora	9	05/13/2008
Ubuntu	6.06	06/01/2006
Ubuntu	8.10	10/30/2008
Fedora	5	03/20/2006

---

Usando un editor de texto (tal vez vim), ingresaremos estos datos y nombraremos el archivo resultante *distros.txt*.

A continuación, intentaremos ordenar el archivo y observar los resultados:

---

```
[me@linuxbox ~]$ ordenar distros.txt
Fedora      10    11/25/2008
Fedora      5     03/20/2006
Fedora      6     10/24/2006
Fedora      7     05/31/2007
Fedora      8     11/08/2007
Fedora      9     05/13/2008
SUSE        10.1   05/11/2006
SUSE        10.2   12/07/2006
SUSE        10.3   10/04/2007
SUSE        11.0   06/19/2008
Ubuntu     6.06   06/01/2006
Ubuntu     6.10   10/26/2006
Ubuntu     7.04   04/19/2007
Ubuntu     7.10   10/18/2007
Ubuntu     8.04   04/24/2008
Ubuntu     8.10   10/30/2008
```

---

Bueno, la mayoría de las veces funcionó. El problema ocurre en la clasificación de los números de versión de Fedora. Dado que un 1 va antes de un 5 en el conjunto de caracteres, la versión 10 termina en la parte superior, mientras que la versión 9 cae en la parte inferior.

Para solucionar este problema, tenemos que ordenar varias claves. Queremos realizar una ordenación alfabética en el primer campo y luego una ordenación numérica en el tercer campo. sort permite varias instancias de la opción -k para que se puedan especificar varias claves de ordenación. De hecho, una clave puede incluir un rango de campos. Si no se especifica ningún rango (como ha sido el caso con nuestros ejemplos anteriores), sort usa una clave que comienza con el campo especificado y se extiende hasta el final de la línea.

Esta es la sintaxis de nuestra ordenación multclave:

---

```
[me@linuxbox ~]$ sort --key=1,1 --key=2n distros.txt
Fedora      5     03/20/2006
Fedora      6     10/24/2006
Fedora      7     05/31/2007
Fedora      8     11/08/2007
Fedora      9     05/13/2008
Fedora     10    11/25/2008
SUSE        10.1   05/11/2006
SUSE        10.2   12/07/2006
SUSE        10.3   10/04/2007
SUSE        11.0   06/19/2008
Ubuntu     6.06   06/01/2006
Ubuntu     6.10   10/26/2006
Ubuntu     7.04   04/19/2007
Ubuntu     7.10   10/18/2007
Ubuntu     8.04   04/24/2008
```

---



Aunque usamos la forma larga de la opción para mayor claridad, `-k 1,1 -k 2n` sería exactamente equivalente. En la primera instancia de la opción de clave, especificamos una serie de campos para incluir en la primera clave. Dado que queríamos limitar la ordenación solo al primer campo, especificamos `1,1`, que significa "comenzar en el campo 1 y terminar en el campo 1". En el segundo caso, especificamos `2n`, lo que significa que el campo 2 es la clave de ordenación y que la ordenación debe ser numérica. Se puede incluir una letra de opción al final de un especificador de clave para indicar el tipo de ordenación que se va a realizar. Estas letras de opción son las mismas que las opciones globales para el programa de ordenación : `b` (ignorar espacios en blanco iniciales), `n` (ordenación numérica), `r` (ordenación inversa), etc.

El tercer campo de nuestra lista contiene una fecha en un formato inconveniente para ordenar. En las computadoras, las fechas generalmente se formatean en orden AAAA-MM-DD para facilitar la clasificación cronológica, pero las nuestras están en el formato estadounidense MM/DD/AAAA. ¿Cómo podemos ordenar esta lista en orden cronológico?

Afortunadamente, `sort` proporciona una manera. La opción de clave permite la especificación de *offsets* dentro de los campos, por lo que podemos definir claves dentro de los campos:

---

```
[me@linuxbox ~]$ sort -k 3.7nbr -k 3.1nbr -k 3.4nbr distros.txt
Fedora      10    11/25/2008
Ubuntu      8.10   10/30/2008
SUSE        11.0   06/19/2008
Fedora      9     05/13/2008
Ubuntu      8.04   04/24/2008
Fedora      8     11/08/2007
Ubuntu      7.10   10/18/2007
SUSE        10.3   10/04/2007
Fedora      7     05/31/2007
Ubuntu      7.04   04/19/2007
SUSE        10.2   12/07/2006
Ubuntu      6.10   10/26/2006
Fedora      6     10/24/2006
Ubuntu      6.06   06/01/2006
SUSE        10.1   05/11/2006
Fedora      5     03/20/2006
```

---

Al especificar `-k 3.7`, le indicamos a `sort` que use una clave de ordenación que comience en el séptimo carácter dentro del tercer campo, que corresponde al comienzo del año. Del mismo modo, especificamos `-k 3.1` y `-k 3.4` para aislar las partes del mes y el día de la fecha. También agregamos las opciones `n` y `r` para lograr una ordenación numérica inversa. La opción `b` se incluye para suprimir los espacios iniciales (cuyos números varían de una línea a otra, lo que afecta al resultado de la ordenación) en el campo de fecha.

Algunos archivos no usan tabuladores y espacios como delimitadores de campo; tomemos, por ejemplo, el archivo `/etc/passwd`:

---

```
[me@linuxbox ~]$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

---

```
bin:x:2:2:bin:/bin:/bin/sh  
sys:x:3:3:sys:/dev:/bin/sh  
sync:x:4:65534:sync:/bin:/bin/sync  
games:x:5:60:games:/usr/games:/bin/sh  
man:x:6:12:man:/var/cache/man:/bin/sh
```

```
lp:x:7:lp:/var/spool/lpd:/bin/sh  
correo:x:8:8:correo:/var/correo:/bin/sh  
noticias:x:9:9:noticias:/var/carrete/noticias:/bin/sh
```

---

Los campos de este archivo están delimitados con dos puntos (:), entonces, ¿cómo ordenaríamos este archivo usando un campo clave? sort proporciona la opción -t para definir el carácter del separador de campos. Para ordenar el archivo passwd en el séptimo campo (el shell predeterminado de la cuenta), podríamos hacer lo siguiente:

```
[me@linuxbox ~]$ sort -t ':' -k 7 /etc/passwd | head  
me:x:1001:1001:Myself,,,:/home/me:/bin/bash  
root:x:0:0:root:/root:/bin/bash  
dhcpc:x:101:102::/nonexistt:/bin/false  
gdm:x:106:114:Administrador de pantalla de Gnome:/var/lib/gdm:/bin/false  
hplip:x:104:7:Sistema HPLIP usuario,,,:/var/run/hplip:/bin/false  
klog:x:103:104::/home/klog:/bin/false  
messagebus:x:108:119::/var/run/dbus:/bin/false  
polkituser:x:110:122:PolicyKit,,,:/var/run/PolicyKit:/bin/false  
pulse:x:107:116:PulseAudio daemon,,,:/var/run/pulse:/bin/false
```

---

Al especificar el carácter de dos puntos como separador de campo, podemos ordenar en el séptimo campo.

### ***uniq: informar u omitir líneas repetidas***

En comparación con sort, el programa uniq es liviano. UNIQ lleva a cabo una tarea aparentemente trivial. Cuando se le da un archivo ordenado (incluida la entrada estándar), elimina las líneas duplicadas y envía los resultados a la salida estándar. A menudo se utiliza junto con la ordenación para limpiar la salida de duplicados.

**Nota:** Si bien uniq es una herramienta tradicional de Unix que se usa a menudo con sort, la versión GNU de sort Admite una opción -u, que elimina los duplicados de la salida ordenada.

Vamos a hacer un archivo de texto para probar esto:

```
[me@linuxbox ~]$ gato > foo.txt  
un  
b  
c  
a  
b  
c
```

---

Recuerde escribir CTRL-D para finalizar la entrada estándar. Ahora, si ejecutamos uniq en nuestro archivo de texto, los resultados no son diferentes de nuestro archivo original; los duplicados no se eliminaron:

```
[me@linuxbox ~]$ uniq foo.txt  
a  
b  
c  
a  
b  
c
```

---



Para que uniq realmente haga su trabajo, la entrada debe ordenarse primero:

---

```
[me@linuxbox ~]$ ordenar foo.txt | uniq  
a  
b  
c
```

---

Esto se debe a que uniq solo elimina las líneas duplicadas que son adyacentes entre sí.

Uniq tiene varias opciones. En la tabla 20-2 se enumeran los más comunes.

Tabla 20-2: Opciones comunes de uniq

Opción	Descripción
-c	Genera una lista de líneas duplicadas precedidas por el número de veces que aparece la línea.
-d	Genere solo líneas repetidas, en lugar de líneas únicas.
-f <i>n</i>	Ignorar <i>n</i> campos iniciales en cada línea. Los campos están separados por espacios en blanco, ya que están en ordenar; Sin embargo, a diferencia de ordenar, Uniq tiene No hay opción para establecer un separador de campo alternativo.
-Yo	Ignore las mayúsculas y minúsculas durante las comparaciones de líneas.
-s <i>n</i>	Omitir (ignorar) el interlineado <i>n</i> caracteres de cada línea.
-u	Salida solo de líneas únicas. Este es el valor predeterminado.

Aquí vemos que uniq se usa para informar el número de duplicados encontrados en nuestro archivo de texto, usando la opción -c:

---

```
[me@linuxbox ~]$ ordenar foo.txt | uniq -c  
2 A  
2 ter  
2 C
```

---

## Cortar en rodajas y cubitos

Los siguientes tres programas que discutiremos se utilizan para despegar columnas de texto de archivos y recombinarlos de manera útil.

### *cut: eliminar secciones de cada línea de archivos*

El programa de corte se utiliza para extraer una sección de texto de una línea y enviar la sección extraída a la salida estándar. Puede aceptar múltiples argumentos de archivo o entradas de entrada estándar.

La especificación de la sección de la línea que se va a extraer es algo incómoda y se especifica utilizando las opciones que se muestran en la Tabla

20-3.

Tabla 20-3: Opciones de selección de corte

Opción	Descripción
<code>-c char_list</code>	Extraiga la parte de la línea definida por <code>char_list</code> . La lista puede constar de uno o más rangos numéricos separados por comas.
<code>-f field_list</code>	Extraiga uno o más campos de la línea definida por <code>field_list</code> . La lista puede contener uno o más campos o Rangos de campos separados por comas.
<code>-d delim_charWhen</code>	<code>-f</code> se especifica, use <code>delim_char</code> como el carácter delimitador del campo. De forma predeterminada, los campos deben estar separados por un Carácter de una sola tabulación.
<code>--complemento</code>	Extraiga toda la línea de texto, excepto las partes especificado por <code>-c</code> y/o <code>-f</code> .

Como podemos ver, la forma en que el corte extrae el texto es bastante inflexible. Cut se utiliza mejor para extraer texto de archivos producidos por otros programas, en lugar de texto escrito directamente por humanos. Echaremos un vistazo a nuestro *archivo de distros.txt* para ver si está lo suficientemente "limpio" como para ser un buen espécimen para nuestros ejemplos de corte. Si usamos cat con la opción -A, podemos ver si el archivo cumple con nuestros requisitos de campos separados por tabulaciones.

```
[me@linuxbox ~]$ gato -A distros.txt
SUSE^I10.2^I12/07/2006$
Fedora^I10^I11/25/2008$
SUSE^I11.0^I06/19/2008$
Ubuntu^I8.04^I04/24/2008$
Fedora^I8^I11/08/2007$
SUSE^I10.3^I10/04/2007$
Ubuntu^I6.10^I10/26/2006$
Fedora^I7^I05/31/2007$
Ubuntu^I7.10^I10/18/2007$
Ubuntu^I7.04^I04/19/2007$
SUSE^I10.1^I05/11/2006$
Fedora^I6^I10/24/2006$
Fedora^I9^I05/13/2008$
Ubuntu^I6.06^I06/01/2006$
Ubuntu^I8.10^I10/30/2008$
Fedora^I5^I03/20/2006$
```

Se ve bien, sin espacios incrustados, solo caracteres de tabulación individuales entre los campos. Dado que el archivo usa tabulaciones en lugar de espacios, usaremos la opción `-f` para extraer un campo:

```
[me@linuxbox ~]$ corte -f 3 distros.txt
12/07/2006
11/25/2008
06/19/2008
```

04/24/2008  
11/08/2007

---

```
10/04/2007
10/26/2006
05/31/2007
10/18/2007
04/19/2007
05/11/2006
10/24/2006
05/13/2008
06/01/2006
10/30/2008
03/20/2006
```

---

Debido a que nuestro *archivo de distribución* está delimitado por tabulaciones, es mejor usar cut para extraer campos en lugar de caracteres. Esto se debe a que cuando un archivo está delimitado por tabulaciones, es poco probable que cada línea contenga el mismo número de caracteres, lo que dificulta o imposibilita el cálculo de las posiciones de los caracteres dentro de la línea. En nuestro ejemplo anterior, sin embargo, ahora hemos extraído un campo que afortunadamente contiene datos de idéntica longitud, por lo que podemos mostrar cómo funciona la extracción de caracteres extrayendo el año de cada línea:

---

```
[me@linuxbox ~]$ corte -f 3 distros.txt | corte -c 7-10
2006
2008
2008
2008
2007
2007
2006
2007
2007
2007
2006
2006
2008
2006
2008
2006
2006
```

---

Al ejecutar corte por segunda vez en nuestra lista, podemos extraer las posiciones de carácter 7 a 10, que corresponden al año en nuestro campo de fecha. La notación 7-10 es un ejemplo de un rango. La página del comando man cut contiene una descripción completa de cómo se pueden especificar los intervalos.

Al trabajar con campos, es posible especificar un delimitador de campo diferente en lugar del carácter de tabulación. Aquí extraeremos el primer campo del archivo /etc/passwd:

---

```
[me@linuxbox ~]$ cut -d ':' -f 1 /etc/passwd | head
root
daemon
bin
sys
sync
games
man
```

Usando la opción `-d`, podemos especificar el carácter de dos puntos como el delimitador de campo.

## EXPANSIÓN DE PESTAÑAS

Nuestro archivo `distros.txt` tiene un formato ideal para extraer campos mediante corte. Pero, ¿qué pasaría si quisieramos un archivo que pudiera manipularse completamente con corte por caracteres, en lugar de campos? Esto requeriría que reemplazáramos los caracteres de tabulación dentro del archivo con el número correspondiente de espacios. Afortunadamente, el paquete GNU coreutils incluye una herramienta para ello. Con el nombre de `expand`, este programa acepta uno o más argumentos de archivo o entrada estándar, y envía el texto modificado a la salida estándar.

Si procesamos nuestro archivo `distros.txt` con `expand`, podemos usar el corte `-c` para extraer cualquier rango de caracteres del archivo. Por ejemplo, podríamos usar el siguiente comando para extraer el año de lanzamiento de nuestra lista expandiendo el archivo y usando `cut` para extraer cada carácter desde la posición 23 hasta el final de la línea:

```
[me@linuxbox ~]$ expandir distros.txt | cortar -c 23-
```

coreutils también proporciona el programa `Unexpand` para sustituir pestañas por espacios.

## pegar: fusionar líneas de archivos

El comando `pegar` hace lo contrario que `cortar`. En lugar de extraer una columna de texto de un archivo, agrega una o más columnas de texto a un archivo. Para ello, lee varios archivos y combina los campos que se encuentran en cada archivo en un único flujo de salida estándar. Al igual que `cortar`, `pegar` acepta

Múltiples argumentos de archivo y/o entrada estándar. Para demostrar cómo funciona el `pegado`, realizaremos una cirugía en nuestro archivo de `distros.txt` para producir una lista cronológica de lanzamientos.

A partir de nuestro trabajo anterior con `sort`, primero produciremos una lista de distribuciones ordenadas por fecha y almacenaremos el resultado en un archivo llamado `distros-by-date.txt`:

```
[me@linuxbox ~]$ sort -k 3.7nbr -k 3.1nbr -k 3.4nbr distros.txt > distribuciones-por- date.txt
```

A continuación, usaremos `cut` para extraer los dos primeros campos del archivo (el nombre de la distribución y la versión) y almacenaremos ese resultado en un archivo llamado `distro-versions.txt`:

```
[me@linuxbox ~]$ corte -f 1,2 distros-by-date.txt > distros-versions.txt
```

```
[me@linuxbox ~]$ cabeza distros-versions.txt
```

---

Fedora	10
Ubuntu	8.10
SUSE	11.0
Fedora	9
Ubuntu	8.04
Fedora	8
Ubuntu	7.10
SUSE	10.3
Fedora	7
Ubuntu	7.04

---

La última parte de la preparación es extraer las fechas de lanzamiento y almacenarlas en un archivo llamado *distro-dates.txt*:

---

```
[me@linuxbox ~]$ corte -f 3 distros-by-date.txt > distros-dates.txt
[me@linuxbox ~]$ cabeza distros-dates.txt
11/25/2008
10/30/2008
06/19/2008
05/13/2008
04/24/2008
11/08/2007
10/18/2007
10/04/2007
05/31/2007
04/19/2007
```

---

Ahora tenemos las piezas que necesitamos. Para completar el proceso, use pegar para poner la columna de fechas antes de los nombres y versiones de la distribución, creando así una lista cronológica. Esto se hace simplemente usando pegar y ordenando sus argumentos en la disposición deseada.

---

```
[me@linuxbox ~]$ pegar distros-dates.txt distros-versions.txt
11/25/2008    Fedora    10
10/30/2008    Ubuntu    8.10
06/19/2008    SUSE      11.0
05/13/2008    Fedora    9
04/24/2008    Ubuntu    8.04
11/08/2007    Fedora    8
10/18/2007    Ubuntu    7.10
10/04/2007    SUSE      10.3
05/31/2007    Fedora    7
04/19/2007    Ubuntu    7.04
12/07/2006    SUSE      10.2
10/26/2006    Ubuntu    6.10
10/24/2006    Fedora    6
06/01/2006    Ubuntu    6.06
05/11/2006    SUSE      10.1
03/20/2006    Fedora    5
```

---

### *join: unir líneas de dos archivos en un campo común*

De alguna manera, join es como pegar en el sentido de que agrega columnas a un archivo, pero lo hace de una manera única. Una *unión* es una operación generalmente asociada con *bases de datos relacionales* donde los datos de múltiples *tablas* con un campo clave compartido se combinan para

forma un resultado deseado. El programa de unión realiza la misma operación. Une datos de varios archivos en función de un campo clave compartido.

Para ver cómo se utiliza una operación de unión en una base de datos relacional, imaginemos una base de datos muy pequeña que consta de dos tablas, cada una de las cuales contiene un único registro. La primera tabla, llamada CUSTOMERS, tiene tres campos: un número de cliente (CUSTNUM), el nombre del cliente (FNAME) y el apellido del cliente (LNAME):

CUSTNUM	FNAME	LNAME
=====	=====	=====
4681934	John	Herrero

La segunda tabla se denomina PEDIDOS y contiene cuatro campos: un número de pedido (ORDERNUM), el número de cliente (CUSTNUM), la cantidad (QUAN) y el artículo pedido (ITEM):

ORDERNUM	CUSTNUM	QUAN	ARTÍCULO
=====	=====	====	====
3014953305	4681934	1	Widget azul

Tenga en cuenta que ambas tablas comparten el campo CUSTNUM. Esto es importante, ya que permite una relación entre las tablas.

Realizar una operación de unión nos permitiría combinar los campos de las dos tablas para conseguir un resultado útil, como por ejemplo preparar una factura. Con los valores coincidentes de los campos CUSTNUM de ambas tablas, una operación de combinación podría producir lo siguiente:

FNAME	LNAME	QUAN	ARTÍCULO
=====	=====	====	====
John	Herrero	1	Widget azul

Para demostrar el programa de unión, necesitaremos crear un par de archivos con una clave compartida. Para ello, utilizaremos nuestro archivo *distros-by-date.txt*. A partir de este archivo, construiremos dos archivos adicionales. Uno contiene las fechas de lanzamiento (que será nuestro campo clave compartido para esta demostración) y los nombres de los lanzamientos:

```
[me@linuxbox ~]$ corte -f 1,1 distros-by-date.txt > distros-names.txt
[me@linuxbox ~]$ pegar distros-dates.txt distros-names.txt > distros-key-names
.Txt
[me@linuxbox ~]$ cabeza distros-key-names.txt
11/25/2008      Fedora
10/30/2008      Ubuntu
06/19/2008      SUSE
05/13/2008      Fedora
04/24/2008      Ubuntu
11/08/2007      Fedora
```

10/18/2007	Ubuntu
10/04/2007	SUSE
05/31/2007	Fedora
04/19/2007	Ubuntu

El segundo archivo contiene las fechas de lanzamiento y los números de versión:

```
[me@linuxbox ~]$ cut -f 2,2 distros-by-date.txt > distros-vernums.txt [me@linuxbox
~]$ pegar distros-dates.txt distros-vernums.txt > distros-key- vernums.txt
[me@linuxbox ~]$ cabeza distros-key-vernums.txt
11/25/2008      10
10/30/2008      8.10
06/19/2008      11.0
05/13/2008       9
04/24/2008      8.04
11/08/2007        8
10/18/2007      7.10
10/04/2007      10.3
05/31/2007         7
04/19/2007      7.04
```

Ahora tenemos dos archivos con una clave compartida (el campo "fecha de lanzamiento"). Es importante señalar que los archivos deben estar ordenados en el campo clave para que la unión funcione correctamente.

```
[me@linuxbox ~]$ unirse a distros-key-names.txt distros-key-vernums.txt | cabeza
11/25/2008 Fedora 10
10/30/2008 Ubuntu 8.10
06/19/2008 SUSE 11.0
05/13/2008 Fedora 9
04/24/2008 Ubuntu 8.04
11/08/2007 Fedora 8
10/18/2007 Ubuntu 7.10
10/04/2007 SUSE 10.3
05/31/2007 Fedora 7
04/19/2007 Ubuntu 7.04
```

Tenga en cuenta también que, de forma predeterminada, la combinación utiliza un espacio en blanco como delimitador de campo de entrada y un solo espacio como delimitador de campo de salida. Este comportamiento se puede modificar especificando opciones. Consulte la página del comando `man join` para obtener más información.

## Comparación de texto

A menudo es útil comparar versiones de archivos de texto. Para los administradores de sistemas y los desarrolladores de software, esto es particularmente importante. Un administrador de sistemas puede, por ejemplo, necesitar comparar un archivo de configuración existente con una versión anterior para diagnosticar un problema del sistema. Del mismo modo, un programador necesita ver con frecuencia qué cambios se han realizado en los programas a lo largo del tiempo.

### *comm: comparar dos archivos ordenados línea por línea*

El programa de comunicaciones compara dos archivos de texto, mostrando las líneas que son únicas para cada uno y las líneas que tienen en común. Para demostrarlo, crearemos dos archivos de texto casi idénticos usando

cat:

---

```
[me@linuxbox ~]$ gato > file1.txt
un
b
```

```
c  
d  
[me@linuxbox ~]$ gato > file2.txt  
b  
c  
d  
e
```

---

A continuación, compararemos los dos archivos usando

```
comm: [me@linuxbox ~]$ comm file1.txt file2.txt  
un  
b  
c  
d  
e
```

---

Como podemos ver, comm produce tres columnas de salida. La primera columna contiene líneas únicas para el primer argumento de archivo; la segunda columna, las líneas únicas del segundo argumento del archivo; y la tercera columna, las líneas compartidas por ambos archivos. comm admite opciones en la forma *-n*, donde *n* es 1, 2 o 3. Cuando se utilizan, estas opciones especifican las columnas que se van a suprimir. Por ejemplo, si quisieramos generar solo las líneas compartidas por ambos archivos, suprimiríamos la salida de las columnas 1 y 2:

```
[me@linuxbox ~]$ comm -12 file1.txt file2.txt  
b  
c  
d
```

---

### ***diff: comparar archivos línea por línea***

Al igual que el programa de comunicaciones, diff se utiliza para detectar las diferencias entre archivos. Sin embargo, diff es una herramienta mucho más compleja, que admite muchos formatos de salida y la capacidad de procesar grandes colecciones de archivos de texto a la vez. Los desarrolladores de software utilizan a menudo DIFF para examinar los cambios entre las diferentes versiones del código fuente del programa porque tiene la capacidad de examinar recursivamente los directorios del código fuente, a menudo denominados árboles de fuentes. Un uso común de diff es la creación de *archivos diff* o *parches* que son utilizados por programas como patch (del que hablaremos en breve) para convertir una versión de un archivo (o archivos) a otra versión.

Si usamos diff para mirar nuestros archivos de ejemplo anteriores, vemos su estilo de salida predeterminado: una descripción concisa de las diferencias entre los dos archivos.

```
[me@linuxbox ~]$ diff file1.txt file2.txt
```

---

1d0  
<  
un  
4a  
4  
> e

En el formato predeterminado, cada grupo de cambios está precedido por un *comando* de cambio (consulte la Tabla 20-4) en forma de rango de *operación para* describir las posiciones y los tipos de cambios necesarios para convertir el primer archivo en el segundo archivo.

**Tabla 20-4: Comandos de cambio de diferencias**

Cambio	Descripción
<i>r1ar2</i>	Anexe las líneas en la posición <i>R2</i> en el segundo archivo a la posición <i>R1</i> en el primer archivo.
<i>R1CR2</i>	Cambiar (reemplazar) las líneas en la posición <i>R1</i> con las líneas en la posición <i>R2</i> en el segundo archivo.
<i>R1DR2</i>	Borra las líneas del primer archivo en la posición <i>R1</i> , que habría aparecido a distancia <i>R2</i> en el segundo archivo

En este formato, un rango es una lista separada por comas de la línea inicial y la línea final. Si bien este formato es el predeterminado (principalmente para el cumplimiento de POSIX y la compatibilidad con versiones anteriores de Unix de diff), no es tan ampliamente utilizado como otros formatos opcionales. Dos de los formatos más populares son el *formato de contexto* y el *formato unificado*.

Cuando se ve con el formato de contexto (la opción *-c*), la salida tiene el siguiente aspecto:

---

```
[me@linuxbox ~]$ diff -c file1.txt file2.txt
file1.txt      2012-12-23 06:40:13.000000000 -0500
--- file2.txt      2012-12-23 06:40:34.000000000 -0500
*****
*** 1,4 ****
- a
 b
 c
 d
--- 1,4 -----
 b

c
d
+ e
```

---

La salida comienza con los nombres de los dos archivos y sus marcas de tiempo.

El primer archivo está marcado con asteriscos y el segundo archivo está marcado con guiones. A lo largo del resto del listado, estos marcadores significarán su archivos respectivos. A continuación, vemos grupos de cambios, incluido el número predeterminado de líneas de contexto circundantes. En el primer grupo, vemos \*\*\* 1,4 \*\*\*, que indica las líneas 1 a 4 en el primer archivo. Más adelante vemos --- 1,4 -----, lo que indica las líneas 1 a 4 en el segundo archivo. Dentro de un grupo de cambio, las líneas comienzan con uno de los cuatro indicadores, como se muestra en la Tabla 20-5.



Tabla 20-5: Indicadores de cambio de formato de contexto

Indicador	Significado
(Ninguno)	Una línea que se muestra para contextualizar. No indica una diferencia entre los dos archivos.
-	Una línea eliminada. Esta línea aparecerá en el primer archivo, pero no en el segundo.
+	Se añadió una línea. Esta línea aparecerá en el segundo archivo, pero no en el primero.
!	Una línea cambió. Se mostrarán las dos versiones de la línea, cada una en su sección respectiva del grupo de cambios.

El formato unificado es similar al formato de contexto, pero es más conciso.

Se especifica con la opción -u:

```
[me@linuxbox ~]$ diff -u file1.txt file2.txt
--- file1.txt      2012-12-23 06:40:13.000000000 -0500
+++ file2.txt      2012-12-23 06:40:34.000000000 -0500
@@ -1,4 +1,4 @@
-a
 b
 c
 d
+e
```

La diferencia más notable entre el formato de contexto y el unificado es la eliminación de las líneas duplicadas de contexto, lo que hace que los resultados del formato unificado sean más cortos que los del formato de contexto. En nuestro ejemplo anterior, vemos marcas de tiempo de archivo como las del formato de contexto, seguidas de la cadena @@ -1,4 +1,4 @@. Esto indica las líneas del primer archivo y las líneas del segundo archivo descritas en el grupo de cambios. A continuación están las líneas propiamente dichas, con las tres líneas de contexto predeterminadas. Como se muestra en la Tabla 20-6, cada línea comienza con uno de los tres caracteres posibles.

Tabla 20-6: Indicadores de cambio de formato unificado diff

Carácter	Significado
(Ninguno)	Esta línea es compartida por ambos archivos.
-	Esta línea se eliminó del primer archivo.
+	Esta línea se agregó al primer archivo.

## ***patch: aplicar una diferencia a un original***

El programa de parches se utiliza para aplicar cambios a los archivos de texto. Acepta la salida de diff y generalmente se usa para convertir versiones anteriores de archivos en versiones más nuevas. Consideremos un ejemplo famoso. El kernel de Linux es desarrollado por un equipo grande y poco organizado de colaboradores que envían un flujo constante de pequeños cambios al código fuente. El kernel de Linux consta de varios millones de líneas de código, mientras que los cambios que realiza un contribuyente a la vez son bastante pequeños. No tiene sentido que un colaborador envíe a cada desarrollador un árbol de fuentes del kernel completo cada vez que se realiza un pequeño cambio. En su lugar, se envía un archivo diff. El fichero diff contiene el cambio de la versión anterior del kernel a la nueva versión con los cambios del contribuyente. A continuación, el receptor utiliza el programa de parches para aplicar el cambio a su propio árbol de fuentes. El uso de diff/patch ofrece dos ventajas significativas:

- El archivo diff es muy pequeño, en comparación con el tamaño completo del árbol de fuentes.
- El archivo diff muestra de forma concisa el cambio que se está realizando, lo que permite a los revisores del parche evaluarlo rápidamente.

Por supuesto, diff/patch funcionará en cualquier archivo de texto, no solo en el código fuente. Sería igualmente aplicable a los archivos de configuración o a cualquier otro texto.

Para preparar un fichero diff para su uso con patch, la documentación de GNU sugiere usar diff de la siguiente manera:

```
diff -Naur old_file new_file > diff_file
```

donde *old\_file* y *new\_file* son archivos únicos o directorios que contienen archivos. La opción r admite la recursividad de un árbol de directorios.

Una vez que se ha creado el archivo diff, podemos aplicarlo para parchear el archivo antiguo en el nuevo archivo:

```
Parche < diff_file
```

Demostraremos con nuestro archivo de prueba:

---

```
[me@linuxbox ~]$ diff -Naur file1.txt file2.txt > patchfile.txt
[me@linuxbox ~]$ parche < patchfile.txt
archivo de parcheo file1.txt
[me@linuxbox ~]$ cat file1.txt b
c
d
e
```

---

En este ejemplo, creamos un archivo de diferencias llamado *patchfile.txt* y luego usamos el programa de parches para aplicar el parche. Tenga en cuenta que no tuvimos que especificar un archivo de destino para parchear, ya que el archivo diff (en formato unificado) ya contiene los nombres de archivo en el encabezado. Una vez aplicado el parche,

podemos ver que *file1.txt* ahora coincide con *file2.txt*.

patch tiene un gran número de opciones, y se pueden utilizar programas de utilidad adicionales para analizar y editar parches.

## Edición sobre la marcha

Nuestra experiencia con los editores de texto ha sido en gran medida *interactiva*, lo que significa que movemos manualmente un cursor y luego escribimos nuestros cambios. Sin embargo, también hay *formas no interactivas* de editar texto. Es posible, por ejemplo, aplicar un conjunto de cambios a varios archivos con un solo comando.

### *tr: transliterar o eliminar caracteres*

El programa tr se utiliza para *transliterar* caracteres. Podemos pensar en esto como una especie de operación de búsqueda y reemplazo basada en caracteres. La transliteración es el proceso de cambiar caracteres de un alfabeto a otro. Por ejemplo, la conversión de caracteres de minúsculas a mayúsculas es transliteración. Podemos realizar dicha conversión con tr de la siguiente manera:

---

```
[me@linuxbox -]$ echo "letras minúsculas" | tr a-z A-Z
LETRAS MINÚSCULAS
```

---

Como podemos ver, tr funciona con una entrada estándar y emite sus resultados con una salida estándar. TR acepta dos argumentos: un conjunto de caracteres desde los que convertir y un conjunto correspondiente de caracteres a los que convertir. Los conjuntos de caracteres se pueden expresar de una de estas tres maneras:

- Una lista enumerada; por ejemplo, ABCDEFGHIJKLMNOPQRSTUVWXYZ.
- Un intervalo de caracteres; por ejemplo, de la A a la Z. Tenga en cuenta que este método a veces está sujeto a los mismos problemas que otros comandos (debido al orden de recopilación de la configuración regional) y, por lo tanto, debe usarse con precaución.
- Clases de caracteres POSIX; por ejemplo, [:upper:].

En la mayoría de los casos, los conjuntos de caracteres deben tener la misma longitud; Sin embargo, es posible que el primer conjunto sea mayor que el segundo, especialmente si deseamos convertir varios caracteres en un solo carácter:

---

```
[me@linuxbox -]$ echo "letras minúsculas" | tr [:lower:] A
AAAAAAAAA AAAAAAA
```

---

Además de la transliteración, tr permite que los caracteres simplemente se eliminen del flujo de entrada. Anteriormente en este capítulo, discutimos el problema de convertir archivos de texto de MS-DOS a texto de estilo Unix. Para realizar esta conversión, los caracteres de retorno de carro deben eliminarse del final de cada línea. Esto se puede realizar con tr de la siguiente manera:

```
tr -d '\r' < dos_file > unix_file
```

donde *dos\_file* es el archivo a convertir y *unix\_file* es el resultado. Esta forma del comando utiliza la secuencia de escape \r para representar el carácter de retorno de carro. Para ver una lista completa de las secuencias y clases de caracteres que admite tr, intente

---

```
[me@linuxbox ~]$ tr -ayuda
```

---

## RO T13: EL ANILLO DECODIFICADOR NO TAN SECRETO

Un uso divertido de tr es realizar la *codificación de texto ROT13*. ROT13 es un tipo trivial de cifrado basado en un cifrado de sustitución simple. Llamar a la *encriptación* ROT13 es ser generoso; La *ofuscación de texto* es más precisa. A veces se utiliza en el texto para ocultar contenido potencialmente ofensivo. El método simplemente mueve cada carácter 13 lugares hacia arriba en el alfabeto. Dado que esto está a la mitad de los 26 caracteres posibles, al realizar el algoritmo por segunda vez en el texto, se restaura a su forma original. Para realizar esta codificación con tr:

```
Echo "Texto secreto" | tr a-zA-Z n-za-mN-ZA-M  
frperg grkg
```

Al realizar el mismo procedimiento por segunda vez, se obtiene la traducción:

```
Eco "Frperg GRKG" | tr a-zA-Z n-za-mN-ZA-M  
Texto secreto
```

Varios programas de correo electrónico y lectores de noticias de Usenet

TR también puede realizar otro truco. Usando la opción *-s*, tr puede "exprimir" (eliminar) instancias repetidas de un carácter:

---

```
[me@linuxbox ~]$ echo "aaabbbccc" | tr -s ab  
ABC
```

---

Aquí tenemos una cadena que contiene caracteres repetidos. Al especificar el conjunto ab a tr, eliminamos las instancias repetidas de las letras en el conjunto, mientras que deja el carácter que falta en el conjunto (c) sin cambios. Tenga en cuenta que los caracteres repetidos deben estar contiguos. Si no lo están, el apretón no tendrá ningún efecto:

---

```
[me@linuxbox ~]$ echo "abcabcabc" | tr -s ab  
Abcabcbc
```

---

## ***sed: editor de secuencias para filtrar y transformar texto***

El nombre sed es la abreviatura de *editor de transmisión*. Realiza la edición de texto en una secuencia de texto, ya sea un conjunto de archivos especificados o una entrada estándar. SED es un programa potente y algo complejo (hay libros enteros sobre él), por lo que no lo cubriremos completamente aquí.

En general, la forma en que funciona sed es que se le da un solo comando de edición (en la línea de comandos) o el nombre de un archivo de script que contiene varios comandos, y luego realiza estos comandos en cada línea en el flujo de texto. Aquí hay un ejemplo muy simple de sed en acción:

---

```
[me@linuxbox ~]$ echo "frente" | sed 's/frente/atrás/'  
Atrás
```

---

En este ejemplo, producimos un flujo de texto de una palabra usando echo y lo canalizamos a sed. sed, a su vez, lleva a cabo la instrucción *s/front/back/* sobre el texto en la secuencia y produce la salida como resultado. También podemos reconocer que este comando se asemeja al comando de sustitución (buscar y reemplazar) en vi.

Los comandos en sed comienzan con una sola letra. En el ejemplo anterior, el comando de sustitución se representa con la letra *s* y va seguido de las cadenas de búsqueda y reemplazo, separadas por el carácter de barra diagonal como delimitador. La elección del carácter delimitador es arbitraria. Por convención, el carácter de barra diagonal se usa a menudo, pero sed aceptará cualquier carácter que siga inmediatamente al comando como delimitador. Podríamos realizar la misma com- mación de la siguiente manera:

---

```
[me@linuxbox ~]$ echo "frente" | sed 's_front_back_  
Atrás
```

---

Cuando el carácter de subrayado se utiliza inmediatamente después del comando, se convierte en el delimitador. La capacidad de establecer el delimitador se puede utilizar para hacer que los comandos sean más legibles, como veremos.

La mayoría de los comandos en sed pueden ir precedidos de una *dirección*, que especifica qué línea(s) del flujo de entrada se editarán. Si se omite la dirección, el comando de edición se lleva a cabo en cada línea del flujo de entrada. La forma más simple de dirección es un número de línea. Podemos añadir uno a nuestro ejemplo:

---

```
[me@linuxbox ~]$ echo "frente" | sed '1s/frente/atrás/'  
Atrás
```

---

Agregar la dirección 1 a nuestro comando hace que nuestra sustitución se realice en la primera línea de nuestro flujo de entrada de una línea. Podemos especificar otro número:

---

```
[me@linuxbox ~]$ echo "frente" | sed '2s/frente/atrás/'  
frente
```

---

Ahora vemos que la edición no se lleva a cabo, porque nuestro flujo de entrada no tiene una línea 2.

Las direcciones pueden expresarse de muchas maneras. En la tabla 20-7 se enumeran los más comunes.

**Tabla 20-7: Notación de direcciones sed**

Dirección	Descripción
<i>n</i>	Un número de línea en el que <i>n</i> es un número entero positivo
\$	La última línea
/regexp/	Líneas que coinciden con una expresión regular básica POSIX. Tenga en cuenta que la expresión regular está delimitada por caracteres de barra diagonal. Opcionalmente, la expresión regular puede ser delimitada por un carácter alternativo, especificando la expresión con \cregexpDónde <i>c</i> es el carácter alternativo.
<i>addr1,addr2</i>	Una gama de líneas desde <i>addr1</i> Para <i>addr2</i> inclusivo. Las direcciones pueden ser cualquiera de los formularios de dirección única anteriores.
<i>Primer ~ paso</i>	Coincide con la línea representada por el número <i>Primer</i> y luego cada línea subsiguiente en <i>paso</i> Intervalos. Por ejemplo 1~2 se refiere a cada línea impar, y 5~5 se refiere a la quinta línea y a cada quinta línea a partir de entonces.
<i>addr1,+n</i>	Cerrilla <i>addr1</i> y los siguientes <i>n</i> líneas.
<i>Addr!</i>	Coincidir con todas las líneas excepto <i>Addr</i> , que puede ser cualquiera de las formas anteriores.

Demostraremos diferentes tipos de direcciones utilizando el *archivo distros.txt* de anteriormente en este capítulo. Primero, un rango de números de línea:

```
[me@linuxbox ~]$ sed -n '1,5p' distros.txt
SUSE      10.2   12/07/2006
Fedora    10      11/25/2008
SUSE      11.0   06/19/2008
Ubuntu    8.04   04/24/2008
Fedora    8       11/08/2007
```

En este ejemplo, imprimimos un rango de líneas, comenzando con la línea 1 y continuando hasta la línea 5. Para hacer esto, usamos el comando *p*, que simplemente hace que se imprima una línea coincidente. Sin embargo, para que esto sea efectivo, debemos incluir la opción *-n* (la opción sin impresión automática) para hacer que *sed* no imprima todas las líneas de forma predeterminada.

A continuación, probaremos con una expresión regular:

---

```
[me@linuxbox ~]$ sed -n '/SUSE/p' distros.txt
SUSE      10.2   12/07/2006
SUSE      11.0   06/19/2008
SUSE      10.3   10/04/2007
SUSE      10.1   05/11/2006
```

---

Al incluir la expresión regular delimitada por barras diagonales /SUSE/, podemos aislar las líneas que la contienen de la misma manera que grep.

Finalmente, intentaremos la negación agregando un signo de exclamación (!) a la dirección:

---

```
[me@linuxbox ~]$ sed -n '/!SUSE/p' distros.txt
Fedora    10      11/25/2008
Ubuntu    8.04    04/24/2008
Fedora    8       11/08/2007
Ubuntu    6.10    10/26/2006
Fedora    7       05/31/2007
Ubuntu    7.10    10/18/2007
Ubuntu    7.04    04/19/2007
Fedora    6       10/24/2006
Fedora    9       05/13/2008
Ubuntu    6.06    06/01/2006
Ubuntu    8.10    10/30/2008
Fedora    5       03/20/2006
```

---

Aquí vemos el resultado esperado: todas las líneas del archivo excepto las que coinciden con la expresión regular.

Hasta ahora, hemos visto dos de los comandos de edición sed, s y p. La tabla 20-8 es una lista más completa de los comandos básicos de edición.

**Tabla 20-8: Comandos de edición básicos de sed**

Mandar	Descripción
=	Número de línea de corriente de salida.
un	Anexe texto después de la línea actual.
d	Elimine la línea actual.
Yo	Inserte texto delante de la línea actual.
p	Imprima la línea actual. De forma predeterminada, sed Imprime cada línea y edita solo las líneas que coinciden con una dirección especificada dentro del archivo. El comportamiento predeterminado se puede invalidar especificando el parámetro -n opción.
q	Salida sed sin procesar más líneas. Si el -n no se especifica la opción, genere la línea actual.

Cuadro 20-8 (continuación )

Mandar	Descripción
Q	Salida sed sin procesar más líneas.
s/ <i>regexp</i> / <i>reemplazo</i> /	Sustituya el contenido de <i>reemplazo</i> dondequiera se encuentra <i>regexp</i> . El <i>reemplazo</i> puede incluir el carácter especial &, que es equivalente al texto coincidido por <i>regexp</i> . Además, el <i>reemplazo</i> puede incluir las secuencias \1 a \9, que son el contenido de las subexpresiones correspondientes en <i>regexp</i> . Para obtener más información sobre esto, consulte la siguiente discusión sobre las referencias inversas. Después de la barra diagonal final después del <i>reemplazo</i> , se puede especificar una
	marca opcional para modificar el comportamiento del comando s.
y/ <i>Conjunto1</i> / <i>Conjunto2</i>	Realizar la transliteración mediante la conversión de caracteres de <i>Conjunto1</i> a los caracteres correspondientes en <i>Conjunto2</i> . Tenga en cuenta que, a diferencia de Tr, sed requiere que ambos conjuntos tengan la misma longitud.

El comando s es, con mucho, el comando de edición más utilizado. Demostraremos solo algunos de sus poderes realizando una edición en nuestro archivo *distros.txt*. Discutimos antes cómo el campo de fecha en *distros.txt* no estaba en un formato "amigable para computadoras". Si bien la fecha tiene el formato MM/DD/AAAA, sería mejor (para facilitar la ordenación) si el formato fuera AAAA-MM-DD. Para Realizar este cambio en el archivo a mano llevaría mucho tiempo y sería propenso a errores, pero con SED, este cambio se puede realizar en un solo paso:

```
[me@linuxbox ~]$ sed 's/([0-9]{2})/\\2/' distros.txt
SUSE      10.2    2006-12-07
Fedora    10       2008-11-25
SUSE      11.0    2008-06-19
Ubuntu    8.04   2008-04-24
Fedora    8        2007-11-08
SUSE      10.3   2007-10-04
Ubuntu    6.10   2006-10-26
Fedora    7        2007-05-31
Ubuntu    7.10   2007-10-18
Ubuntu    7.04   2007-04-19
SUSE      10.1   2006-05-11
Fedora    6        2006-10-24
Fedora    9        2008-05-13
Ubuntu    6.06   2006-06-01
Ubuntu    8.10   2008-10-30
Fedora    5        2006-03-20
```

¡Uau! Eso sí que es un comando de aspecto feo. Pero funciona. En un solo paso, hemos cambiado el formato de fecha en nuestro archivo. También es un ejemplo perfecto de por qué las expresiones regulares a veces se denominan en broma "solo escritura"

Medio. Podemos escribirlos, pero a veces no podemos leerlos. Antes de que nos sintamos tentados a huir despavoridos de este mandamiento, veamos cómo se construyó. Primero, sabemos que el comando tendrá esta estructura básica:

```
sed 's/regexp/replacement/' distros.txt
```

Nuestro siguiente paso es encontrar una expresión regular que aísle la fecha. Dado que está en formato MM/DD/AAAA y aparece al final de la línea, podemos usar una expresión como esta:

```
[0-9]{2}/[0-9]{2}/[0-9]{4}$
```

que coincide con dos dígitos, una barra diagonal, dos dígitos, una barra oblicua, cuatro dígitos y el final de la línea. Así que eso se encarga de la *expresión regular*, pero ¿qué pasa con *el reemplazo*? Para manejar eso, debemos introducir una nueva característica de expresión regular que aparece en algunas aplicaciones que usan BRE. Esta característica se denomina *referencias inversas* y funciona de la siguiente manera: si la secuencia `\n` aparece en *reemplazo* donde *n* es un número del uno al nueve, la secuencia se referirá a la subexpresión correspondiente en la expresión regular anterior. Para crear las subexpresiones, simplemente las encerramos entre paréntesis de la siguiente manera:

```
([0-9]{2})/([0-9]{2})/([0-9]{4})$
```

Ahora tenemos tres subexpresiones. El primero contiene el mes, el segundo el día del mes y el tercero el año. Ahora podemos construir *el reemplazo* de la siguiente manera:

```
\3-\1-\2
```

que nos da el año, un guión, el mes, un guión y el día.

Ahora, nuestro comando se ve así:

```
sed 's/([0-9]{2})/([0-9]{2})/([0-9]{4})$/\3-\1-\2/' distros.txt
```

Nos quedan dos problemas. La primera es que las barras adicionales en nuestra expresión regular confundirán a sed cuando intente interpretar el comando *s*. La segunda es que, dado que sed, de forma predeterminada, solo acepta expresiones regulares básicas, varios de los caracteres de nuestra expresión regular se tomarán como literales, en lugar de como metacaracteres. Podemos resolver estos dos problemas con una aplicación liberal de barras invertidas para escapar de los caracteres ofensivos:

```
sed 's/\([0-9]\{2\}\)\(\([0-9]\{2\}\)\)\(\([0-9]\{4\}\)\)$/\3-\1-\2/' distros.txt
```

¡Y ahí lo tienes!

Otra característica del comando *s* es el uso de indicadores opcionales que pueden seguir a la cadena de reemplazo. El más importante de ellos es el indicador *g*, que indica a sed que aplique la búsqueda y el reemplazo globalmente a una línea, no solo a la primera instancia, que es la predeterminada.

He aquí un ejemplo:

---

```
[me@linuxbox ~]$ echo "aaabbbccc" | sed 's/b/B/'  
aaaBbbccc
```

---

Vemos que el reemplazo se realizó pero solo en la primera instancia de la letra *b*, mientras que las instancias restantes se dejaron sin cambios. Al agregar la bandera *g*, podemos cambiar todas las instancias:

---

```
[me@linuxbox ~]$ echo "aaabbbccc" | sed 's/b/B/g'  
aaaBBBccc
```

---

Hasta ahora, hemos dado comandos individuales *sed* solo a través de la línea de comandos. También es posible construir comandos más complejos en un archivo de script utilizando la opción *-f*. Para demostrarlo, usaremos *sed* con nuestro *archivo distros.txt* para crear un informe. Nuestro informe incluirá un título en la parte superior, nuestras fechas de modificación y todos los nombres de distribución convertidos a mayúsculas. Para ello, necesitaremos escribir un script, por lo que encenderemos nuestro editor de texto e introduciremos lo siguiente:

---

```
# script sed para producir distribuciones de Linux  
  
informe 1 \\  
\  
Informe de distribuciones de Linux\\  
  
s/([0-9]{2})\|([0-9]{2})\|([0-9]{4})$/\3-\1\2/  
y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNPQRSTUVWXYZ/
```

---

Guardaremos nuestro script *sed* como *distros.sed* y lo ejecutaremos así:

---

```
[me@linuxbox ~]$ sed -f distros.sed distros.txt
```

Informe de distribuciones de Linux

SUSE	10.2	2006-12-07
FEDORA	10	2008-11-25
SUSE	11.0	2008-06-19
UBUNTU	8.04	2008-04-24
FEDORA	8	2007-11-08
SUSE	10.3	2007-10-04
UBUNTU	6.10	2006-10-26
FEDORA	7	2007-05-31
UBUNTU	7.10	2007-10-18
UBUNTU	7.04	2007-04-19
SUSE	10.1	2006-05-11
FEDORA	6	2006-10-24
FEDORA	9	2008-05-13
UBUNTU	6.06	2006-06-01
UBUNTU	8.10	2008-10-30
FEDORA	5	2006-03-20

---

Como podemos ver, nuestro script produce los resultados deseados, pero ¿cómo lo hace? Echemos otro vistazo a nuestro guión. Usaremos `cat` para numerar las líneas.

---

```
[me@linuxbox ~]$ cat -n distros.sed
1  # script sed para producir el informe de
distribuciones de Linux 2
3  1 Yo\
4  \
5  Informe de distribuciones
de Linux\ 6
7  s/^\([0-9]\{2\}\)\(\([0-9]\{2\}\)\)\(\([0-9]\{4\}\)\)$/\3-\1-\2/
8  y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNPQRSTUVWXYZ/
```

---

La línea 1 de nuestro script es un *comentario*. Al igual que en muchos archivos de configuración y lenguajes de programación en sistemas Linux, los comentarios comienzan con el carácter `#` y son seguidos por texto legible por humanos. Los comentarios se pueden colocar en cualquier parte del script (aunque no dentro de los comandos en sí) y son útiles para cualquier persona que necesite identificar y/o mantener el script.

La línea 2 es una línea en blanco. Al igual que los comentarios, se pueden agregar líneas en blanco para mejorar la legibilidad.

Muchos comandos `sed` admiten direcciones de línea. Se utilizan para especificar sobre qué líneas de la entrada se va a actuar. Las direcciones de línea pueden expresarse como números de línea simples, rangos de números de línea y el número de línea especial `$`, que indica la última línea de entrada.

Las líneas 3 a 6 contienen texto que se insertará en la dirección 1, la primera línea de la entrada. El comando `i` es seguido por la secuencia barra invertida-retorno de carro para producir un retorno de carro de escape, o lo que se llama un *carácter de continuación de línea*. Esta secuencia, que se puede utilizar en muchas circunstancias, incluidos los scripts de shell, permite incrustar un retorno de carro en

Un flujo de texto sin indicar al intérprete (en este caso `sed`) que se ha llegado al final de la línea. El comando `i` y los comandos `a` (que anexa texto) y `c` (que reemplaza texto) permiten varias líneas de texto, siempre que cada línea, excepto la última, termine con un carácter de continuación de línea. La sexta línea de nuestro script es en realidad el final de nuestro texto insertado y termina con un retorno de carro simple en lugar de un carácter de continuación de línea, lo que indica el final del comando `i`.

**Nota:** Un carácter de continuación de línea está formado por una barra invertida seguida inmediatamente por un retorno de transporte. No se permiten espacios intermedios.

La línea 7 es nuestro comando de búsqueda y reemplazo. Dado que no está precedido por una dirección, cada línea del flujo de entrada está sujeta a su acción.

La línea 8 realiza la transliteración de las letras minúsculas a letras mayúsculas. Tenga en cuenta que, a diferencia de `tr`, el comando `y` en `sed` no admite rangos de caracteres (por ejemplo, `[a-z]`), ni admite clases de caracteres POSIX. De nuevo, dado que el comando `y` no está precedido por una dirección, se aplica a todas las líneas del flujo de entrada.

## A LA GENTE A LA QUE LE GUSTA SED TAMBIÉN LE GUSTA...

SED es un programa muy capaz, capaz de realizar tareas de edición bastante complejas de flujos de texto. Se usa con mayor frecuencia para tareas simples de una línea en lugar de scripts largos. Muchos usuarios prefieren otras herramientas para tareas más grandes. Los más populares son awk y perl. Estos van más allá de meras herramientas como los programas tratados aquí y se extienden al ámbito de los lenguajes de programación completos. Perl, en particular, se utiliza a menudo en lugar de scripts de shell para muchas tareas de gestión y administración de sistemas, además de ser un medio muy popular para el desarrollo web. AWK es un poco más especializado. Su fuerza específica es su capacidad para manipular datos tabulares. Se asemeja a sed en que los programas awk normalmente procesan archivos de texto línea por línea, utilizando un esquema similar al concepto sed de una dirección seguida de

### **aspell—Corrector ortográfico interactivo**

La última herramienta que veremos es aspell, un corrector ortográfico interactivo. El programa aspell es el sucesor de un programa anterior llamado ispell, y puede ser usado, en su mayor parte, como un reemplazo directo. Si bien el programa aspell es utilizado principalmente por otros programas que requieren la capacidad de revisión ortográfica, también se puede usar de manera muy efectiva como una herramienta independiente desde la línea de comandos. Tiene la capacidad de verificar de manera inteligente varios tipos de archivos de texto, incluidos documentos HTML, programas C / C ++, mensajes de correo electrónico y otros tipos de textos especializados.

Para revisar la ortografía de un archivo de texto que contiene prosa simple, aspell podría usarse de la siguiente manera:

aArchivo de texto de revisión ortográfica

donde *textfield* es el nombre del archivo que se va a comprobar. Como ejemplo práctico, vamos a crear un archivo de texto simple llamado *foo.txt* que contiene algunos errores ortográficos deliberados:

---

```
[me@linuxbox ~]$ gato > foo.txt
El rápido zorro marrón se abalanzó sobre el perro holgazán.
```

---

A continuación, verificaremos el archivo usando aspell: [me@linuxbox ~]\$

---

```
aspell check foo.txt
```

---

Como aspell es interactivo en el modo de verificación, veremos una pantalla como esta:

---

```
El rápido zorro marrón se abalanzó sobre el perro holgazán.
```

---

1) Saltó  
2) entorchado

6) Cobarde  
7) Acampado

- |                |                     |
|----------------|---------------------|
| 3) Compensaron | 8) jorobado         |
| 4) Cojeando    | 9) Impedir          |
| 5) chulo       | 0) Umped            |
| i) Ignorar     | 1) Ignorar todo     |
| r) Reemplazar  | R) Reemplace todo   |
| a) Agregar     | l) Agregar más bajo |
| b) Abortar     | x) Salir            |

?

En la parte superior de la pantalla, vemos nuestro texto con una palabra sospechosamente deletreada resaltada. En el medio, vemos 10 sugerencias de ortografía numeradas del 0 al 9, seguidas de una lista de otras acciones posibles. Finalmente, en la parte inferior, vemos un mensaje listo para aceptar nuestra elección.

Si introducimos 1, aspell sustituye la palabra ofensiva por la palabra *saltada* y pasa a la siguiente palabra mal escrita, que es *laxy*. Si seleccionamos el reemplazo *perezoso*, aspell lo reemplaza y termina. Una vez finalizado aspell, podemos examinar nuestro fichero y ver que las faltas de ortografía han sido corregidas:

---

```
[me@linuxbox ~]$ gato foo.txt
El rápido zorro marrón saltó sobre el perro perezoso.
```

---

A menos que se indique lo contrario a través de la opción de línea de comandos --don't-backup, aspell crea un archivo de copia de seguridad que contiene el texto original añadiendo la extensión *.bak* al nombre del archivo.

Haciendo gala de nuestra destreza en la edición de sed, volveremos a poner nuestros errores de ortografía para poder reutilizar nuestro archivo:

---

```
[me@linuxbox ~]$ sed -i 's/perezoso/laxo/; s/saltó/jimped/' foo.txt
```

---

La opción sed -i le dice a sed que edite el archivo "en su lugar", lo que significa que en lugar de enviar la salida editada a la salida estándar, reescribirá el archivo con los cambios aplicados. También vemos la capacidad de colocar más de un comando de edición en la línea separándolos con un punto y coma.

A continuación, veremos cómo aspell puede manejar diferentes tipos de archivos de texto.

Usando un editor de texto como vim (los aventureros pueden querer probar sed), agregaremos algo de marcado HTML a nuestro archivo:

---

```
<html>
  <cabez
    a>      <título>Archivo HTML mal escrito</título>

    </cabez
    a>      <p>El rápido zorro marrón se abalanzó sobre el perro laxo.</p>
    <cuerp
  </html> o>

  </cuerp
  o>
```

---

Ahora, si intentamos revisar la ortografía de nuestro archivo modificado,

nos encontramos con un problema. Si lo hacemos de esta manera:

---

[me@linuxbox -] \$ **aspell check foo.txt**

---

Conseguiremos esto:

```
<html>
  <cabez
    a>      <título>Archivo HTML mal escrito</título>

    </cabez
    a>      <p>El rápido zorro marrón se abalanzó sobre el perro laxo.</p>
    <cuerp
  </html> o>

  </cuerp
  o>
```

- ```
1) HTML                      4) Hamel
2) HT ML                     5) Hamil
3) Ht-Ml                     6) Hotel
i) Ignorar                   I) Ignorar todo
r) Reemplazar                R) Reemplace todo
a) Agregar                   l) Agregar más bajo
b) Abortar                   x) Salir
```

```
?
```

aspell verá el contenido de las etiquetas HTML mal escrito. Este problema se puede superar incluyendo la opción de modo de verificación -H (HTML), de la siguiente manera:

```
[me@linuxbox ~]$ aspell -H check foo.txt
```

Nuestro resultado es el siguiente:

```
<html>
  <cabez
    a>      <título>Archivo HTML mal escrito</título>

    </cabez
    a>      <p>El rápido zorro marrón se abalanzó sobre el perro laxo.</p>
    <cuerp
  </html> o>

  </cuerp
  o>
```

- ```
1) Mi deletreado              6) Mal aplicado
2) Mi-deletreado              7) Mal llamado
3) Mal escrito                8) Reescrito
4) Disipado                   9) Error ortográfico
5) Deletreado                 0) Engañado
i) Ignorar                   I) Ignorar todo
r) Reemplazar                R) Reemplace todo
a) Agregar                   l) Agregar más bajo
b) Abortar                   x) Salir
```

```
?
```

El HTML se ignora y solo se comprueban las partes del archivo que no son de marcado. En este modo, el contenido de las etiquetas HTML se

ignora y no

Se ha comprobado la ortografía. Sin embargo, el contenido de las etiquetas ALT, que se benefician de la comprobación, se comprueba en este modo.

**Nota:** *De forma predeterminada, aspell ignorará las URL y las direcciones de correo electrónico en el texto. Este comportamiento se puede invalidar con opciones de línea de comandos. También es posible especificar qué etiquetas de marcado se comprueban y se omiten. Consulte la página del manual de aspell para obtener más detalles.*

## Nota final

En este capítulo, hemos analizado algunas de las muchas herramientas de línea de comandos que operan con texto. En el próximo capítulo, veremos varios más. Es cierto que puede que no parezca inmediatamente obvio cómo o por qué se pueden utilizar algunas de estas herramientas en el día a día, aunque hemos tratado de mostrar algunos ejemplos semiprácticos de su uso. Encontraremos en capítulos posteriores que estas herramientas forman la base de un conjunto de herramientas que se utiliza para resolver una gran cantidad de problemas prácticos. Esto será particularmente cierto cuando nos adentremos en el shell scripting, donde estas herramientas realmente mostrarán su valor.

## Crédito Extra

Hay algunos comandos de manipulación de texto más interesantes que vale la pena invertir. Entre estos se encuentran `split` (dividir archivos en partes), `csplit` (dividir archivos en partes según el contexto) y `sdiff` (combinación en paralelo de diferencias de archivos).

# 21

## **FORMATO DE LA SALIDA**

En este capítulo, continuamos analizando las herramientas relacionadas con el texto, centrándonos en los programas que se utilizan para formatear la salida de texto en lugar de cambiar el texto en sí. Estas herramientas se utilizan a menudo para preparar el texto para la impresión, un tema que trataremos en el próximo capítulo. Los programas que cubriremos en este capítulo incluyen los siguientes:

- `nl`: rectas numéricas.
- `fold`: ajuste cada línea a una longitud especificada.
- `fmt`: un formateador de texto simple.
- `pr`: formatee el texto para imprimir.
- `printf`: formatear e imprimir datos.
- `groff`: un sistema de formato de documentos.

## Herramientas de formato sencillas

Primero veremos algunas de las herramientas de formato simples. Estos son en su mayoría programas de un solo propósito, y un poco poco sofisticados en lo que hacen, pero se pueden usar para tareas pequeñas y como partes de tuberías y scripts.

### *nl: rectas numéricas*

El programa `nl` es una herramienta bastante arcana que se utiliza para realizar una tarea simple: numerar líneas. En su uso más simple, se asemeja al gato `-n`:

---

```
[me@linuxbox ~]$ nl distros.txt | cabeza
 1 SUSE      10.2  12/07/2006
 2 Fedora    10     11/25/2008
 3 SUSE      11.0  06/19/2008
 4 Ubuntu    8.04   04/24/2008
 5 Fedora    8      11/08/2007
 6 SUSE      10.3  10/04/2007
 7 Ubuntu    6.10   10/26/2006
 8 Fedora    7      05/31/2007
 9 Ubuntu    7.10   10/18/2007
10 Ubuntu    7.04   04/19/2007
```

---

Al igual que `cat`, `nl` puede aceptar varios nombres de archivo como argumentos de línea de comandos o entrada estándar. Sin embargo, `nl` tiene una serie de opciones y admite una forma primitiva de marcado para permitir tipos más complejos de numeración.

`NL` admite un concepto llamado *páginas lógicas* al numerar. Esto permite a `nl` restablecer (volver a empezar) la secuencia numérica al numerar.

Usando opciones, es posible establecer el número inicial en un valor específico y, hasta cierto punto, establecer su formato. Una página lógica se divide a su vez en un encabezado, un cuerpo y un pie de página.

Dentro de cada una de estas secciones, la numeración de líneas puede restablecerse y/o asignarse un estilo diferente. Si a `nl` se le asignan varios archivos, los trata como una sola secuencia de texto. Las secciones en el flujo de texto se indican por la presencia de algunas marcas de aspecto bastante extraño agregadas al texto, como se muestra en la Tabla 21-1.

**Tabla 21-1: Margen de beneficio `nl`**

Margen	Significado
\:\:\:\:	Inicio del encabezado de página lógica
\:\:\:	Inicio del cuerpo de la página lógica
\:	Inicio del pie de página lógico

Cada uno de los elementos de marcado de la Tabla 21-1 debe aparecer solo en su propia línea. Después de procesar un elemento de marcado, `nl` lo elimina del flujo de texto.

En la tabla 21-2 se enumeran las opciones comunes para `nl`.

Tabla 21-2: Opciones comunes de `nl`

Opción	Significado
<code>-b estilo</code>	Establezca la numeración de cuerpos en <i>estilo</i> Dónde <i>estilo</i> es uno de los siguientes: <ul style="list-style-type: none"><li>• <code>un</code> Numere todas las líneas.</li><li>• <code>t</code> Numere solo las líneas que no estén en blanco. Este es el valor predeterminado.</li><li>• <code>n</code> Ninguno.</li><li>• <code>pregexp</code> Líneas de solo números que coinciden con la expresión regular básica <i>regexp</i>.</li></ul>
<code>-f estilo</code>	Establezca la numeración del pie de página en <i>estilo</i> . El valor predeterminado es <code>n</code> (ninguno).
<code>-h estilo</code>	Establezca la numeración de encabezado en <i>estilo</i> . El valor predeterminado es <code>n</code> (ninguno).
<code>-Yo número</code>	Establezca el incremento de numeración de páginas en <i>número</i> . El valor predeterminado es 1.
<code>-n formatSet</code>	Formato de numeración dos <i>formato</i> Dónde <i>formato</i> es uno de los siguientes: <ul style="list-style-type: none"><li>• <code>En</code> Justificado a la izquierda, sin ceros a la izquierda.</li><li>• <code>Rn</code> Justificado a la derecha, sin ceros a la izquierda. Este es el valor predeterminado.</li><li>• <code>RZ</code> Justificado a la derecha, con ceros a la izquierda.</li></ul>
<code>-p</code>	No restablezca la numeración de páginas al principio de cada página lógica.
<code>-s cuerda</code>	Agregar <i>cuerda</i> al final de cada número de línea para crear un separador. El valor predeterminado es un solo carácter de tabulación.
<code>-v número</code>	Establezca el número de primera línea de cada página lógica en <i>número</i> . El valor predeterminado es 1.
<code>-w Ancho</code>	Establezca el ancho del campo de número de línea en <i>Ancho</i> . El valor predeterminado es 6.

Es cierto que probablemente no numeraremos líneas con tanta frecuencia, pero podemos usar `nl` para ver cómo podemos combinar múltiples herramientas para realizar tareas más complejas. Nos basaremos en el trabajo realizado en el capítulo anterior para elaborar un informe sobre las distribuciones de Linux. Dado que usaremos `nl`, será útil incluir su marcado de encabezado / cuerpo / pie de página. Para ello, lo añadiremos al script `sed` del último capítulo. Usando nuestro editor de texto, cambiaremos el script de la siguiente manera y lo guardaremos como `distros-nl.sed`:

---

```
# script sed para producir un informe de distribuciones de Linux
```

```
1 i\
\\:\\:\\:\\\\
\
Informe de distribuciones de Linux\
\
Nombre      Ver.      Liberado\
----- \
\\:\\:\\
s/([0-9]{2\})\\/([0-9]{2\})\\/([0-9]{4\})$/\\3-\\1-\\2/
```

```
$ a\  
\\:\\  
\  
Fin del informe
```

---

El script ahora inserta el marcado de página lógica nl y agrega un pie de página al final del informe. Tenga en cuenta que tuvimos que duplicar las barras invertidas en nuestro marcado, porque sed normalmente las interpreta como caracteres de escape.

A continuación, produciremos nuestro informe mejorado combinando sort, sed,

```
y nl: [me@linuxbox ~]$ sort -k 1,1 -k 2n distros.txt | sed -f distros-nl.sed | nl
```

### Informe de distribuciones de Linux

Nombre	Ver.	Liberado
1 Fedora	5	2006-03-20
2 Fedora	6	2006-10-24
3 Fedora	7	2007-05-31
4 Fedora	8	2007-11-08
5 Fedora	9	2008-05-13
6 Fedora	10	2008-11-25
7 SUSE	10.1	2006-05-11
8 SUSE	10.2	2006-12-07
9 SUSE	10.3	2007-10-04
10 SUSE	11.0	2008-06-19
11 Ubuntu	6.06	2006-06-01
12 Ubuntu	6.10	2006-10-26
13 Ubuntu	7.04	2007-04-19
14 Ubuntu	7.10	2007-10-18
15 Ubuntu	8.04	2008-04-24
16 Ubuntu	8.10	2008-10-30

---

### Fin del informe

Nuestro informe es el resultado de nuestra canalización de comandos. Primero, ordenamos la lista por nombre de distribución y versión (campos 1 y 2), y luego procesamos los resultados con sed, agregando el encabezado del informe (incluido el marcado de página lógica para nl) y el pie de página. Finalmente, procesamos el resultado con nl, que, de forma predeterminada, numera solo las líneas del flujo de texto que pertenecen a la sección del cuerpo de la página lógica.

Podemos repetir el comando y experimentar con diferentes opciones para nl. Algunos interesantes son:

```
nl -n rz
```

y

```
nl -w 3 -s " "
```

## *fold: ajustar cada línea a una longitud especificada*

*El plegado* es el proceso de dividir líneas de texto a un ancho específico. Al igual que nuestros otros comandos, fold acepta uno o más archivos de texto o entrada estándar. Si enviamos fold a un simple flujo de texto, podemos ver cómo funciona:

```
[me@linuxbox -]$ echo "El rápido zorro marrón saltó sobre el perro perezoso." | fold  
-w 12  
El rápido  
zorro saltó  
sobre el  
perro  
perezoso.
```

Aquí vemos el pliegue en acción. El texto enviado por el comando echo se divide en segmentos especificados por la opción -w. En este ejemplo, especificamos un ancho de línea de 12 caracteres. Si no se especifica ningún ancho, el valor predeterminado es 80 caracteres. Observe que las líneas se rompen independientemente de los límites de las palabras. La adición de la opción -s hará que fold rompa la línea en el último espacio disponible antes de que se alcance el ancho de línea:

```
[me@linuxbox -]$ echo "El rápido zorro marrón saltó sobre el perro perezoso." | fold  
-w 12 -s El  
rápido zorro  
marrón  
saltó sobre  
el perro  
perezoso.
```

## *fmt: un formateador de texto simple*

El programa fmt también dobla texto, y mucho más. Acepta archivos o entradas estándar y realiza el formato de párrafo en el flujo de texto. Básicamente, rellena y une líneas en el texto mientras conserva las líneas en blanco y la sangría.

Para demostrarlo, necesitaremos algo de texto. Vamos a levantar algunos de la fmt  
Página de información:

'fmt' lee de los argumentos FILE especificados (o la entrada estándar si no se proporciona ninguno) y escribe en la salida estándar.

De forma predeterminada, las líneas en blanco, los espacios entre palabras y la sangría se conservan en la salida; las líneas de entrada sucesivas con sangría diferente no se unen; las tabulaciones se expanden en la entrada y se introducen en la salida.

'fmt' prefiere romper líneas al final de una oración, y trata de evitar saltos de línea después de la primera palabra de una oración o antes de la última palabra de una oración. Un "salto de oración" se define como el final de un párrafo o una palabra que termina en cualquiera de los '.?!', seguido de dos espacios o al final de la línea, ignorando cualquier paréntesis o comillas intermedias. Al igual que TeX, 'fmt' lee "párrafos" enteros antes de elegir saltos de línea; el algoritmo es una variante del dado por Donald E. Knuth y Michael F.

Plass en "Breaking Paragraphs Into Lines", 'Software--Practice & Experience' 11, 11 (noviembre de 1981), 1119-1184.

Copiaremos este texto en nuestro editor de texto y guardaremos el archivo como *fmt-info.txt*. Ahora, digamos que queremos reformatar este texto para que quepa en una columna de 50 caracteres. Podríamos hacer esto procesando el archivo con *fmt* y la opción *-w*:

---

```
[me@linuxbox -]$ fmt -w 50 fmt-info.txt | cabeza
'fmt' lee de los argumentos FILE especificados (o
la entrada estándar si
no se proporciona ninguno) y escribe en la salida estándar.
```

De forma predeterminada, las líneas en blanco, los espacios entre palabras y la sangría son

Se conservan en la salida; las líneas de entrada sucesivas con diferente sangría no se unen; las pestañas se expanden en la entrada y se

---

introducen en la salida.

Bueno, ese es un resultado incómodo. Tal vez deberíamos leer este texto, ya que explica lo que está pasando:

De forma predeterminada, las líneas en blanco, los espacios entre palabras y la sangría se conservan en la salida; Las líneas de entrada sucesivas con diferente sangría no se unen; Las pestañas se expanden en la entrada y se introducen en la salida.

Por lo tanto, *fmt* conserva la sangría de la primera línea. Afortunadamente, *fmt* proporciona una opción para corregir esto:

---

```
[me@linuxbox -]$ fmt -cw 50 fmt-info.txt
'fmt' lee de los argumentos FILE especificados
(o la entrada estándar si no se proporciona
ninguno) y escribe en la salida estándar.
```

De forma predeterminada, las líneas en blanco, los espacios entre palabras y la sangría se conservan en la salida; las líneas de entrada sucesivas con sangría diferente no se unen; las tabulaciones se expanden en la entrada y se introducen en la salida.

'fmt' prefiere las líneas separadas al final de una oración, y trata de evitar los saltos de línea después de la primera palabra de una oración o antes de la última palabra de una frase. Un "salto de oración" se define como el final de un párrafo o una palabra que termina en cualquiera de los dos puntos, seguido de ':?!'

por dos espacios o al final de la línea, ignorando cualquier paréntesis o comilla intermedia. Al igual que TeX, 'fmt' lee "párrafos" enteros antes de elegir saltos de línea; el algoritmo es una variante de

la dada por Donald E. Knuth y Michael F. plass en  
"Dividir párrafos en líneas",  
'Software: Práctica y Experiencia' 11, 11  
(noviembre de 1981), 1119-1184.

---

Mucho mejor. Al agregar la opción -c, ahora tenemos el resultado deseado.



fmt tiene algunas opciones interesantes, como se muestra en la Tabla 21-3.

**Tabla 21-3: Opciones de FMT**

Opción	Descripción
-c	Operar en <i>Margen de la corona</i> modo. Esto conserva la sangría de las dos primeras líneas de un párrafo. Las líneas siguientes se alinean con la sangría de la segunda línea.
-p <i>stringFormat</i>	<i>solo las líneas que comienzan con la cadena de prefijo</i> . Después de formatear, el contenido de <i>cuerda</i> tienen un prefijo en cada línea reformateada. Esta opción se puede utilizar para dar formato al texto en los comentarios del código fuente. Por ejemplo, cualquier lenguaje de programación o fichero de configuración que utilice un # para delinejar un comentario se podría formatear especificando -p '# ' de modo que solo se formateará los comentarios. Vea el ejemplo a continuación.
-s	Modo de solo división. En este modo, las líneas se dividirán solo para que se ajusten al ancho de columna especificado. Las líneas cortas no se unirán para llenar líneas. Este modo es útil cuando se da formato a texto, como código, donde no se desea unir.
-u	Realice un espaciado uniforme. Esto aplicará el formato tradicional de "estilo máquina de escribir" al texto. Esto significa un solo espacio entre palabras y dos espacios entre oraciones. Este modo es útil para eliminar <i>justificación</i> , es decir, alineación forzada a los márgenes izquierdo y derecho.
-w <i>width</i>	<i>Formatear el texto para que quepa dentro del ancho de una columna</i> caracteres de ancho. El valor predeterminado es de 75 caracteres. Nota: Fmt De hecho, da formato a las líneas ligeramente más cortas que el ancho especificado para permitir el equilibrio de líneas.

La opción -p es particularmente interesante. Con él, podemos formatear partes seleccionadas de un archivo, siempre que las líneas a formatear comiencen todas con la misma secuencia de caracteres. Muchos lenguajes de programación utilizan la almohadilla (#) para indicar el comienzo de un comentario y, por lo tanto, se pueden formatear con esta opción. Vamos a crear un archivo que simule un programa que usa comentarios:

```
[me@linuxbox ~]$ gato > fmt-code.txt  
# Este archivo contiene código con comentarios.
```

```
# Esta línea es un comentario.  
# Seguido de otra línea de
```

comentario.# Y otro.

Esto, por otro lado, es una línea de código.  
Y otra línea de código.  
Y otro.

Nuestro archivo de muestra contiene comentarios, que comienzan con la cadena # (un # seguido de un espacio) y líneas de "código", que no lo hacen. Ahora, usando fmt, podemos formatear los comentarios y dejar el código intacto:

---

```
[me@linuxbox ~]$ fmt -w 50 -p '# ' fmt-code.txt  
# Este archivo contiene código con comentarios.
```

```
# Esta línea es un comentario. Seguido de otra  
línea de # comentario . Y otro.
```

```
Este, por otro lado, es una línea de código.  
Y otra línea de código.  
Y otro.
```

---

Observe que las líneas de comentario adyacentes se unen, mientras que las líneas en blanco y las líneas que no comienzan con el prefijo especificado se conservan.

### **pr: Formato de texto para impresión**

El programa pr se utiliza para *paginar* texto. Al imprimir texto, a menudo es conveniente separar las páginas de salida con varias líneas de espacio en blanco para proporcionar un margen superior e inferior para cada página. Además, este espacio en blanco se puede utilizar para insertar un encabezado y un pie de página en cada página.

Demostraremos pr formateando nuestro archivo *de distros.txt* en una serie de páginas muy cortas (solo se muestran las dos primeras páginas):

---

```
[me@linuxbox ~]$ pr -l 15 -w 65 distros.txt
```

2012-12-11 18:27

distros.txt

Página 1

SUSE	10.2	12/07/2006
Fedora	10	11/25/2008
SUSE	11.0	06/19/2008
Ubuntu	8.04	04/24/2008
Fedora	8	11/08/2007

2012-12-11 18:27

distros.txt

Página 2

SUSE	10.3	10/04/2007
Ubuntu	6.10	10/26/2006

Fedora	7	05/31/2007
Ubuntu	7.10	10/18/2007
Ubuntu	7.04	04/19/2007

En este ejemplo, empleamos la opción `-l` (para la longitud de la página) y la opción `-w` (ancho de la página) para definir una "página" de 65 caracteres de ancho y 15 líneas de largo. PR pagina el contenido del archivo `distros.txt`, separa cada página con varias líneas de espacio en blanco y crea un encabezado predeterminado que contiene la hora de modificación del archivo, el nombre de archivo y el número de página. El programa pr proporciona muchas opciones para controlar el diseño de la página. Echaremos un vistazo a más de ellos en el Capítulo 22.

### **printf: formatear e imprimir datos**

A diferencia de los otros comandos de este capítulo, el comando `printf` no se usa para tuberías (no acepta entradas estándar), ni encuentra aplicaciones frecuentes directamente en la línea de comandos (se usa principalmente en scripts). Entonces, ¿por qué es importante? Porque es muy utilizado.

`printf` (de la frase *print formatted*) se desarrolló originalmente para el lenguaje de programación C y se ha implementado en muchos lenguajes de programación, incluido el shell. De hecho, en bash, `printf` es un incorporado.

`printf` funciona de la siguiente manera:

#### **Argumentos `printf` "format"**

Al comando se le asigna una cadena que contiene una descripción de formato, que luego se aplica a una lista de argumentos. El resultado formateado se envía a la salida estándar. He aquí un ejemplo trivial:

---

```
[me@linuxbox ~]$ printf "Formateé la cadena: %s\n" foo
Formateé la cadena: foo
```

---

La cadena de formato puede contener texto literal (como `yo formateé la cadena:`); secuencias de escape (como `\n`, un carácter de nueva línea); y secuencias que comienzan con el carácter `%`, que se denominan *especificaciones de conversión*. En el ejemplo anterior, la especificación de conversión `%s` se utiliza para formatear la cadena `foo` y colocarla en la salida del comando. Aquí está de nuevo:

---

```
[me@linuxbox ~]$ printf "Formateé '%s' como una cadena.\n" foo
Formateé 'foo' como una cadena.
```

---

Como podemos ver, la especificación de conversión `%s` se reemplaza por la cadena `foo` en la salida del comando. La conversión `s` se utiliza para dar formato a los datos de cadena. Hay otros especificadores para otros tipos de datos. En la Tabla 21-4 se enumeran los tipos de datos más utilizados.

**Tabla 21-4: Especificadores de tipo de datos `printf` comunes**

Especificador	Descripción
<code>d</code>	Dar formato a un número como un entero decimal con signo.
<code>f</code>	Formatee y genere un número de coma flotante.

*(continuación)*

Cuadro 21-4 (*continuación* )

Especificador	Descripción
o	Formatee un número entero como un número octal.
s	Dar formato a una cadena.
x	Dar formato a un número entero como un número hexadecimal usando minúsculas <i>un-f</i> donde sea necesario.
X	Igual que x, pero usa letras mayúsculas.
%	Imprima un símbolo % literal (es decir, especifique "%%").

Demostraremos el efecto de cada uno de los especificadores de conversión en la cadena 380 :

```
[me@linuxbox ~]$ printf "%d, %f, %o, %s, %x, %X\n" 380 380 380 380 380 380  
380, 380.000000, 574, 380, 17C, 17C
```

Dado que especificamos seis especificadores de conversión, también debemos proporcionar seis argumentos para que printf los procese. Los seis resultados muestran el efecto de cada especificador.

Se pueden agregar varios componentes opcionales al especificador de conversión para ajustar su salida. Una especificación de conversión completa puede constar de lo siguiente:

*%[banderas][ancho][.precisión]conversion\_specification*

Múltiples componentes opcionales, cuando se utilizan, deben aparecer en el orden especificado anteriormente para ser interpretados correctamente. En la Tabla 21-5 se describe cada componente.

Tabla 21-5: Componentes de especificación de conversión printf

Componente	Descripción
<i>Banderas</i>	Hay cinco banderas diferentes: <ul style="list-style-type: none"><li>• # Utilice el <i>formato alternativo</i> para la salida. Esto varía según el tipo de datos. Para la conversión o (número octal), la salida tiene el prefijo 0 (cero). En el caso de las conversiones x y X (número hexadecimal), la salida tiene el prefijo 0x o 0X, respectivamente.</li><li>• 0 (cero) Rellena la salida con ceros. Esto significa que el campo se llenará con ceros a la izquierda, como en 000380.</li><li>• - (guión) Alinee a la izquierda la salida. De forma predeterminada, printf alinea a la derecha la salida.</li></ul>

- (espacio) Produce un espacio inicial para números positivos.
- + (signo más) Signo positivo de números. De forma predeterminada, printf solo firma números negativos.

Cuadro 21-5 (continuación )

Componente	Descripción
Ancho	Un número que especifica el ancho de campo mínimo
.precisión	En el caso de los números de coma flotante, especifique el número de dígitos de precisión que se generarán después del separador decimal. Para la conversión de cadenas, <i>precisión</i> Especifica el número de caracteres que se van a generar.

En la Tabla 21-6 se enumeran algunos ejemplos de diferentes formatos en acción.

Tabla 21-6: Ejemplos de especificaciones de conversión de impresión

Argumento	Formato	Resultado	Notas
380	"%d"	380	Formato simple de un número entero
380	"%#x"	0x17c	Entero formateado como hexanúmero decimal usando el Indicador de formato alternativo
380	"%05d"	00380	Entero formateado con interlineado ceros (relleno) y un mínimo ancho de campo de cinco caracteres
380	"%05.5f"	380.00000	Número formateado como un número de punto con relleno y 5 decimales de precisión. Dado que el mínimo especificado ancho de campo (5) es menor que el Ancho real del formato número, el relleno no tiene efecto.
380	"%010.5f"	0380.00000	Aumento del campo mínimo ancho a 10 hace que el acolchado visible.
380	"%+d"	+380	La bandera + firma un signo positivo número.
380	"%-d"	380	La marca - alinea a la izquierda el formateo.
abcdefghijklm	"%5s"	abcdefghijklm	Una cadena se formatea con un Ancho de campo mínimo.
abcdefghijklm	"%.5s"	ABC	Al aplicar precisión a un

string, se trunca.

---



Una vez más, printf se usa principalmente en scripts, donde se emplea para formatear datos tabulares, en lugar de en la línea de comandos directamente. Pero aún podemos mostrar cómo se puede usar para resolver varios problemas de formato. Primero, generemos algunos campos separados por caracteres de tabulación:

---

```
[me@linuxbox ~]$ printf "%s\t%s\t%s\n" str1 str2 str3
str1    str2    str3
```

---

Al insertar \t (la secuencia de escape para una pestaña), logramos el efecto deseado. A continuación, algunos números con un formato ordenado:

---

```
[me@linuxbox ~]$ printf "Línea: %05d %15.3f Resultado: %+15d\n" 1071
3.14156295 32589
Línea: 01071      3.142 Resultado: +32589
```

---

Esto muestra el efecto del ancho de campo mínimo en el espaciado de los campos. ¿O qué tal formatear una pequeña página web?

---

```
[me@linuxbox ~]$ printf "<html>\n\t<head>\n\t\t<title>%s</title>\n\t</head>
\n\t<cuerpo>\n\t\t<p>%s</p>\n\t</cuerpo>\n</html>\n" "Título de la página" "Contenido
de la página"
<html>
  <cabez
    a>      <título>Título de la página </título>

  </cabez
    a>      <p>Contenido de la página</p>
    <cuerpo
  </html> >

  </cuerp
  o>
```

---

## Sistemas de formateo de documentos

Hasta ahora, hemos examinado las sencillas herramientas de formato de texto. Estos son buenos para tareas pequeñas y simples, pero ¿qué pasa con los trabajos más grandes? Una de las razones por las que Unix se convirtió en un sistema operativo popular entre los usuarios técnicos y científicos (además de proporcionar un potente entorno multitarea y multiusuario para todo tipo de desarrollo de software) es que ofrecía herramientas que podían utilizarse para producir muchos tipos de documentos, en particular publicaciones científicas y académicas. De hecho, como se describe en la documentación de GNU, la preparación de documentos fue fundamental para el desarrollo de Unix:

La primera versión de UNIX se desarrolló en un PDP-7 que se encontraba en los Laboratorios Bell. En 1971, los desarrolladores querían obtener un PDP-11 para seguir trabajando en el sistema operativo. Con el fin de justificar el costo de este sistema, propusieron que implementarían un sistema de formateo de documentos para la división de patentes de AT&T. Este primer

programa de formateo fue una reimplementación del roff de McIlroy,  
escrito por J.F. Ossanna.

## *La familia roff y el TEX*

Dos familias principales de formateadores de documentos dominan el campo: los descendientes del programa roff original, incluidos nroff y troff, y los basados en el sistema de composición tipográfica TEX (pronunciado "tek") de Donald Knuth. Y sí, la "E" caída en el medio es parte de su nombre.

El nombre *roff* se deriva del término "Voy a correr una copia para tí". El programa nroff se utiliza para formatear documentos para enviarlos a dispositivos que utilizan fuentes monoespaciadas, como terminales de caracteres y

Impresoras tipo máquina de escribir. En el momento de su introducción, esto incluía casi todos los dispositivos de impresión conectados a las computadoras. El programa troff posterior formatea los documentos para su salida en *máquinas tipográficas*, dispositivos utilizados para producir tipos "listos para la cámara" para la impresión comercial. Hoy en día, la mayoría de las imprentas informáticas son capaces de simplificar la producción de las máquinas de composición tipográfica. La familia roff también incluye algunos otros programas que se utilizan para preparar partes de documentos. Estos incluyen eqn (para ecuaciones matemáticas) y tbl (para tablas).

El sistema TEX (en forma estable) apareció por primera vez en 1989 y, hasta cierto punto, ha desplazado al troff como la herramienta de elección para la producción de tipografías. No cubriremos TEX aquí, debido tanto a su complejidad (hay libros enteros sobre él) como al hecho de que no está instalado de forma predeterminada en la mayoría de los sistemas Linux modernos.

**Nota:** Para aquellos interesados en instalar TEX, echa un vistazo al paquete *texlive*, que se puede encontrar en la mayoría de los repositorios de distribución, y al editor de contenido gráfico *LyX*.

### *groff: un sistema de formateo de documentos*

groff es un conjunto de programas que contiene la implementación GNU de troff. También incluye un script que se utiliza para emular nroff y el resto de la familia roff.

Si bien roff y sus descendientes se utilizan para crear documentos formateados, lo hacen de una manera que es bastante extraña para los usuarios modernos. La mayoría de los documentos de hoy en día se producen utilizando procesadores de texto que son capaces de realizar tanto la composición como el diseño de un documento en un solo paso. Antes de la llegada del procesador de textos gráfico, los documentos a menudo se producían en un proceso de dos pasos que involucraba el uso de un editor de texto para realizar la composición y un procesador, como troff, para aplicar el formato. Las instrucciones para el programa de formateo se incrustaron en el texto compuesto mediante el uso de un lenguaje de marcado. El análogo moderno de este tipo de proceso es la página web, que se compone utilizando un editor de texto de algún tipo y luego se representa mediante un navegador web que utiliza HTML como lenguaje de marcado para describir el diseño final de la página.

No vamos a cubrir groff en su totalidad, ya que muchos elementos de su lenguaje de marcado se ocupan de detalles bastante arcanos de la tipografía. En su lugar, nos concentraremos en uno de sus *paquetes de macros* que sigue siendo ampliamente utilizado. Estos paquetes de macros condensan muchos de sus comandos de bajo nivel en un conjunto más pequeño de comandos de alto nivel que facilitan mucho el uso de groff .

Por un momento, consideremos la humilde página del manual. Vive en el */usr/share/man* como un archivo de texto comprimido con gzip. Si examináramos su contenido sin comprimir, veríamos lo siguiente (se muestra la página de manual para ls en la sección 1):

---

```
[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | cabeza
.\"
." NO MODIFIQUE ESTE ARCHIVO! Fue generado por help2man 1.35.
.TH LS "1" "Abril de 2008" "GNU coreutils 6.10" "Comandos de usuario"
.SH NOMBRE
.ls \- lista de contenidos del directorio
.SH SINOPSIS
.B ls
[OPTION\fR]... [\fFILO\fR]...
.SH DESCRIPCIÓN
.\" Agregue cualquier descripción adicional aquí
.PP
```

---

En comparación con la página de manual en su presentación normal, podemos comenzar a ver una correlación entre el lenguaje de marcado y sus resultados:

---

```
[me@linuxbox ~]$ hombre ls | cabeza
LS(1) Comandos de usuario LS(1)
```

NOMBRE  
ls - Listar el contenido del directorio  
SINOPSIS  
ls [OPCIÓN]... [ARCHIVO]...

---

Esto es de interés porque las páginas de manual son representadas por groff, utilizando el paquete de macros mandoc. De hecho, podemos simular el comando man con esta tubería.

---

```
[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | groff -mandoc -T ascii | cabeza
LS(1) Comandos de usuario LS(1)
```

NOMBRE  
ls - Listar el contenido del directorio  
SINOPSIS  
ls [OPCIÓN]... [ARCHIVO]...

---

Aquí usamos el programa groff con las opciones configuradas para especificar el paquete de macros mandoc y el controlador de salida para ASCII. Groff puede producir resultados en varios formatos. Si no se especifica ningún formato, PostScript se genera de forma predeterminada:

---

```
[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | groff -mandoc | cabeza
%! PS-Adobe-3.0
%%Creador: groff versión 1.18.1
%%CreationDate: Thu Feb 2 13:44:37 2012
%%DocumentNeededResources: fuente Times-Roman
```

```
%%+ fuente Times-Bold  
%%+ fuente Times-Italic  
%%DocumentSuppliedResources: procset grops 1.18 1  
%%Páginas: 4  
%%PageOrder: Ascender  
%%Orientación: Vertical
```

---

PostScript es un lenguaje de descripción de página que se utiliza para describir el contenido de una página impresa en un dispositivo similar a la composición tipográfica. Podemos tomar la salida de nuestro comando y almacenarla en un archivo (asumiendo que estamos usando un escritorio gráfico con un *directorio de escritorio*):

---

```
[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | groff -mandoc > ~/Escritorio/fo.ps
```

---

Debería aparecer un ícono para el archivo de salida en el escritorio. Al hacer doble clic en el ícono, un visor de páginas debería iniciarse y revelar el archivo en su forma renderizada (Figura 21-1).

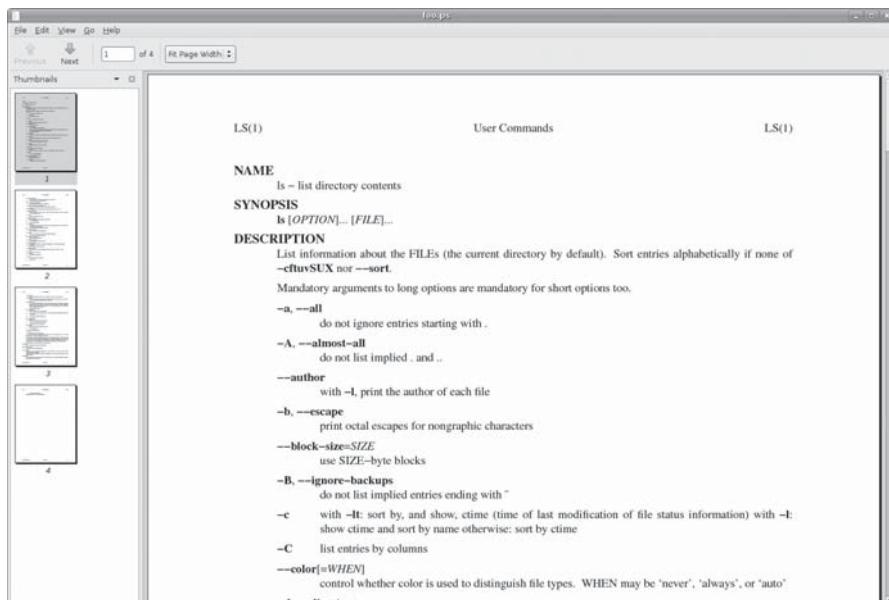


Figura 21-1: Visualización de la salida PostScript con un visor de páginas en GNOME

Lo que vemos es una página de manual muy bien compuesta para ls! De hecho, es posible convertir el archivo PostScript en un *archivo PDF (Portable Document Format)* con este comando:

---

```
[me@linuxbox ~]$ ps2pdf ~/Escritorio/fo.ps ~/Escritorio/ls.pdf
```

---

El programa ps2pdf es parte del paquete ghostscript, que se instala en la mayoría de los sistemas Linux que admiten impresión.

**Nota:** Los sistemas Linux a menudo incluyen muchos programas de línea de comandos para la conversión de formatos de archivo. A menudo se nombran utilizando la convención `format2format`. Intente usar el comando `ls /usr/bin/*[:alpha:]2[:alpha:]*` para identificarlos. Intente también buscar programas llamados `formattoformat`.

Para nuestro último ejercicio con groff, volveremos a visitar a nuestro viejo amigo `distros.txt`. Esta vez, usaremos el programa `tbl`, que se usa para formatear tablas, para componer nuestra lista de distribuciones de Linux. Para hacer esto, vamos a usar nuestro script `sed` anterior para agregar marcado a un flujo de texto que alimentaremos a `groff`.

Primero, necesitamos modificar nuestro script `sed` para agregar las solicitudes necesarias que `TBL` requiere. Usando un editor de texto, cambiaremos `distros.sed` a lo siguiente:

---

```
# script sed para producir un informe de distribuciones de Linux
```

```
1 i\
.TS\
caja
central;\ cb
s s\
cb cb cb\
l n c.\Informe de distribuciones de Linux\
=\Nombre Versión publicada\
s/([0-9]\{2\})/([0-9]\{2\})/([0-9]\{4\})$/3-1-2/
$ a\
.TE
```

---

Tenga en cuenta que para que el script funcione correctamente, se debe tener cuidado de que las palabras *Nombre Versión publicada* estén separadas por tabulaciones, no por espacios. Guardaremos el archivo resultante como `distros-tbl.sed`. `tbl` utiliza el archivo `.TS` y `.TE` solicitan iniciar y finalizar la tabla. Las filas que siguen a `.TS` definen las propiedades globales de la tabla, que, por ejemplo, están centradas horizontalmente en la página y rodeadas por un cuadro. Las líneas restantes de la definición describen el diseño de cada fila de la tabla. Ahora, si volvemos a ejecutar nuestra canalización de generación de informes con el nuevo script `sed`, obtendremos lo siguiente:

```
[me@linuxbox ~]$ sort -k 1,1 -k 2n distros.txt | sed -f distros-tbl.sed | groff
-t -T ascii 2>/dev/null
+-----+
| Informe de distribuciones de Linux |
+-----+
| Nombre Versión Lanzado |
+-----+
|Fedora      5      2006-03-20 |
|Fedora      6      2006-10-24 |
|Fedora      7      2007-05-31 |
|Fedora      8      2007-11-08 |
|Fedora      9      2008-05-13 |
|Fedora     10      2008-11-25 |
|SUSE       10.1    2006-05-11 |
```

SUSE	10.2	2006-12-07	
SUSE	10.3	2007-10-04	
SUSE	11.0	2008-06-19	
Ubuntu	6.06	2006-06-01	

Ubuntu	6.10	2006-10-26
Ubuntu	7.04	2007-04-19
Ubuntu	7.10	18/10/2007
Ubuntu	8.04	2008-04-24
Ubuntu	8.10	2008-10-30

Al agregar la opción `-t` a `groff`, se le indica que preprocese el flujo de texto con `tbl`. Del mismo modo, la opción `-T` se utiliza para emitir a ASCII en lugar de al medio de salida predeterminado, PostScript.

El formato de la salida es el mejor que podemos esperar si estamos limitados a las capacidades de una pantalla de terminal o una impresora estilo máquina de escribir. Si especificamos la salida PostScript y vemos gráficamente la salida resultante, obtenemos un resultado mucho más

satisfactorio (ver Figura 21-2).

```
[me@linuxbox ~]$ sort -k 1,1 -k 2n distros.txt | sed -f distros-tbl.sed | groff -t > ~/Escritorio/foo.ps
```

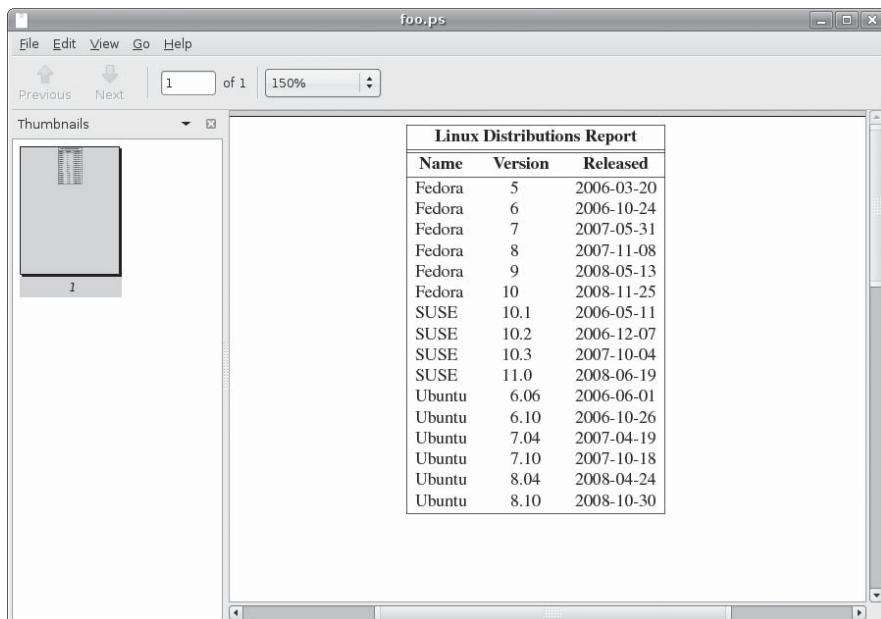


Figura 21-2: Visualización de la tabla terminada

## Nota final

Dado que el texto es tan central para el carácter de los sistemas operativos tipo Unix, tiene sentido que haya muchas herramientas que se utilicen para manipular y formatear el texto. Como hemos visto, ¡los hay! Las herramientas de formato simples como `fmt` y `pr` encontrarán muchos usos en scripts que producen documentos cortos, mientras que `groff` (y amigos) se pueden usar para escribir libros. Es posible que nunca escribamos un documento técnico utilizando herramientas de línea de comandos (¡aunque mucha gente lo hace!), pero es bueno saber que podríamos.



# 22

## IMPRENTA

Después de pasar los últimos dos capítulos manipulando el texto, es hora de poner ese texto en papel. En este capítulo, veremos las herramientas de línea de comandos que se utilizan para imprimir archivos y controlar el funcionamiento de la impresora. No lo seremos

En cuanto a la configuración de la impresión, ya que varía de una distribución a otra y suele configurarse automáticamente durante la instalación. Tenga en cuenta que necesitaremos una configuración de impresora que funcione para realizar los ejercicios de este capítulo.

Discutiremos los siguientes comandos:

- **pr:** convierte archivos de texto para imprimir.
- **lpr:** imprime archivos.
- **lp:** Imprimir archivos (Sistema V).
- **a2ps:** formatee archivos para imprimir en una impresora PostScript.
- **lpstat:** muestra la información de estado de la impresora.
- **lpq:** muestra el estado de la cola de la impresora.
- **lprm:** cancela los trabajos de impresión.
- **cancel:** cancela trabajos de impresión (Sistema V).



## Breve historia de la impresora

Para comprender completamente las características de impresión que se encuentran en los sistemas operativos tipo Unix, primero debemos aprender algo de historia. La impresión en sistemas tipo Unix se remonta a los inicios del propio sistema operativo. En aquellos días, las impresoras y la forma en que se usaban eran muy diferentes de cómo son hoy.

### *Imprimir en tiempos oscuros*

Al igual que las propias computadoras, las impresoras en la era anterior a la PC tendían a ser grandes, caras y centralizadas. El usuario típico de computadoras de 1980 trabajaba en Un terminal conectado a una computadora a cierta distancia. La impresora estaba situada cerca de la computadora y estaba bajo la atenta mirada de los operadores del ordenador.

Cuando las impresoras eran caras y centralizadas, como a menudo lo eran en los primeros días de Unix, era una práctica común que muchos usuarios compartieran una impresora. Para identificar los trabajos de impresión que pertenecían a un usuario en particular, a menudo se imprimía una página de banner que mostraba el nombre del usuario al comienzo de cada trabajo de impresión.

A continuación, el personal de soporte informático cargaba un carro que contenía los trabajos de impresión del día y los entregaba a los usuarios individuales.

### *Impresoras basadas en caracteres*

La tecnología de impresión de los años 80 era muy diferente en dos aspectos. En primer lugar, las imprentas de esa época eran casi siempre imprentas de impacto. *Las impresoras de impacto* utilizan un mecanismo mecánico que golpea una cinta contra el papel para formar impresiones de caracteres en la página. Dos de las tecnologías populares de esa época fueron la impresión en rueda de margarita y la impresión matricial.

La segunda característica, y más importante, de las primeras impresoras era que utilizaban un conjunto fijo de caracteres que eran intrínsecos al propio dispositivo. Por ejemplo, una impresora de rueda de margarita podría imprimir solo los caracteres realmente moldeados en los pétalos de la rueda de margarita. Esto hizo que las impresoras se parecieran mucho a las máquinas de escribir de alta velocidad. Al igual que con la mayoría de las máquinas de escribir, imprimían usando fuentes monoespaciadas (de ancho fijo). Esto significa que cada carácter tiene el mismo ancho. La impresión se realizaba en posiciones fijas de la página, y el área imprimible de una página contenía un número fijo de caracteres. La mayoría de las impresoras imprimían 10 caracteres por pulgada (CPI) horizontalmente y 6 líneas por pulgada (LPI) verticalmente. Con este esquema, una hoja de papel de carta estadounidense tiene 85 caracteres de ancho y 66 líneas de alto. Teniendo en cuenta un pequeño margen en cada lado, se consideró que 80 caracteres era el ancho máximo de una línea de impresión. Esto explica por qué las pantallas de terminal (y nuestros

emuladores de terminal) suelen tener 80 caracteres de ancho. Proporciona una *vista WYSIWYG* (*Lo que ves es lo que obtienes*) de la salida impresa, utilizando una fuente monoespaciada.

Los datos se envían a una impresora similar a una máquina de escribir en un simple flujo de bytes con los caracteres que se van a imprimir. Por ejemplo, para imprimir una *a*, se envía el código de carácter ASCII 97. Además, los códigos de control ASCII de número bajo proporcionaban un medio para mover el carro y el papel de la impresora, mediante códigos

para retorno de carro, avance de línea, avance de forma, etc. Con los códigos de control, es posible lograr algunos efectos de fuente limitados, como la negrita, haciendo que la impresora imprima un carácter, la barra espaciadora y vuelva a imprimir el carácter para obtener una impresión de impresión más oscura en la página. De hecho, podemos presenciar esto si usamos nroff para renderizar una página de manual y examinamos la salida usando cat -A:

```
[me@linuxbox ~]$ zcat /usr/share/man/man1/ls.1.gz | nroff -man | cat -A | head
LS(1)                               Comandos de usuario      LS(1)
$                                $
$                                $
n^HNA^Hum^HME^he$
          ls - Contenido del directorio de listas $
$                                $
$^HSY^HYN^HNO^HOP^HPS^HSI^HIS^HS$
        l^Hls^Hs [ _^HO_ ^HP_ ^HT_ ^HI_ ^HO_ ^HN]... [ _^HF_ ^HI_ ^HL_ ^HE]...$
```

Los caracteres ^H (CTRL-H) son los espacios inversos utilizados para crear el efecto de negrita. Del mismo modo, también podemos ver una secuencia de retroceso/guion bajo utilizada para producir subrayado.

### *Impresoras gráficas*

El desarrollo de las interfaces gráficas de usuario supuso cambios importantes en la tecnología de las impresoras.

A medida que las computadoras pasaron a pantallas más basadas en imágenes, la impresión pasó de las técnicas basadas en caracteres a las gráficas. Esto se vio facilitado por la llegada de la impresora láser de bajo costo, que, en lugar de imprimir caracteres fijos, podía imprimir pequeños puntos en cualquier lugar del área imprimible de la página. Esto hizo posible la impresión de fuentes proporcionales (como las que usan los tipógrafos), e incluso fotografías y diagramas de alta calidad.

Sin embargo, pasar de un esquema basado en caracteres a un esquema gráfico presentó un desafío técnico formidable. He aquí por qué: El número de bytes necesarios para llenar una página usando una impresora basada en caracteres se puede calcular de esta manera (suponiendo 60 líneas por página, cada una con 80 caracteres):  $60 \times 80 = 4.800$  bytes.

En comparación, una impresora láser de 300 puntos por pulgada (DPI) (suponiendo un área de impresión de 8 por 10 pulgadas por página) requiere  $(8 \times 300) \times (10 \times 300) \div 8 = 900.000$  bytes.

Muchas de las lentes de PC simplemente no podían manejar los casi 1 megabyte de datos necesarios para imprimir una página completa en una impresora láser, por lo que estaba claro que se necesitaba un invento inteligente.

Ese invento resultó ser el lenguaje de descripción de páginas. Un *Lenguaje de descripción de página (PDL)* es un lenguaje de programación que describe el contenido de una página. Básicamente dice: "Ve a esta posición, dibuja el personaje *un* en Helvética de 10 puntos, ve a esta posición,

....." hasta que todo en la página esté Descrito. La primera PDL importante fue PostScript de Adobe Systems, que todavía se usa ampliamente en la actualidad. El lenguaje PostScript es un lenguaje de programación completo diseñado para tipografía y otros tipos de gráficos e imágenes. Incluye soporte incorporado para 35 fuentes estándar de alta calidad, además de la capacidad

para aceptar definiciones de fuente adicionales en tiempo de ejecución. Al principio, el soporte para Post-Script estaba integrado en las propias impresoras. Esto resolvió el problema de transmisión de datos . Si bien el programa PostScript típico era detallado en comparación con el flujo de bytes simple de las impresoras basadas en caracteres, era mucho más pequeño que el número de bytes necesarios para representar toda la página impresa.

Una *impresora PostScript* aceptaba un programa PostScript como entrada. La impresora contenía su propio procesador y memoria (a menudo haciendo de la impresora una computadora más poderosa que la computadora a la que estaba conectada) y ejecutaba un programa especial llamado *intérprete PostScript*, que leía el programa PostScript entrante y procesaba los resultados en la memoria interna de la impresora, formando así el patrón de bits (puntos) que se transferían al papel. El nombre genérico para este proceso de renderizar algo en un patrón de bits grande (llamado mapa de *bits*) es *procesador de imágenes rasterizadas o RIP*.

Con el paso de los años, tanto los ordenadores como las redes se volvieron mucho más rápidos. Esto permitió que el RIP pasara de la impresora a la computadora host, lo que, a su vez, permitió que las impresoras de alta calidad fueran mucho menos costosas.

Hoy en día, muchas impresoras todavía aceptan secuencias basadas en caracteres, pero muchas impresoras de bajo costo no lo hacen. Se basan en el RIP de la computadora host para proporcionar un flujo de bits para imprimir como puntos. También hay algunas impresoras PostScript.

## Impresión con Linux

Los sistemas Linux modernos emplean dos paquetes de software para realizar y administrar la impresión. El primero, CUPS (Common Unix Printing System), proporciona controladores de impresión y gestión de trabajos de impresión; el segundo, Ghostscript, un intérprete de PostScript, actúa como RIP.

CUPS gestiona las impresoras mediante la creación y el mantenimiento de colas de impresión. Como comentamos en nuestra breve lección de historia, la impresión Unix se diseñó originalmente para gestionar una impresora centralizada compartida por varios usuarios. Dado que las impresoras son lentas por naturaleza, en comparación con los ordenadores que se alimentan. Ellos, los sistemas de impresión necesitan una forma de programar múltiples trabajos de impresión y mantener las cosas organizadas. CUPS también tiene la capacidad de reconocer diferentes tipos de datos (dentro de lo razonable) y puede convertir archivos en un formulario imprimible.

## Preparación de archivos para impresión

Como usuarios de la línea de comandos, estamos interesados principalmente en imprimir texto, aunque ciertamente también es posible imprimir otros formatos de datos.

### *pr: convertir archivos de texto para imprimir*

Vimos un poco las relaciones públicas en el capítulo anterior. Ahora examinaremos algunas de sus muchas opciones utilizadas junto con la impresión. En nuestra historia de la impresión, vimos que las impresoras basadas en caracteres utilizan fuentes monoespaciadas, lo que da como resultado

Número fijo de caracteres por línea y líneas por página. PR se utiliza para ajustar el texto para que se ajuste a un tamaño de página específico, con encabezados y márgenes de página opcionales. En la tabla 22-1 se resumen las opciones más utilizadas.

**Tabla 22-1: Opciones comunes de pr**

Opción	Descripción
+primero[:último]	Genera un rango de páginas que comienzan con <i>Primero</i> y, opcionalmente, terminando con <i>último</i> .
-Columnas	Organice el contenido de la página en el número de columnas especificado por <i>Columnas</i> .
-un	De forma predeterminada, la salida de varias columnas se muestra verticalmente. Al agregar el atributo -un (a través), el contenido se enumera horizontalmente.
-d	Salida de doble espacio.
-D <i>formatFormat</i>	La fecha que se muestra en los encabezados de página usando <i>formato</i> . Consulte la página del manual para ver el fecha para obtener una descripción de la cadena de formato.
-f	Utilice feeds de formularios en lugar de retornos de carro a páginas separadas.
-h <i>headerIn</i>	la parte central del encabezado de la página, use <i>encabezado</i> en su lugar, el nombre del archivo que se está procesando.
-l <i>lengthSet</i>	longitud de página a <i>largo</i> . El valor predeterminado es de 66 líneas (carta de EE. UU. a 6 líneas por pulgada).
-n	Rectas numéricas.
-o <i>compensar</i>	Crear un margen izquierdo <i>compensar</i> caracteres de ancho.
-w <i>Ancho</i>	Establezca el ancho de página en <i>Ancho</i> . El valor predeterminado es de 72 caracteres.

PR se usa a menudo en tuberías como filtro. En este ejemplo, produciremos una lista de directorio de */usr/bin* y la formatearemos en una salida paginada de tres columnas usando pr:

```
[me@linuxbox ~]$ ls /usr/bin | pr -3 -w 65 | cabeza
```

2012-02-18 14:00		Página 1
[	apturl	bsd-write
411 horas	Ar	Bsh
A2P	arecord	btcflash
A2PS	arecordmidi	Amigo de insectos
A2PS-LPR-Envoltura	arca	buildhash

## Envío de un trabajo de impresión a una impresora

La suite de impresión CUPS admite dos métodos de impresión utilizados históricamente en sistemas tipo Unix. Un método, llamado Berkeley o LPD (usado en la versión de Berkeley Software Distribution de Unix), usa el programa lpr; el otro método, llamado SysV (de la versión System V de Unix), usa el programa lp. Ambos programas hacen más o menos lo mismo. Elegir uno sobre el otro es una cuestión de gusto personal.

### *lpr: imprimir archivos (estilo Berkeley)*

El programa lpr se puede utilizar para enviar archivos a la impresora. También se puede utilizar en tuberías, ya que acepta entradas estándar. Por ejemplo, para imprimir los resultados de nuestra lista de directorios de varias columnas anterior, podríamos hacer lo siguiente:

---

```
[me@linuxbox ~]$ ls /usr/bin | pr -3 | lpr
```

---

El informe se enviaría a la impresora predeterminada del sistema. Para enviar el archivo a una impresora diferente, la opción -P se puede usar de la siguiente manera:

```
lpr -P printer_name
```

donde *printer\_name* es el nombre de la impresora deseada. Para ver una lista de los impresores conocidos por el sistema:

---

```
[me@linuxbox ~]$ lpstat -a
```

---

**Nota:** *Muchas distribuciones de Linux le permiten definir una "impresora" que genera archivos en PDF, en lugar de imprimir en la impresora física. Esto es muy útil para experimentar con comandos de impresión. Compruebe el programa de configuración de su impresora para ver si es compatible con esta configuración. En algunas distribuciones, es posible que necesite instalar paquetes adicionales (como cups-pdf) para habilitar esta capacidad.*

La Tabla 22-2 muestra algunas de las opciones comunes para lpr.

Tabla 22-2: Opciones comunes de lpr

Opción	Descripción
- # <i>número</i>	Establezca el número de copias en <i>número</i> .
-p	Imprima cada página con un encabezado sombreado con la fecha, la hora, el nombre del trabajo y el número de página. Esta opción llamada " impresión bonita" se puede utilizar cuando se imprimen archivos de texto.
-P <i>printerEspecificar</i>	El nombre de la impresora utilizada para la salida. Si no se especifica ninguna impresora, se utiliza la impresora predeterminada del sistema.
-r	Elimine los archivos después de imprimirlos. Esto sería útil

para programas que producen archivos temporales de salida de impresora.

## **lp: Imprimir archivos (estilo System V)**

Al igual que lpr, lp acepta archivos o entradas estándar para imprimir. Se diferencia de lpr en que admite un conjunto de opciones diferente (y un poco más sofisticado). En la tabla 22-3 se enumeran las opciones comunes.

**Tabla 22-3: Opciones comunes de lp**

Opción	Descripción
-d <i>printerSet</i>	el destino (impresora) a <i>impresora</i> . Si no d , se utiliza la impresora predeterminada del sistema.
-n <i>número</i>	Establezca el número de copias en <i>número</i> .
-o paisaje	Establezca la salida en orientación horizontal.
-o Diagrama de ajuste	Escale el archivo para que se ajuste a la página. Esto es útil cuando se imprimen imágenes, como archivos JPEG.
-o escalado= <i>número</i>	Escalar archivo a <i>número</i> . El valor de 100 rellena la página. Los valores inferiores a 100 se reducen, mientras que los valores superiores a 100 hacen que el archivo se imprima en varias páginas.
-o cpi= <i>número</i> Predeterminado	Establezca los caracteres de salida por pulgada en <i>número</i> . Es 10.
-o lpi= <i>número</i> predeterminado es 6.	Establezca las líneas de salida por pulgada en <i>número</i> . El valor
-o page-bottom= <i>puntos</i> -o página-izquierda= <i>puntos</i> -o page-right= <i>puntos</i> -o page-top= <i>puntos</i>	Establezca los márgenes de página. Los valores se expresan en <i>puntos</i> , una unidad de medida tipográfica. Hay 72 puntos por pulgada.
-P <i>pagesEspecificar</i>	la lista de páginas. <i>Páginas</i> puede expresarse como una lista separada por comas y/o un rango, por ejemplo 1,3,5,7-10.

Volveremos a producir nuestra lista de directorio, esta vez imprimiendo 12 CPI y 8 LPI con un margen izquierdo de media pulgada. Tenga en cuenta que tenemos que ajustar las opciones de pr para tener en cuenta el nuevo tamaño de página:

```
[me@linuxbox ~]$ ls /usr/bin | pr -4 -w 90 -l 88 | lp -o page-left=36 -o cpi= 12  
-o lpi=8
```

Esta canalización genera una lista de cuatro columnas con un tipo más pequeño que el predeterminado. El aumento del número de caracteres por pulgada nos permite colocar más columnas en la página.

## Otra opción: a2ps

El programa a2ps es interesante. Como podemos deducir por su nombre, es un programa de conversión de formatos, pero también es mucho más. Su nombre originalmente significaba *ASCII a PostScript*, y se utilizaba para preparar archivos de texto para imprimir en impresoras PostScript. A lo largo de los años, sin embargo, las capacidades del programa han crecido, y ahora su nombre significa *cualquier cosa para PostScript*. Aunque su nombre sugiere un programa de conversión de formatos, en realidad se trata de un programa de impresión. Envía su salida predeterminada, en lugar de la salida estándar, a la impresora predeterminada del sistema. El comportamiento predeterminado del programa es el de una "impresora bonita", lo que significa que mejora la apariencia de la salida. Podemos usar el programa para crear un archivo PostScript en nuestro escritorio:

```
[me@linuxbox ~]$ ls /usr/bin | pr -3 -t | a2ps -o ~/Desktop/ls.ps -L 66
[stdin (llano): 11 páginas en 6 hojas]
[Total: 11 páginas en 6 hojas] guardadas en el archivo '/home/me/Desktop/ls.ps'
```

Aquí filtramos el flujo con pr, usando la opción -t (omitar encabezados y pies de página) y luego, con a2ps, especificando un archivo de salida (opción -o) y 66 líneas por página (opción -L) para que coincida con la paginación de salida de pr. Si visualizamos el archivo resultante con un visor de archivos adecuado, veremos la salida que se muestra en la Figura 22-1.

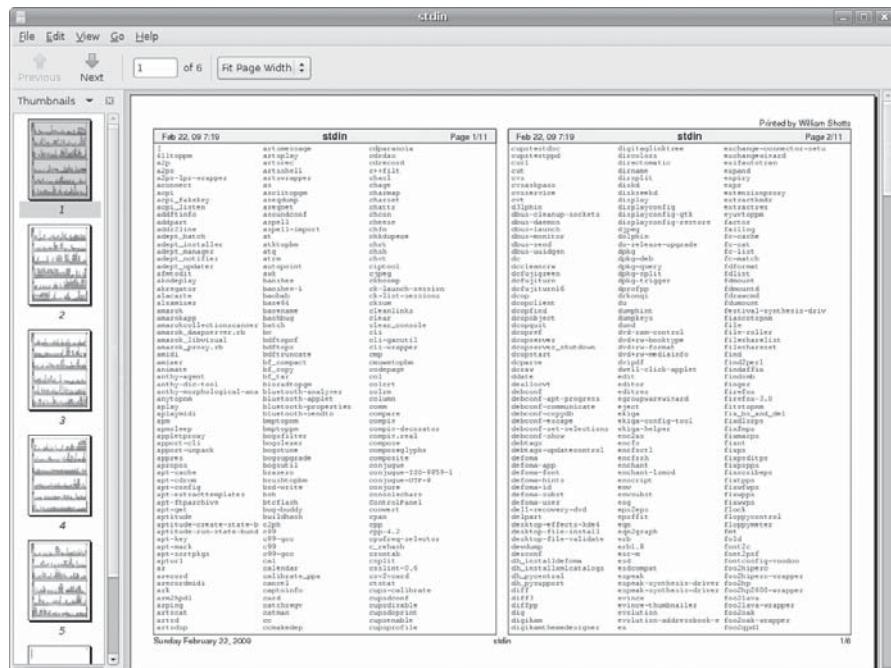


Figura 22-1: Visualización de la salida a2ps

Como podemos ver, el diseño de salida predeterminado es el formato "dos arriba". Esto hace que el contenido de dos páginas se imprima en cada hoja de papel. A2PS también aplica buenos encabezados y pies de página.

A2PS tiene muchas opciones. En la tabla 22-4 se resumen estos temas.

**Tabla 22-4: Opciones de a2ps**

Opción	Descripción
--centro-título <i>Mensaje de texto</i>	Establezca el título de la página central en <i>Mensaje de texto</i> .
--Columnas <i>número</i>	Organizar las páginas en <i>número</i> Columnas. El valor predeterminado es 2.
--pie de página <i>Mensaje de texto</i>	Establezca el pie de página en <i>Mensaje de texto</i> .
--adivinar	Indique los tipos de archivos proporcionados como argumentos. Desde A2PS intenta convertir y formatear todo tipo de datos, esta opción puede ser útil para predecir lo que A2PS lo hará cuando se le dé un archivo en particular.
--pie de página izquierdo <i>Mensaje de texto</i>	Establezca el pie de página izquierdo en <i>Mensaje de texto</i> .
--título-izquierdo <i>Mensaje de texto</i>	Establezca el título de la página izquierda en <i>Mensaje de texto</i> .
--números-de-línea=intervalo	Rectas numéricas de salida cada <i>intervalo</i> líneas.
--lista=valores predeterminados	Mostrar la configuración predeterminada.
--lista=tema	Configuración de pantalla para <i>tema</i> Dónde <i>tema</i> es uno de los siguientes: delegaciones (programas externos que se utilizarán para convertir datos), codificaciones, características, variables, medios (tamaños de papel y similares), ppd (descripciones de impresoras PostScript), impresoras, prólogos (partes de código que tienen el prefijo de salida normal), hojas de estilo u opciones de usuario.
--Páginas gama	Imprima páginas en rango.
--pie de página derecho <i>Mensaje de texto</i>	Establezca el pie de página derecho en <i>Mensaje de texto</i> .
--Derecha-título <i>Mensaje de texto</i>	Establezca el título de la página derecha en <i>Mensaje de texto</i> .
--Filas <i>número</i>	Organizar las páginas en <i>número</i> Filas. El valor predeterminado es 1.
-B	Sin encabezados de página.
-b <i>Mensaje de texto</i>	Establezca el encabezado de la página en <i>Mensaje de texto</i>

*texto.*

---

**-f tamaño**

Uso *tamaño* fuente de punto.

---

**-l numberSet**

caracteres por línea a *número*. Esto y el -L (abajo) se puede utilizar para hacer archivos paginados con otros programas, como Pr, encajan correctamente en la página.

---

(continuación)

Cuadro 22-4 (continuación )

Opción	Descripción
-L <i>número</i>	Establezca las líneas por página en <i>número</i> .
-M <i>nombre</i>	Utilice el nombre del medio, por ejemplo, A4.
-n <i>número</i>	Salida <i>número</i> copias de cada página.
-o <i>archivo</i>	Enviar salida a <i>archivo</i> . Si <i>archivo</i> se especifica como -, utilice la salida estándar.
-P <i>printerUsar impresora</i>	. Si no se especifica una impresora, se utiliza la impresora predeterminada del sistema.
-R	Orientación vertical
-r	Paisaje orientación
-T <i>número</i>	Establezca las tabulaciones en cada <i>número</i> Caracteres.
-u <i>Mensaje de texto</i>	Páginas subyacentes (marca de agua) con <i>Mensaje de texto</i> .

Esto es solo un resumen. A2PS tiene varias opciones más.

**Nota:** *a2ps todavía está en desarrollo activo. Durante mis pruebas, noté un comportamiento diferente en varias distribuciones. En CentOS 4, la salida siempre iba a la salida estándar de forma predeterminada. En CentOS 4 y Fedora 10, la salida predeterminada es A4, a pesar de que el programa está configurado para usar medios de tamaño carta de forma predeterminada. Podría superar estos problemas especificando explícitamente la opción deseada. En Ubuntu 8.04, a2ps funcionó como se documentó.*

También tenga en cuenta que hay otro formateador de salida que es útil para convertir texto en PostScript. Llamado *enscript*, puede realizar muchos de los mismos tipos de trucos de formato e impresión, pero a diferencia de *a2ps*, solo acepta entradas de texto.

## Supervisión y control de trabajos de impresión

Como los sistemas de impresión Unix están diseñados para manejar múltiples trabajos de impresión de múltiples usuarios, CUPS está diseñado para hacer lo mismo. A cada impresora se le asigna una *cola de impresión*, donde los trabajos se estacionan hasta que se pueden *poner en cola* en la impresora. CUPS suministra varios programas de línea de comandos que se utilizan para administrar el estado de la impresora y las colas de impresión. Al igual que los programas *lpr* y *lp*, estos programas de gestión se basan en los programas correspondientes de los sistemas de impresión Berkeley y System V.

***lpstat: muestra el estado del sistema de impresión***

El programa lpstat es útil para determinar los nombres y la disponibilidad de las impresoras en el sistema. Por ejemplo, si tuviéramos un sistema con un

impresora (llamada *impresora*) y una impresora virtual PDF (llamada *PDF* ), podríamos comprobar su estado de la siguiente manera:

---

```
[me@linuxbox ~]$ lpstat -a
PDF aceptando solicitudes desde Mon 05 Dec 2011 03:05:59 PM EST
impresora aceptando solicitudes desde Tue 21 Feb 2012 08:43:22 AM EST
```

---

Además, podríamos determinar una descripción más detallada de la configuración del sistema de impresión de esta manera:

---

```
[me@linuxbox ~]$ lpstat -s
Destino predeterminado del sistema:
dispositivo de impresión para PDF:
cups-pdf:/ 
Dispositivo para impresora: ipp://print-server:631/printers/printer
```

---

En este ejemplo, vemos que *printer* es la impresora predeterminada del sistema y que es una impresora de red que utiliza el Protocolo de impresión de Internet (*ipp://*) conectada a un sistema llamado *print-server*.

Las opciones comúnmente utilizadas se describen en la Tabla 22-5.

Tabla 22-5: Opciones comunes de *lpstat*

Opción	Descripción
-a [ <i>impresora...</i> ] Nota	Mostrar el estado de la cola de impresión para <i>impresora</i> . Si no se especifica ninguna
	que este es el estado de la capacidad de la cola de impresión para aceptar trabajos, no el estado de las impresoras físicas.
-d	impresora, se muestran todas las colas de impresión.
sistema.	Muestra el nombre de la impresora predeterminada del sistema.
-p [ <i>impresora...</i> ] no	Mostrar el estado de la propiedad especificada <i>impresora</i> . Si no se especifica ninguna
	impresoras, se muestran todas las impresoras.
-r	Muestra el estado del servidor de impresión.
-s	Mostrar un resumen de estado.
-t	Mostrar un informe de estado completo.

### *lpq: mostrar el estado de la cola de la impresora*

Para ver el estado de una cola de impresora, se utiliza el programa *lpq*. Esto nos permite ver el estado de la cola y los trabajos de impresión que contiene. A continuación, se muestra un ejemplo de una cola vacía para una impresora predeterminada del sistema denominada *printer* :

---

```
[me@linuxbox ~]$ lpq
La impresora
está lista No hay
entradas
```

---

Si no especificamos una impresora (usando la opción `-P`), se muestra la impresora predeterminada del sistema. Si enviamos un trabajo a la impresora y luego miramos la cola, lo veremos listado:

---

```
[me@linuxbox ~]$ ls *.txt | pr -3 | lp El ID  
de solicitud es Printer-603 (1 archivo(s))  
[me@linuxbox ~]$ lpq  
La impresora está lista e imprimiendo  
Rango Dueño Trabajo Archivo(s) Tamaño total  
Activa yo 603 (stdin) 1024 bytes
```

---

### *lprm y cancel: cancelar trabajos de impresión*

CUPS suministra dos programas que se utilizan para finalizar los trabajos de impresión y eliminarlos de la cola de impresión. Uno es el estilo Berkeley (`lprm`) y el otro es el Sistema V (`cancelar`). Difieren ligeramente en las opciones que admiten, pero hacen básicamente lo mismo. Usando nuestro trabajo de impresión anterior como ejemplo, podríamos detener el trabajo y eliminarlo de esta manera:

---

```
[me@linuxbox ~]$ cancelar  
603 [me@linuxbox ~]$ La  
impresora LPQ está lista  
Sin entradas
```

---

Cada comando tiene opciones para eliminar todos los trabajos que pertenecen a un usuario en particular, una impresora en particular y varios números de trabajo. Sus respectivas páginas de manual tienen todos los detalles.

# 23

## COMPILACIÓN DE PROGRAMAS

En este capítulo, veremos cómo construir programas compilando código fuente. La disponibilidad del código fuente es la libertad esencial que hace posible Linux. Todo el ecosistema de desarrollo de Linux se basa en el libre intercambio entre desarrolladores.

### Para muchos ordenadores de sobremesa

usuarios, compilar es un arte perdido. Solía ser bastante común, pero hoy en día, los proveedores de distribución mantienen enormes repositorios de binarios precompilados, listos para descargar y usar. En el momento de escribir este artículo, el repositorio de Debian (uno de los más grandes de todas las distribuciones) contiene casi 23.000 paquetes.

Entonces, ¿por qué compilar software? Hay dos razones:

- **Disponibilidad.** A pesar del número de programas precompilados en los repositorios de distribución, es posible que algunas distribuciones no incluyan todas las aplicaciones deseadas. En este caso, la única forma de obtener el programa deseado es compilarlo desde el código fuente.
- **Puntualidad.** Si bien algunas distribuciones se especializan en versiones de programas de vanguardia, muchas no lo hacen. Esto

significa que para tener la última versión de un programa, es necesario compilar.



Compilar software a partir de código fuente puede llegar a ser muy complejo y técnico, mucho más allá del alcance de muchos usuarios. Sin embargo, muchas tareas de compilación son bastante fáciles e implican solo unos pocos pasos. Todo depende de la edad del paquete. Vamos a ver un caso muy simple con el fin de proporcionar una visión general del proceso y como punto de partida para aquellos que deseen emprender un estudio más profundo.

Introduciremos un nuevo comando:

- make: utilidad para mantener programas.

## ¿Qué es compilar?

En pocas palabras, la compilación es el proceso de traducir el *código fuente* (la descripción legible por humanos de un programa escrito por un programador) al lenguaje nativo del procesador de la computadora.

El procesador (o *CPU*) del ordenador funciona a un nivel muy elemental, ejecutando programas en lo que se denomina *lenguaje de máquina*. Se trata de un código numérico que describe operaciones muy pequeñas, como "agregar este byte", "apuntar a esta ubicación en la memoria" o "copiar este byte". Cada una de estas instrucciones se expresa en binario (unos y ceros). Los primeros programas informáticos se escribieron utilizando este código numérico, lo que puede explicar por qué se decía que los programadores que lo escribieron fumaban mucho, bebían galones de café y usaban gafas gruesas.

Este problema fue superado por el advenimiento del *lenguaje ensamblador*, que reemplazó los códigos numéricos con mnemotécnicos de caracteres (ligeramente) más fáciles de usar, como CPY (para copiar) y MOV (para mover). Los programas escritos en lenguaje ensamblador son procesados en lenguaje de máquina por un programa llamado *ensamblador*. El lenguaje ensamblador se sigue utilizando hoy en día para ciertas tareas de programación especializadas, como *controladores de dispositivos y sistemas integrados*.

A continuación, llegamos a lo que se denomina lenguajes de *programación de alto nivel*. Se llaman así porque permiten que el programador se preocupe menos por los detalles de lo que está haciendo el procesador y más por resolver el problema en cuestión. Los primeros (desarrollados durante la década de 1950) incluyeron *FORTRAN* (diseñado para tareas científicas y técnicas) y *COBOL* (diseñado para aplicaciones comerciales). Ambos todavía están en uso limitado en la actualidad.

Si bien hay muchos lenguajes de programación populares, dos predominan. La mayoría de los programas escritos para sistemas modernos están escritos en C o C++. En los siguientes ejemplos, compilaremos un programa en C.

Los programas escritos en lenguajes de programación de alto nivel se convierten al lenguaje de máquina procesándolos con otro programa, llamado *compilador*. Algunos compiladores traducen instrucciones de alto nivel al lenguaje ensamblador y luego utilizan un ensamblador para realizar la etapa final de la traducción al lenguaje de máquina.

Un proceso que se utiliza a menudo junto con la compilación se denomina *vinculación*. Los programas realizan muchas tareas comunes. Tomemos, por ejemplo, abrir un archivo.

Muchos programas realizan esta tarea, pero sería un desperdicio que cada programa implementara su propia rutina para abrir archivos. Tiene más sentido tener una sola pieza de programación que sepa cómo abrir archivos y permitir que todos los programas que lo necesiten lo compartan. Proporcionar soporte para tareas comunes se lleva a cabo mediante lo que se denomina *bibliotecas*. Contienen múltiples *rutinas*, cada una de las cuales realiza una tarea común que varios programas pueden compartir. Si miramos en los *directorios* `/lib` y `/usr/lib`, podemos ver dónde viven muchos de ellos. Se utiliza un programa llamado *enlazador* para formar las conexiones entre la salida del compilador y las bibliotecas que requiere el programa compilado.

El resultado final de este proceso es el *archivo de programa ejecutable*, listo para su uso.

### **¿Están compilados todos los programas?**

No. Como hemos visto, algunos programas, como los scripts de shell, no requieren compilación, sino que se ejecutan directamente. Estos están escritos en lo que se conoce como *scripting* o lenguajes *interpretados*. Estos lenguajes, que han crecido en popularidad en los últimos años, incluyen Perl, Python, PHP, Ruby y muchos otros.

Los lenguajes de scripts son ejecutados por un programa especial llamado *intérprete*. Un intérprete ingresa el archivo de programa y lee y ejecuta cada instrucción contenida en él. En general, los programas interpretados se ejecutan mucho más lentamente que los programas compilados. Esto se debe a que cada instrucción de código fuente en un programa interpretado se traduce cada vez que se lleva a cabo, mientras que con un programa compilado, una instrucción de código fuente se traduce solo una vez, y esta traducción se registra permanentemente en el archivo ejecutable final.

Entonces, ¿por qué son tan populares los lenguajes interpretados? Para muchas tareas de programación, los resultados son "suficientemente rápidos", pero la verdadera ventaja es que generalmente es más rápido y fácil desarrollar programas interpretados que programas compilados. Los programas generalmente se desarrollan en un ciclo repetitivo de código, compilación, prueba. A medida que un programa crece en tamaño, la fase de compilación del ciclo puede llegar a ser bastante larga. Los lenguajes interpretados eliminan el paso de compilación y, por lo tanto, aceleran el desarrollo del programa.

## **Compilar un programa en C**

Vamos a compilar algo. Sin embargo, antes de hacer eso, vamos a necesitar algunas herramientas como el compilador, el enlazador y make. El compilador de C, utilizado casi universalmente en el entorno Linux, se llama `gcc` (GNU C Compiler), escrito originalmente por Richard Stallman. La mayoría de las distribuciones no instalan `gcc` de forma predeterminada. Podemos comprobar si el compilador está presente de la siguiente manera:

---

```
[me@linuxbox ~]$ que gcc
```

/usr/bin/gcc

---

Los resultados de este ejemplo indican que el compilador está instalado.

**Nota:** Su distribución puede tener un metapaquete (una colección de paquetes) para el desarrollo de software. Si es así, considere instalarlo si tiene la intención de compilar programas en su sistema. Si su sistema no proporciona un metapaquete, intente instalar los paquetes gcc y make. En muchas distribuciones, son suficientes para llevar a cabo el ejercicio a continuación.

### Obtención del código fuente

Para nuestro ejercicio de compilación, vamos a compilar un programa del Proyecto GNU llamado dicción. Este pequeño y práctico programa comprueba la calidad y el estilo de escritura de los archivos de texto . A medida que avanzan los programas, es bastante pequeño y fácil de construir.

Siguiendo la convención, primero vamos a crear un directorio para nuestro código fuente llamado *src* y luego descargar el código fuente en él usando ftp:

```
[me@linuxbox ~]$ mkdir src
[me@linuxbox ~]$ cd src [me@linuxbox
src]$ ftp ftp.gnu.org Conectado a
ftp.gnu.org.
Servidor FTP 220 GNU listo.
Nombre (ftp.gnu.org:me): anónimo
230 Inicio de sesión exitoso.
El tipo de sistema remoto es
UNIX.
Uso del modo binario para transferir
archivos. ftp> cd gnu/dicion
250 Directorio cambiado con éxito.
ftp> ls
200 PORT comando exitoso. Considera la posibilidad de
utilizar PASV. 150 Aquí viene la lista del directorio.
-rw-r--r--    1 1003 65534   68940 28 de agosto de 1998 diction-0.7.tar.gz
-rw-r--r--    1 1003 65534   90957 4 de marzo de 2002 diction-1.02.tar.gz
-rw-r--r--    1 1003 65534 141062 17 de septiembre de 2007
diction-1.11.tar.gz 226 Directorio enviar OK.
FTP> obtener diction-1.11.tar.gz
local: diction-1.11.tar.gz remoto: diction-1.11.tar.gz
comando 200 PORT se ha realizado correctamente.
Considera la posibilidad de utilizar PASV.
150 Apertura de la conexión de datos en modo BINARIO para diction-1.11.tar.gz
(141062 bytes).
226 Envío de archivo OK.
141062 bytes recibidos en 0,16 segundos (847,4
KB/s) ftp> bye
221 Adiós.
[me@linuxbox src]$ ls
diction-1.11.tar.gz
```

**Nota:** Dado que somos el mantenedor de este código fuente mientras lo compilamos, lo mantendremos en

~/src. El código fuente instalado por su distribución se instalará en /usr/src, mientras que el código fuente destinado a ser utilizado por múltiples usuarios generalmente se instala en /usr/local/src.

Como podemos ver, el código fuente generalmente se suministra en forma de un archivo tar comprimido. A veces llamado *tarball*, este archivo

contiene el *árbol de fuentes*, o jerarquía de directorios y archivos que componen el código fuente. Después de llegar al sitio FTP, examinamos la lista de archivos tar disponibles y seleccionamos la versión más reciente para descargar. Usando el comando `get` dentro de `ftp`, copiamos el archivo del servidor FTP a la máquina local.

Una vez descargado el archivo tar, hay que descomprimirlo. Esto se hace con el programa tar:

---

```
[me@linuxbox src]$ tar xzf diccion-1.11.tar.gz
[me@linuxbox src]$ ls
dicción-1.11          diccion-1.11.tar.gz
```

---

**Nota:** El programa de dicción, como todo el software del Proyecto GNU, sigue ciertos estándares para el empaquetado del código fuente. La mayoría del resto del código fuente disponible en el ecosistema de Linux también sigue este estándar. Un elemento del estándar es que cuando se desempaqueteta el archivo tar del código fuente, se creará un directorio que contiene el árbol de fuentes y que este directorio se llamará *project-x.xx*, conteniendo así tanto el nombre del proyecto como su número de versión. Este esquema permite una fácil instalación de múltiples versiones del mismo programa. Sin embargo, a menudo es una buena idea examinar la disposición del árbol antes de desempacarlo. Algunos proyectos no crearán el directorio, sino que entregarán los archivos directamente en el directorio actual. Esto hará un desastre en su directorio *src*, que de otro modo estaría bien organizado . Para evitar esto, utilice el siguiente comando para examinar el contenido de la carpeta Archivo tar:

```
tar tzvf tarfile | cabeza
```

### Examinando el árbol de fuentes

Al desempaquetar el archivo tar, se crea un nuevo directorio, llamado *dicción-1.11*. Este directorio contiene el árbol de fuentes. Echemos un vistazo al interior:

---

```
[me@linuxbox src]$ cd dicción-1.11
[me@linuxbox dicción-1.11]$ ls
config.adivinar  dicción.c      getopt.c      NL
config.h.in       dicción.pot    getopt.h      nl.po
config.sub        dicción.spec   getopt_int.h LÉAME
configurar       diction.spec.in INSTALAR
                  sentencia.c  configure.in diction.texi.in
install-sh        sentencia.h
COPIADOR         en            Makefile.in   style.1.in
de               en_GB         Misc.c        estilo.c
de.po            en_GB.po     misc.h        prueba
diction.1.in     getopt1.c    NOTICIA
```

---

En él, vemos una serie de archivos. Los programas que pertenecen al Proyecto GNU, así como muchos otros, proporcionarán los archivos de documentación *README*, *INSTALL*, *NEWS* y *COPYING*. Estos archivos contienen la descripción del programa, información sobre cómo compilarlo e instalarlo y sus términos de licencia. Siempre es una buena idea leer cuidadosamente los archivos *README* e *INSTALL* antes de intentar construir el programa.

Los otros archivos interesantes de este directorio son los que terminan en *.c* y *.h*:

```
[me@linuxbox dicción-1.11]$ ls *.c  
dicción.c getopt1.c getopt.c misc.c frase.c estilo.c  
[me@linuxbox dicción-1.11]$ ls *.h  
getopt.h getopt_int.h misc.h sentencia.h
```

---

Los archivos .c contienen los dos programas C suministrados por el paquete (*estilo* y *dicción*), divididos en módulos. Es una práctica común que los programas grandes se dividan en partes más pequeñas y fáciles de administrar. Los archivos de código fuente son texto ordinario y se pueden examinar con menos:

---

```
[me@linuxbox dicción-1.11]$ menos dicción.c
```

---

Los archivos .h se conocen como *archivos de encabezado*. Estos, también, son textos ordinarios. Los archivos de encabezado contienen descripciones de las rutinas incluidas en un archivo de código fuente o biblioteca. Para que el compilador conecte los módulos, debe recibir una descripción de todos los módulos necesarios para completar todo el programa.

Cerca del comienzo del archivo *diction.c*, vemos esta línea:

---

```
#include "getopt.h"
```

---

Esto indica al compilador que lea el archivo *getopt.h* como lee el código fuente en *diction.c* para "saber" lo que hay en *getopt.c*. El archivo *getopt.c* proporciona rutinas que son compartidas por los programas de estilo y dicción.

Por encima de la declaración `include` para *getopt.h*, vemos algunos otros estados de inclusión como estos:

---

```
#include <regex.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

---

Estos también se refieren a los archivos de encabezado, pero se refieren a los archivos de encabezado que se encuentran fuera del árbol de código fuente actual. Son suministrados por el sistema para apoyar la compilación de cada programa. Si miramos en */usr/include*, podemos verlos:

---

```
[me@linuxbox dicción-1.11]$ ls /usr/include
```

---

Los archivos de encabezado de este directorio se instalaron cuando instalamos el compilador.

### *Construyendo el programa*

La mayoría de los programas se construyen con una secuencia simple de dos comandos:

```
./configure  
make
```

El programa de configuración es un script de shell que se suministra con el árbol de fuentes. Su trabajo es analizar el entorno de compilación. La mayoría del código fuente está diseñado para ser *portátil*. Es decir, está diseñado para construir sobre más de un tipo de sistema tipo Unix. Pero para hacer eso, es posible que el código fuente deba someterse a ligeros

ajustes durante la compilación para adaptarse a las diferencias entre los sistemas. Configure también comprueba que se hayan instalado las herramientas y componentes externos necesarios.

Vamos a ejecutar `configure`. Dado que `configure` no se encuentra donde el shell normalmente espera que se ubiquen los programas, debemos decirle explícitamente al shell su ubicación prefijando el comando con `./`. Esto indica que el programa se encuentra en el directorio de trabajo actual:

---

```
[me@linuxbox dicción-1.11]$ ./configurar
```

---

`configure` generará una gran cantidad de mensajes a medida que pruebe y configure la compilación. Cuando termine, la salida tendrá un aspecto similar al siguiente:

---

```
Comprobación de la presencia de
libintl.h... Sí, comprobando
libintl.h... Sí
Comprobación de la biblioteca que contiene gettext... No se
requiere ninguno Configure: Creación de ./config.status
config.status: creando un archivo
Makefile config.status: creando dicion.1
config.status: creando diction.texi
config.status: creando diction.spec
config.status: creando estilo.1
config.status: creando test/rundiction
config.status: creando config.h
[me@linuxbox dicción-1.11]$
```

---

Lo importante aquí es que no hay mensajes de error. Si lo hubiera, la configuración habría fallado y el programa no se compilaría hasta que se corrigieran los errores.

Vemos configurar creados varios archivos nuevos en nuestro directorio de origen. El más importante es *Makefile*. *Makefile* es un archivo de configuración que indica al programa make exactamente cómo construir el programa. Sin él, la voluntad de Make se negará a correr. *Makefile* es un archivo de texto ordinario, por lo que podemos verlo:

---

```
[me@linuxbox dicción-1.11]$ menos Makefile
```

---

El programa make toma como entrada un *makefile* (que normalmente se denomina *Makefile*), que describe las relaciones y dependencias entre los componentes que componen el programa terminado.

La primera parte del archivo make define las variables que se sustituyen en las secciones posteriores del archivo make. Por ejemplo, vemos la línea

---

CC=	Gcc
-----	-----

---

que define el compilador de C como gcc. Más adelante en el makefile, vemos una instancia en la que se utiliza:

---

dicción:	diccion.o sentencia.o misc.o getopt.o getopt1.o \$(CC) -o \$@ \$(LDFLAGS) dicción.o sentencia.o misc.o \ getopt.o getopt1.o \$(LIBS)
----------	--

---

Aquí se realiza una sustitución y el valor `$(CC)` se reemplaza por `gcc` en tiempo de ejecución.

La mayor parte del archivo make consta de líneas, que definen un

*objetivo*, en este caso la dirección del archivo ejecutable, y los archivos de los que depende. El

Las líneas restantes describen los comandos necesarios para crear el destino a partir de sus componentes. Vemos en este ejemplo que el archivo ejecutable *diction* (uno de los productos finales) depende de la existencia de *diction.o*, *sentence.o*, *misc.o*, *getopt.o* y *getopt1.o*. Más adelante, en el makefile, vemos definiciones de cada uno de estos como objetivos.

---

dicción.o:	dicción.c config.h getopt.h misc.h sentencia.h
getopt.o:	getopt.c getopt.h getopt_int.h
getopt1.o:	getopt1.c getopt.h getopt_int.h
Varios:	misc.c config.h misc.h
sentence.o:	sentence.c config.h misc.h sentence.h
style.o:	estilo.c config.h getopt.h misc.h frase.h

---

Sin embargo, no vemos ningún comando especificado para ellos. Esto se controla mediante un destino general, anteriormente en el archivo, que describe el comando utilizado para compilar cualquier *archivo .c* en un archivo *.o*:

---

.C.O:	\$(CC) -c \$(CPPFLAGS) \$(CFLAGS) \$<
-------	---------------------------------------

---

Todo esto parece muy complicado. ¿Por qué no simplemente enumerar todos los pasos para compilar las partes y terminar con ello? La respuesta se aclarará en un momento. Mientras tanto, ejecutemos, hagamos y construyamos nuestros programas:

---

[me@linuxbox dicción-1.11]\$ hacer

---

El programa make se ejecutará, utilizando el contenido de *Makefile* para guiar sus acciones. Producirá muchos mensajes.

Cuando termine, veremos que todos los objetivos ya están presentes en nuestro directorio:

---

[me@linuxbox dicción-1.11]\$ ls				
config.guess	de.po	en	install-sh	sentencia.c
config.h	dicción	en_GB	Makefile	sentencia.h
config.h.in	dicción.1	en_GB.mo	Makefile.in	sentencia.o
config.log	diction.1.in	en_GB.po	Misc.c	estilo
config.status	dicción.c	getopt1.c	misc.h	estilo.1
config.sub	dicción.o	getopt1.o	Misc.o	style.1.in
configurar	dicción.pot	getopt.c	NOTICIA	estilo.c
configure.in	dicción.spec	getopt.h	NL	estilo.o
COPIADOR	diction.spec.in	getopt_int.h	nl.mo	prueba
de	dicción.texi	getopt.o	nl.po	
de.mo	diction.texi.in	INSTALAR	LÉAME	

---

Entre los archivos, vemos la dicción y el estilo, los programas que nos propusimos construir. ¡Enhorabuena! ¡Acabamos de compilar nuestros primeros programas a partir del código fuente!

Pero solo por curiosidad, corramos de nuevo:

---

[me@linuxbox dicción-1.11]\$ hacer  
make: No hay nada que hacer por 'todos'.

---

Sólo produce este extraño mensaje. ¿Qué pasa? ¿Por qué no vuelve a construir el programa? Ah, esta es la magia de hacer. En lugar de simplemente construir todo de nuevo, haga construcciones solo lo que necesita construirse. Con todos los objetivos presentes, decídate a que no había nada que hacer. Podemos demostrar esto eliminando uno de los objetivos y ejecutando make de nuevo para ver qué hace.

---

```
[me@linuxbox dicción-1.11]$ rm getopt.o  
[me@linuxbox dicción-1.11]$ hacer
```

Vemos que hacemos reconstrucciones *getopt.o* y reenlazamos los programas de dicción y estilo, ya que dependen del módulo que falta. Este comportamiento también señala otra característica importante de make: mantiene los objetivos actualizados. make insiste en que los objetivos sean más nuevos que sus dependencias. Esto tiene mucho sentido, ya que un programador a menudo actualizará un poco de código fuente y luego usará make para construir una nueva versión del producto terminado. make garantiza que se construya todo lo que se necesita compilar en función del código actualizado. Si usamos el programa táctil para "actualizar" uno de los archivos de código fuente, podemos ver que sucede lo siguiente:

```
[me@linuxbox dicción-1.11]$ ls -l dicción getopt.c  
-rwxr-xr-x 1 yo      me    37164 2009-03-05 06:14 dicción  
-rw-r--r-- 1 yo      me    33125 30/03/2007 17:45 getopt.c  
[me@linuxbox dicción-1.11]$ toca getopt.c  
[me@linuxbox dicción-1.11]$ ls -l dicción getopt.c  
-rwxr-xr-x 1 yo      me    37164 2009-03-05 06:14 dicción  
-rw-r--r-- 1 yo      me    33125 05/03/2009 06:23 getopt.c  
[me@linuxbox dicción-1.11]$ hacer
```

Después de realizar ejecuciones, vemos que ha restaurado el destino para que sea más nuevo que la dependencia:

```
[me@linuxbox dicción-1.11]$ ls -l dicción getopt.c  
-rwxr-xr-x 1 yo      me    37164 2009-03-05 06:24 dicción  
-rw-r--r-- 1 me      me    33125 05/03/2009 06:23 getopt.c
```

La capacidad de make para construir inteligentemente solo lo que hay que construir es un gran beneficio para los programadores. Si bien el ahorro de tiempo puede no ser evidente con nuestro proyecto pequeño, es significativo con proyectos más grandes. Recuerde que el kernel de Linux (un programa que se somete a continuas modificaciones y mejoras) contiene varios *millones* de líneas de código.

### *Instalación del programa*

El código fuente bien empaquetado a menudo incluye un objetivo make especial llamado *install*. Este objetivo instalará el producto final en un directorio del sistema para su uso. Por lo general, este directorio es */usr/local/bin*, la ubicación tradicional para el software construido localmente. Sin embargo, este directorio normalmente no es editable por usuarios comunes, por lo que debemos convertirnos en el superusuario para realizar la instalación:

[me@linuxbox diction-1.11]\$ sudo make install

---

Después de realizar la instalación, podemos comprobar que el programa está listo para funcionar:

---

```
[me@linuxbox dicción-1.11]$ que dicción  
/usr/local/bin/dicción  
[me@linuxbox dicción-1.11]$ hombre dicción
```

---

¡Y ahí lo tenemos!

## Nota final

En este capítulo, hemos visto cómo tres comandos simples (`./configure`, `make`, `make install`) se pueden usar para construir muchos paquetes de código fuente. También hemos visto el importante papel que desempeña la marca en el mantenimiento de los programas. El programa `make` se puede usar para cualquier tarea que necesite mantener una relación destino/dependencia, no solo para compilar código fuente.

# **PARTE 4**

**ESCRITURA DE SCRIPTS DE SHELL**



# 24

## ESCRIBIR TU PRIMER GUIÓN

En los capítulos anteriores, hemos reunido un arsenal de herramientas de línea de comandos. Si bien estas herramientas pueden resolver muchos tipos de problemas informáticos, todavía estamos limitados a usarlas manualmente una por una en el línea de comandos. ¿No sería genial si pudiéramos hacer que el caparazón hiciera más trabajo? Podemos. Al unir nuestras herramientas en programas de nuestro propio diseño, el shell puede llevar a cabo secuencias complejas de tareas por sí mismo. Le permitimos hacer esto escribiendo *scripts de shell*.

### ¿Qué son los scripts de shell?

En los términos más simples, un script de shell es un archivo que contiene una serie de comandos. El shell lee este archivo y ejecuta los comandos como si se hubieran introducido directamente en la línea de comandos.

El shell es distintivo, ya que es a la vez una poderosa interfaz de línea de comandos para el sistema y un intérprete de lenguaje de scripting. Como veremos, la mayoría de las cosas que se pueden hacer en la línea de comandos se pueden hacer en scripts, y la mayoría de las cosas que se pueden hacer en scripts se pueden hacer en la línea de comunicación.



Hemos cubierto muchas características del shell, pero nos hemos centrado en las características que se utilizan con mayor frecuencia directamente en la línea de comandos. El shell también proporciona un conjunto de características que generalmente (pero no siempre) se utilizan al escribir programas.

## Cómo escribir un script de shell

Para crear y ejecutar con éxito un script de shell, debemos hacer tres cosas:

1. **Escribe un guión.** Los scripts de shell son archivos de texto ordinarios. Así que necesitamos un editor de texto para escribirlos. Los mejores editores de texto proporcionarán *resaltado de sintaxis*, lo que nos permitirá ver una vista codificada por colores de los elementos del script. El resultado de sintaxis nos ayudará a detectar ciertos tipos de errores comunes. Vim, Gedit, Kate y muchos otros editores son buenos candidatos para escribir guiones.
2. **Haga que el script sea ejecutable.** El sistema es quisquilloso a la hora de no permitir que ningún archivo de texto antiguo sea tratado como un programa, ¡y por una buena razón! Necesitamos establecer los permisos del archivo de script para permitir la ejecución.
3. **Coloque el script en algún lugar donde el shell pueda encontrarlo.** El shell busca automáticamente ciertos directorios para archivos ejecutables cuando no se especifica un nombre de ruta explícito. Para mayor comodidad, colocaremos nuestros scripts en estos directorios.

### Formato de archivo de script

De acuerdo con la tradición de la programación, crearemos un programa de "hola mundo" para demostrar un guión extremadamente simple. Así que encendamos nuestros editores de texto e introduzcamos el siguiente script:

---

```
#!/bin/bash

# Este es nuestro

primer guión. echo

'¡Hola Mundo!'
```

---

La última línea de nuestro script es bastante familiar, solo un comando echo con un argumento de cadena. La segunda línea también es familiar. Parece un comentario que hemos visto en muchos de los archivos de configuración que hemos examinado y editado. Una cosa acerca de los comentarios en los scripts de shell es que también pueden aparecer al final de las líneas, así:

---

```
echo '¡Hola Mundo!' # Esto también es un comentario
```

---

Todo, desde el símbolo # en adelante en la línea, se ignora.

Como muchas cosas, esto también funciona en la línea de comandos:

---

```
[me@linuxbox ~]$ echo '¡Hola Mundo!' # Esto también es un comentario  
¡Hola mundo!
```

---

Aunque los comentarios son de poca utilidad en la línea de comandos, funcionarán.

La primera línea de nuestro guión es un poco misteriosa. Parece que debería ser un comentario, ya que comienza con #, pero parece demasiado intencionado para ser solo eso. El#! La secuencia de caracteres es, de hecho, una construcción especial llamada *shebang*. El shebang se usa para decirle al sistema el nombre del intérprete que debe usarse para ejecutar el script que sigue. Cada script de shell debe incluir esto como su primera línea.

Guardemos nuestro archivo de script como *hello\_world*.

### Permisos ejecutables

Lo siguiente que tenemos que hacer es hacer que nuestro script sea ejecutable. Esto se hace fácilmente usando chmod:

---

```
[me@linuxbox ~]$ ls -l hello_world  
-Rw-r-r-- 1 meme 63 2012-03-07 10:10 hello_world  
[me@linuxbox ~]$ CHMOD 755 hello_world  
[me@linuxbox ~]$ ls -l hello_world  
-rwxr-xr-x 1 yo me 63 07/03/2012 10:10 hello_world
```

---

Hay dos configuraciones de permisos comunes para los scripts: 755 para los scripts que todos pueden ejecutar y 700 para los scripts que solo el propietario puede ejecutar. Tenga en cuenta que los scripts deben ser legibles para poder ser ejecutados.

### Ubicación del archivo de script

Con los permisos establecidos, ya podemos ejecutar nuestro script:

---

```
[me@linuxbox ~]$ ./hello_world  
¡Hola mundo!
```

---

Para que el script se ejecute, debemos preceder el nombre del script con una ruta explícita. Si no lo hacemos, obtenemos esto:

---

```
[me@linuxbox ~]$ hello_world  
bash: hello_world: comando no encontrado
```

---

¿A qué se debe esto? ¿Qué hace que nuestro script sea diferente de otros programas? Resulta que nada. Nuestro guión está bien. Su ubicación es el problema. En el Capítulo 11, discutimos la variable de entorno PATH y su efecto en la forma en que el sistema busca programas ejecutables. En resumen, el sistema busca en una lista de directorios cada vez que necesita encontrar un programa ejecutable, si no hay se especifica la ruta de acceso explícita. Así es como el sistema sabe ejecutar /bin/ls cuando escribimos ls en la línea de comandos. El directorio /bin es uno de los directorios que el sistema busca automáticamente. La lista de directorios se encuentra dentro de una variable de entorno denominada PATH. La variable PATH contiene una lista separada por dos puntos de los directorios en los que se van a realizar búsquedas. Podemos ver el contenido de PATH:

---

```
[me@linuxbox ~]$ echo $PATH  
/inicio/m/bin:/efecto/local/sabin:/efecto/local/bin:/efecto/sabin:/efecto/bin:/sabin:/bin:  
/efecto  
/juegos
```

---



Aquí vemos nuestra lista de directorios. Si nuestro script estuviera ubicado en alguno de los directorios de la lista, nuestro problema estaría resuelto. Observe el primer directorio de la lista, */home/me/bin*. La mayoría de las distribuciones de Linux configuran la variable PATH para que contenga un directorio *bin* en el directorio principal del usuario para permitir que los usuarios ejecuten sus propios programas. Así que si creamos el directorio *bin* y colocamos nuestro script dentro de él, debería empezar a funcionar como otros programas:

---

```
[me@linuxbox ~]$ mkdir bin  
[me@linuxbox ~]$ mv hello_world bin  
[me@linuxbox ~]$ hello_world  
¡Hola mundo!
```

---

Si la variable PATH no contiene el directorio, podemos agregarlo fácilmente incluyendo esta línea en nuestro archivo *.bashrc*:

---

```
export PATH=~/bin:"$PATH"
```

---

Una vez realizado este cambio, entrará en vigor en cada nueva sesión terminal. Para aplicar el cambio a la sesión de terminal actual, debemos hacer que el shell vuelva a leer el archivo *.bashrc*. Esto se puede hacer "abasteciéndose":

---

```
[me@linuxbox ~]$ . .bashrc
```

---

El punto (.) es un sinónimo del comando fuente, un shell que lee un fichero especificado de comandos de shell y lo trata como una entrada del teclado.

**Nota:** Ubuntu agrega automáticamente el directorio *~/bin* a la variable PATH si el directorio *~/bin* existe cuando se ejecuta el archivo *.bashrc* del usuario. Entonces, en los sistemas Ubuntu, si creamos el directorio *~/bin* y luego cerramos la sesión y volvemos a iniciarla, todo funciona.

### Buenas ubicaciones para los guiones

El directorio *~/bin* es un buen lugar para colocar scripts destinados al uso personal. Si escribimos un script que todos en un sistema pueden usar, la ubicación tradicional es */usr/local/bin*. Los scripts destinados a ser utilizados por el administrador del sistema a menudo se encuentran en */usr/local/sbin*. En la mayoría de los casos, el software suministrado localmente, ya sean scripts o programas compilados, debe colocarse en la jerarquía */usr/local* y no en */bin* o */usr/bin*. Estos directorios están especificados por el Estándar de jerarquía del sistema de archivos de Linux para contener solo archivos suministrados y mantenidos por el distribuidor de Linux.

## Más trucos de formato

Uno de los objetivos clave de la escritura de guiones seria es la facilidad de *mantenimiento*, es decir, la facilidad con la que un guión puede ser

modificado por su autor u otros para adaptarlo a las necesidades cambiantes. Hacer que un script sea fácil de leer y entender es una forma de facilitar el mantenimiento.

## *Nombres de opciones largos*

Muchos de los comandos que hemos estudiado incluyen nombres de opciones cortos y largos. Por ejemplo, el comando ls tiene muchas opciones que se pueden expresar en forma corta o larga. Por ejemplo:

---

```
[me@linuxbox ~]$ ls -ad
```

---

y

---

```
[me@linuxbox ~]$ ls --all --directorio
```

---

son comandos equivalentes. En aras de la reducción de la escritura, se prefieren las opciones cortas al introducir opciones en la línea de comandos, pero al escribir scripts, las opciones largas pueden mejorar la legibilidad.

## *Sangría y continuación de línea*

Cuando se emplean comandos largos, la legibilidad se puede mejorar distribuyendo el comando en varias líneas. En el capítulo 17, vimos un ejemplo particularmente largo del comando find:

---

```
[me@linuxbox ~]$ find playground \( -type f -not -perm 0600 -exec chmod 0600
'{}' ';' \) -or \( -type d -not -perm 0700 -exec chmod 0700 '{}' ';' \)
```

---

Este comando es un poco difícil de entender a primera vista. En un script, este comando podría ser más fácil de entender si se escribe de esta manera:

---

```
Buscar patio de recreo \
  \(\ \
    -tipo f \
    -no -perm 0600 \
    -exec chmod 0600 '{}' ';' \
  \) \
  -o \
  \(\ \
    -tipo d \
    -no -perm 0700 \
    -exec chmod 0700 '{}' ';' \
  \)
```

---

Mediante el uso de continuaciones de línea (secuencias de barra invertida) y sangría, la lógica de este comando complejo se describe más claramente al lector. Esta técnica también funciona en la línea de comandos, aunque rara vez se usa, ya que es muy incómodo de escribir y editar. Una diferencia entre un script y la línea de comandos es que un script puede emplear caracteres de tabulación para lograr la sangría, mientras que la línea de comandos no puede porque las tabulaciones se utilizan para activar la finalización.

## CONFIGURING VIM FOR SCRIPT WRITING

El editor de texto vim tiene muchos, muchos ajustes de configuración. Varias opciones comunes pueden facilitar la escritura de guiones.

:syntax on activa el resaltado de sintaxis. Con esta configuración, los diferentes elementos de la sintaxis del shell se mostrarán en diferentes colores al ver un script. Esto es útil para identificar ciertos tipos de errores de programación. También se ve genial. Tenga en cuenta que para que esta función funcione, debe tener instalada una versión completa de vim, y el archivo que está editando debe tener un shebang que indique que el archivo es un script de shell. Si tiene dificultades con :syntax, pruebe :set syntax=sh en su lugar.

:set hlsearch activa la opción para resaltar los resultados de la búsqueda. Digamos que buscamos la palabra *eco*. Con esta opción activada, cada instancia de la palabra se resaltará.

:set tabstop=4 establece el número de columnas ocupadas por un carácter de tabulación. El valor predeterminado es de ocho columnas. Establecer el valor en 4 (que es una práctica común) permite que las líneas largas quepan más fácilmente en la pantalla.

:set autoindent activa la función de sangría automática. Esto hace que vim infine una nueva línea en la misma cantidad que la línea que se acaba de escribir. Esto acelera la escritura en muchos tipos de construcciones de programación. Para detener la sangría, escriba **CTRL-B**.

## Nota final

En este primer capítulo sobre scripting, hemos visto cómo se escriben los scripts y cómo se hacen para que se ejecuten fácilmente en nuestro sistema. También vimos cómo podemos usar varias técnicas de formato para mejorar la legibilidad (y, por lo tanto, la mantenibilidad) de nuestros scripts. En futuros capítulos, la facilidad de mantenimiento surgirá una y otra vez como un principio central en una buena escritura de guiones.

# 25

## INICIAR UN PROYECTO

A partir de este capítulo, comenzaremos a construir un programa. El propósito de este proyecto es ver cómo se utilizan varias características del shell para crear programas y, lo que es más importante, crear *buenos* programas.

El programa que escribiremos es un *generador de informes*. Presentará varias estadísticas sobre nuestro sistema y su estado, y producirá este informe en formato HTML para que podamos verlo con un navegador web.

Los programas generalmente se construyen en una serie de etapas, y cada etapa agrega características y capacidades. La primera etapa de nuestro programa producirá una página HTML mínima que no contiene información del sistema. Eso vendrá más adelante.

### Primera Etapa: Documento Mínimo

Lo primero que debemos saber es el formato de un documento HTML bien formado. Se ve así:

---

```
<HTML>
  <CABEZ
    A>      <TÍTULO>Título de la página</TÍTULO>
```



```
</CABEZ  
A>  
<CUERP Cuerpo de la página.  
O>  
</HTML>  
</CUERP  
O>
```

---

Si introducimos esto en nuestro editor de texto y guardamos el archivo como *foo.html*, podemos usar la siguiente URL en Firefox para ver el archivo: *file:///home/username/foo.html*.

La primera etapa de nuestro programa será capaz de enviar este archivo HTML a la salida estándar. Podemos escribir un programa para hacer esto con bastante facilidad. Comencemos nuestro editor de texto y creamos un nuevo archivo llamado *~/bin/sys\_info\_page*:

---

```
[me@linuxbox ~]$ vim ~/bin/sys_info_page
```

---

Y entraremos en el siguiente programa:

```
#!/bin/bash  
  
# Programa para generar una página de  
  
información del sistema echo "<HTML>"  
echo " <CABEZA>"  
echo "<TÍTULO>Título de la página</TÍTULO>"  
echo " </CABEZA>"  
echo " <CUERPO>"  
echo "Cuerpo de la página".  
echo  
" </CUERPO>"  
echo "</HTML>"
```

---

Nuestro primer intento de resolver este problema contiene un shebang, un comentario (siempre es una buena idea) y una secuencia de comandos de *echo* , uno para cada línea de salida. Después de guardar el archivo, lo haremos ejecutable e intentaremos ejecutarlo:

---

```
[me@linuxbox ~]$ chmod 755 ~/bin/sys_info_page  
[me@linuxbox ~]$ sys_info_page
```

---

Cuando se ejecuta el programa, deberíamos ver el texto del documento HTML en la pantalla, ya que los comandos *echo* en el script envían su salida a la salida estándar. Volveremos a ejecutar el programa y redirigiremos la salida del programa al archivo *sys\_info\_page.html*, para que podamos ver el resultado con un navegador web:

---

```
[me@linuxbox ~]$ sys_info_page > sys_info_page.html  
[me@linuxbox ~]$ Firefox sys_info_page.html
```

---

Hasta ahora, bien.

Al escribir programas, siempre es una buena idea esforzarse por la simplicidad y la claridad. El mantenimiento es más fácil cuando un programa es fácil de leer y entender, sin mencionar que el programa es más

fácil de escribir cuando reducimos la cantidad de escritura. Nuestra versión actual del programa funciona bien, pero podría ser más simple. Podríamos combinar todos los comandos de eco en uno, lo cual

sin duda, facilitaría la adición de más líneas a la salida del programa. Entonces, cambiemos nuestro programa a esto:

---

```
#!/bin/bash

# Programa para generar una página de información del sistema

echo "<HTML>
    <CABEZA>
        <TITLE>Título de la
    </CABEZ
    A>
        <CUERP  página</TITLE> Cuerpo de la
    O>
</HTML>" 
    </CUERP  página.
    O>
```

---

Una cadena entrecomillada puede incluir saltos de línea y, por lo tanto, contener varias líneas de texto. La concha seguirá leyendo el texto hasta que encuentre las comillas de cierre. También funciona de esta manera en la línea de comandos:

```
[me@linuxbox ~]$ echo "<HTML>
>     <CABEZA>
>         <TÍTULO>Título de la página</TÍTULO>
>     </CABEZA>
>     <CUERPO>
>         Cuerpo de la página.
>     </CUERPO>
> </HTML>"
```

---

El carácter > inicial es el símbolo del sistema de shell contenido en la variable de shell de PS2. Aparece cada vez que escribimos una declaración de varias líneas en el shell. Esta característica es un poco oscura en este momento, pero más adelante, cuando cubramos las declaraciones de programación multilínea, resultará bastante útil.

## Segunda etapa: Agregar un poco de datos

Ahora que nuestro programa puede generar un documento mínimo, vamos a poner algunos datos en el informe. Para ello, realizaremos los siguientes cambios:

---

```
#!/bin/bash

# Programa para generar una página de información

del sistema echo "<HTML>
    <CABEZA>
        O>
    </CABEZ
    A>
        </CUERPO>
    </HTML>" 
    <CUERP
```

<TÍTULO>Informe de información del sistema</TÍTULO>

<H1>Informe de información del sistema </H1>

---

Agregamos un título de página y un encabezado al cuerpo del informe.

# Variables y constantes

Sin embargo, hay un problema con nuestro script. ¿Observa cómo se repite la cadena Informe de información del sistema? Con nuestro pequeño script no es un problema, pero imaginemos que nuestro script era muy largo y teníamos varias instancias de esta cadena. Si quisieramos cambiar el título a otra cosa, tendríamos que cambiarlo en varios lugares, lo que podría ser mucho trabajo. ¿Qué pasaría si pudiéramos organizar el script de modo que la cadena apareciera solo una vez y no varias veces? Eso haría que el mantenimiento futuro del script fuera mucho más fácil. Así es como podríamos hacerlo:

---

```
#!/bin/bash

# Programa para generar una página de información del sistema

title="Informe de información del sistema"

echo "<HTML>
    <CABEZA>
        <TÍTULO>$title</TÍTULO>
    </CABEZ
    A>
    <CUERP <H1>$title</H1>
    O>

    </CUERP
    O>
</HTML>"
```

---

Al crear una variable llamada title y asignarle el valor Informe de información del sistema, podemos aprovechar la expansión de parámetros y colocar la cadena en varias ubicaciones.

## *Creación de variables y constantes*

Entonces, ¿cómo creamos una variable? Simple, simplemente lo usamos. Cuando el shell encuentra una variable, la crea automáticamente. Esto difiere de muchos lenguajes de programación en los que las variables deben declararse o definirse explícitamente antes de su uso. El caparazón es muy laxo al respecto, lo que puede provocar algunos problemas. Por ejemplo, considere este escenario que se desarrolla en la línea de comandos:

---

```
[me@linuxbox ~]$ foo="sí"
[me@linuxbox ~]$ echo $foo
sí
[me@linuxbox ~]$ echo $fool
[me@linuxbox ~]$
```

---

Primero asignamos el valor yes a la variable foo y luego mostramos su valor con echo. A continuación, mostramos el valor del nombre de la variable mal escrito como fool y obtenemos un resultado en blanco. Esto se debe a que el shell creó felizmente la variable fool cuando la encontró y luego le dio el valor predeterminado de nothing,

o vacío. De esto aprendemos que debemos prestar mucha atención a nuestra ortografía. También es importante entender lo que realmente sucedió en este ejemplo. A partir de nuestro vistazo anterior a cómo el shell realiza expansiones, sabemos que el comando

---

```
[me@linuxbox ~]$ echo $foo
```

---

se somete a la expansión de parámetros y da como resultado

---

```
[me@linuxbox ~]$ echo sí
```

---

Por otro lado, el comando

---

```
[me@linuxbox ~]$ echo $fool
```

---

se expande a

---

```
[me@linuxbox ~]$ echo
```

---

¡La variable vacía se expande a la nada! Esto puede causar estragos con los comandos que requieren argumentos. He aquí un ejemplo:

---

```
[me@linuxbox ~]$ foo=foo.txt  
[me@linuxbox ~]$ foo1=foo1.txt  
[me@linuxbox ~]$ cp $foo $foo1  
cp: falta el operando del archivo de destino después  
de 'foo.txt' Pruebe 'cp --help' para obtener más  
información.
```

---

Asignamos valores a dos variables, `foo` y `foo1`. A continuación, realizamos un `cp` pero escribimos mal el nombre del segundo argumento. Después de la expansión, al comando `cp` se le envía solo un argumento, aunque requiere dos.

Hay algunas reglas sobre los nombres de las variables:

- Los nombres de las variables pueden constar de caracteres alfanuméricos (letras y números) y caracteres de subrayado.
- El primer carácter del nombre de una variable debe ser una letra o un guión bajo.
- No se permiten espacios ni signos de puntuación.

La palabra *variable* implica un valor que cambia, y en muchas aplicaciones, las variables se usan de esta manera. Sin embargo, la variable de nuestra aplicación, `title`, se utiliza como una *constante*. Una constante es como una variable en el sentido de que tiene un nombre y contiene un valor. La diferencia es que el valor de una constante no cambia. En una aplicación que realiza cálculos geométricos, podríamos definir `PI` como una constante y asignarle el valor de 3,1415, en su lugar de usar el número literalmente a lo largo de nuestro programa. El shell no hace distinción entre variables y constantes; Estos términos son principalmente para el

conveniencia del programador. Una convención común es usar letras mayúsculas para designar constantes y letras minúsculas para variables verdaderas. Modificaremos nuestro script para cumplir con esta convención:

---

```
#!/bin/bash

# Programa para generar una página de información del sistema

TITLE="Informe de información del sistema para

$HOSTNAME" echo "<HTML>
<CABEZA>
    <TÍTULO>$TITLE</TÍTULO>
</CABEZ
A>
<CUERP <H1>$TITLE</H1>
O>

</CUERP
O>
</HTML>"
```

---

También aprovechamos la oportunidad para darle vida a nuestro título agregando el valor de la variable de shell `HOSTNAME`. Este es el nombre de red de la máquina.

**Nota:** *El shell en realidad proporciona una forma de hacer cumplir la inmutabilidad de las constantes, mediante el uso del comando incorporado declare con la opción -r (solo lectura). Si hubiéramos asignado TITLE de esta manera:*

```
declare -r TITLE="Título de la página"
```

*el shell impediría cualquier asignación posterior a TITLE. Esta característica rara vez se usa, pero existe para scripts muy formales.*

### **Asignación de valores a variables y constantes**

Aquí es donde nuestro conocimiento de la expansión realmente comienza a dar sus frutos. Como hemos visto, a las variables se les asignan valores de esta manera:

```
variable=valor
```

donde `variable` es el nombre de la variable y `value` es una cadena. A diferencia de otros lenguajes de programación, el shell no se preocupa por el tipo de datos asignados a una variable; los trata a todos como cadenas. Puede forzar al shell a restringir la asignación a números enteros mediante el comando `declare` con la opción `-i`, pero, al igual que establecer variables como de solo lectura, esto rara vez se hace.

Tenga en cuenta que en una asignación, no debe haber espacios entre el nombre variable, el signo igual y el valor. Entonces, ¿en qué puede consistir el valor? Cualquier cosa que podamos expandir en una cadena.

```
a=z          # Asigne la cadena "z" a la variable a.  
b="una cadena"      # Los espacios incrustados deben estar entre  
comillas. c="una cadena y $b"    # Otras expansiones, como las  
variables, pueden ser  
-l foo.txt        # expandido en la tarea. d=$(ls  
# Resultados de un comando.
```

---

```
e=$((5 * 7))          # Expansión aritmética.  
f="\t\ta cadena\n"      # Secuencias de escape como tabulaciones y saltos de línea.
```

---

Se pueden realizar varias asignaciones de variables en una sola línea:

---

```
a=5 b="una cadena"
```

---

Durante la expansión, los nombres de las variables pueden estar rodeados por llaves opcionales {}. Esto es útil en los casos en los que el nombre de una variable se vuelve ambiguo debido al contexto que la rodea. Aquí, intentamos cambiar el nombre de un archivo de *myfile* a *myfile1*, usando una variable:

---

```
[me@linuxbox ~]$ filename="miarchivo"  
[me@linuxbox ~]$ toque $filename  
[me@linuxbox ~]$ mv $filename $filename 1  
mv: falta el operando de archivo de destino  
después de 'myfile' Pruebe 'mv --help' para obtener  
más información.
```

---

Este intento falla porque el shell interpreta el segundo argumento del comando mv como una variable nueva (y vacía). El problema se puede superar de la siguiente manera:

---

```
[me@linuxbox ~]$ mv $filename ${nombre de archivo}1
```

---

Al agregar las llaves circundantes, nos aseguramos de que el shell ya no interprete el 1 final como parte del nombre de la variable.

Aprovecharemos esta oportunidad para agregar algunos datos a nuestro informe, a saber, la fecha y la hora en que se creó el informe y el nombre de usuario del creador:

---

```
#!/bin/bash

# Programa para generar una página de información del
sistema TITLE="Informe de información del sistema para
$HOSTNAME"
CURRENT_TIME=$(fecha +"%x %r %Z")
TIME_STAMP="Generado $CURRENT_TIME, por $USER"

echo "<HTML>
    <CABEZ
        A>      <TÍTULO>$TITLE</TÍTULO>
    </CABEZ
        A>      <H1>$TITLE</H1>
        <CUERP  <P>$TIME_SELLO</P>
        O>

    </CUERP
    O>
</HTML>"
```

---

## Aquí Documentos

Hemos visto dos métodos diferentes para generar nuestro texto, ambos usando el comando echo. Hay una tercera forma llamada *aquí documento* o *aquí script*. Un documento here es una forma adicional de redirección de E/S en la que incrustamos

un cuerpo de texto en nuestro script y alimentarlo en la entrada estándar de un comando. Funciona de la siguiente manera:

Token de << de comandos  
Mensaje de texto  
señal

donde *comando* es el nombre de un comando que acepta entrada estándar y *token* es una cadena que se usa para indicar el final del texto incrustado. Modificaremos nuestro script para usar un documento aquí:

---

```
#!/bin/bash

# Programa para generar una página de información del sistema

TITLE="Informe de información del sistema para $HOSTNAME"
CURRENT_TIME=$(date +"%x %r %Z")
TIME_STAMP="Generado $CURRENT_TIME, por $USER"

gato << _EOF_
<HTML>
    <CABEZ
        A>      <TÍTULO>$TITLE</TÍTULO>

    </CABEZ
        A>      <H1>$TITLE</H1>
        <CUERP  <P>$TIME_SEULLO</P>
        O>
    </HTML>
_EF_
    </CUERP
    O>
```

---

En lugar de usar echo, nuestro script ahora usa cat y un documento here. La cadena \_EOF\_ (que significa *fin de archivo*, una convención común) se seleccionó como token y marca el final del texto incrustado. Tenga en cuenta que el token debe aparecer solo y que no debe haber espacios finales en la línea.

Entonces, ¿cuál es la ventaja de usar un documento aquí? Es casi lo mismo que echo, excepto que, de forma predeterminada, las comillas simples y dobles dentro de los documentos aquí pierden su significado especial para el shell. A continuación, se muestra un ejemplo de línea de comandos:

---

```
[me@linuxbox ~]$ foo="algo de texto"
[me@linuxbox ~]$ gato << _EOF_
> $foo
> "$foo"
> '$foo'
> \$foo
> _EOF_
algún texto
"algún
texto"
'algún
texto'
$foo
```

---

Como podemos ver, el shell no presta atención a las comillas. Los trata

como personajes ordinarios. Esto nos permite incrustar citas libremente dentro de un documento aquí. Esto podría resultar útil para nuestro programa de informes.

Aquí, los documentos se pueden usar con cualquier comando que acepte entrada estándar. En este ejemplo, usamos un documento here para pasar una serie de comandos al programa `ftp` con el fin de recuperar un archivo de un servidor FTP remoto:

---

```
#!/bin/bash

# Script para recuperar un fichero vía FTP

FTP_SERVER=ftp.nl.debian.org
FTP_PATH=/debian/dists/lenny/main/installer-i386/current/images/cdrom
REMOTE_FILE=debian-cd_info.tar.gz

ftp -n << _EOF_
abrir $FTP_SERVER
usuario anónimo me@linuxbox
cd $FTP_PATH
picadillo
obtener
$REMOTE_FILE
adiós
_EF_
ls -l $REMOTE_FICHERO
```

---

Si cambiamos el operador de redirección de `<<` a `<<-`, el shell ignorará los caracteres de tabulación iniciales en el documento aquí. Esto permite sangrar un documento aquí, lo que puede mejorar la legibilidad:

---

```
#!/bin/bash

# Script para recuperar un fichero vía FTP

FTP_SERVER=ftp.nl.debian.org
FTP_PATH=/debian/dists/lenny/main/installer-i386/current/images/cdrom
REMOTE_FILE=debian-cd_info.tar.gz

ftp -n <<- _EOF_
    abrir $FTP_SERVER
    usuario anónimo me@linuxbox
    cd $FTP_PATH
    picadillo
    obtener
    $REMOTE_FILE
    adiós
_EF_

ls -l $REMOTE_FICHERO
```

---

## Nota final

En este capítulo, comenzamos un proyecto que nos llevará a través del proceso de construcción de un guión exitoso. Introdujimos el concepto de variables y constituyentes y cómo pueden emplearse. Son la primera de muchas aplicaciones que encontraremos para la expansión de parámetros. También analizamos cómo producir salidas a partir de nuestro script y varios métodos para incrustar bloques de texto.



# 26

## DISEÑO TOP-DOWN

A medida que los programas se vuelven más grandes y complejos, se vuelven más difíciles de diseñar, codificar y mantener. Al igual que con cualquier proyecto grande, a menudo es una buena idea dividir las tareas grandes y complejas en una serie de tareas pequeñas y simples.

Imaginemos que estamos tratando de describir una tarea común y cotidiana: ir al mercado a comprar comida a una persona de Marte. Podríamos describir el proceso general como la siguiente serie de pasos:

1. Súbete al coche.
2. Conduzca al mercado.
3. Aparca el coche.
4. Entrar en el mercado.
5. Compra comida.
6. Regreso al coche.
7. Conduzca a casa.
8. Aparca el coche.
9. Entra en casa.



Sin embargo, es probable que una persona de Marte necesite más detalles. Podríamos dividir aún más la subtarea "Aparcar coche" en otra serie de pasos.

1. Encuentre espacio para estacionar.
2. Conduce el coche al espacio.
3. Apague el motor.
4. Ponga el freno de mano.
5. Salga del coche.
6. Cerradura del coche.

La subtarea "Apagar el motor" podría dividirse en pasos que incluyen "Apagar el encendido", "Quitar la llave de encendido", etc., hasta que cada paso de todo el proceso de ir al mercado se haya definido por completo.

Este proceso de identificación de los pasos de nivel superior y el desarrollo de vistas cada vez más detalladas de esos pasos se denomina *diseño de arriba hacia abajo*. Esta técnica nos permite dividir tareas grandes y complejas en muchas tareas pequeñas y simples. El diseño de arriba hacia abajo es un método común de diseño de programas y uno que se adapta bien a la programación de shell en particular.

En este capítulo, utilizaremos el diseño de arriba hacia abajo para desarrollar aún más nuestro script generador de informes.

## Funciones de shell

Nuestro script actualmente realiza los siguientes pasos para generar el documento HTML:

1. Página abierta.
2. Abra el encabezado de la página.
3. Establezca el título de la página.
4. Cerrar encabezado de página.
5. Abra el cuerpo de la página.
6. Encabezado de la página de salida.
7. Marca de tiempo de salida.
8. Cierre el cuerpo de la página.
9. Cerrar página.

Para nuestra siguiente etapa de desarrollo, agregaremos algunas tareas entre los pasos 7 y 8. Entre ellas se encuentran:

- **Tiempo de actividad y carga del sistema.** Esta es la cantidad de tiempo transcurrido desde el último apagado o reinicio y el número promedio de tareas que se están ejecutando actualmente en el procesador durante varios intervalos de tiempo.
- **Espacio en disco.** El uso general del espacio en los dispositivos de almacenamiento del sistema.
- **Espacio en el hogar.** La cantidad de espacio de almacenamiento que utiliza

cada usuario.

Si tuviéramos un comando para cada una de estas tareas, podríamos agregarlas a nuestro script simplemente a través de la sustitución de comandos:

---

```
#!/bin/bash

# Programa para generar una página de información del
sistema TITLE="Informe de información del sistema para
$HOSTNAME"
```

```

CURRENT_TIME=$(fecha +"%x %r %Z")
TIME_STAMP="Generado $CURRENT_TIME, por $USER"

gato << _EOF_
<HTML>
    <CABEZ
        A>      <TÍTULO>$TITLE</TÍTULO>

    </CABEZ
        A>      <H1>$TITLE</H1>
        <CUERP  $TIME_SELLO</P>
        O>      $(report_uptime)
                $(report_disk_space)
                $(report_home_space)

    </HTML>
_EF_
</CUERP
O>

```

---

Podríamos crear estos comandos adicionales de dos maneras. Podríamos escribir tres scripts separados y colocarlos en un directorio listado en nuestro PATH, o podríamos incrustar los scripts dentro de nuestro programa como funciones de shell. Como hemos mencionado antes, las funciones de shell son "miniscripts" que se encuentran dentro de otros scripts y pueden actuar como programas autónomos. Las funciones de shell tienen dos formas sintácticas. El primero se ve así:

```

nombre de la función {
    Comandos
    devolución
}

```

donde *name* es el nombre de la función y *commands* es una serie de comandos contenidos dentro de la función. El segundo se ve así:

```

nombre () {
    Comandos
    devolución
}

```

Ambas formas son equivalentes y se pueden usar indistintamente. A continuación vemos un script que demuestra el uso de una función de shell:

---

```

1  #!/bin/bash
2
3  # Demostración de la
función de shell 4
5  function funct {
6      echo "Paso 2"
7      devolución
8  }
9
10 # El programa principal
comienza aquí 11
12 echo "Paso 1"
13 funct
14 echo "Paso 3"

```

---

A medida que el shell lee el script, pasa por las líneas 1 a 11, ya que

esas líneas constan de comentarios y la definición de la función. La ejecución comienza a las

línea 12, con un comando echo. La línea 13 llama a la función de shell funct, y el shell ejecuta la función como lo haría con cualquier otro comando. A continuación, el control del programa pasa a la línea 6 y se ejecuta el segundo comando echo. A continuación, se ejecuta la línea 7. Su comando return termina la función y devuelve el control al programa en la línea que sigue a la llamada a la función (línea 14), y se ejecuta el comando echo final. Tenga en cuenta que para que las llamadas a funciones se reconozcan como funciones de shell y no se interpreten como los nombres de programas externos, las definiciones de funciones de shell deben aparecer en el script antes de que se llamen.

Agregaremos definiciones mínimas de funciones de shell a nuestro script:

---

```
#!/bin/bash

# Programa para generar una página de información del sistema

TITLE="Informe de información del sistema para $HOSTNAME"
CURRENT_TIME=$(date +"%x %r %Z")
TIME_STAMP="Generado $CURRENT_TIME, por $USER"

report_uptime () {
    devolución
}

report_disk_space () {
    return
}

report_home_space () {
    return
}

gato << _EOF_
<HTML>
    <CABEZ
        A>      <TÍTULO>$TITLE</TÍTULO>

    </CABEZ
        A>      <H1>$TITLE</H1>
        <CUERP
            O>      $(report_uptime)
                    $(report_disk_space)
                    $(report_home_space)

    </CUERP
    O>
</HTML>
_EF_
```

---

Los nombres de las funciones de shell siguen las mismas reglas que las variables. Una función debe contener al menos un comando. El comando return (que es opcional) satisface el requisito.

## Variables locales

En los scripts que hemos escrito hasta ahora, todas las variables (incluidas las constantes) han sido *variables globales*. Las variables globales mantienen su

existencia a lo largo de todo el programa. Esto está bien para muchas cosas, pero a veces puede complicarse

el uso de funciones de shell. Dentro de las funciones de shell, a menudo es deseable tener

*variables locales*. Las variables locales solo son accesibles dentro de la función de shell en la que están definidas y dejan de existir una vez que finaliza la función de shell.

Tener variables locales permite al programador usar variables con nombres que pueden ya existir, ya sea en el script globalmente o en otras funciones de shell, sin tener que preocuparse por posibles conflictos de nombres.

A continuación, se muestra un script de ejemplo que muestra cómo se definen y utilizan las variables locales:

---

```
#!/bin/bash

# local-vars: script para demostrar las variables

locales foo=0 # variable global foo

funct_1 () {

    Foo local      # variable foo local a funct_1

    foo=1
    echo "funct_1: foo = $foo"
}

funct_2 () {

    Foo local      # variable foo local a funct_2

    foo=2
    echo "funct_2: foo = $foo"
}

echo "global: foo = $foo"
funct_1
echo "global: foo = $foo"
funct_2
echo "global: foo = $foo"
```

---

Como podemos ver, las variables locales se definen precediendo el nombre de la variable con la palabra `local`. Esto crea una variable que es local a la función de shell en la que se define. Una vez que el script está fuera de la función de shell, la variable ya no existe. Cuando ejecutamos este script, vemos los resultados:

---

```
[me@linuxbox ~]$ local-vars
global:foo=0
funct_1:foo=1
global:foo=0
funct_2:foo=2
global:foo=0
```

---

Vemos que la asignación de valores a la variable local `foo` dentro de ambas funciones de shell no tiene ningún efecto sobre el valor de `foo` definido fuera de las funciones.

Esta característica permite que las funciones de shell se escriban de manera

que permanezcan independientes entre sí y del script en el que aparecen. Esto es

Muy valioso, ya que ayuda a evitar que una parte de un programa interfiera con otra. También permite que las funciones de shell se escriban para que puedan ser portátiles. Es decir, se pueden cortar y pegar de un guión a otro, según sea necesario.

## Mantener los scripts en ejecución

Al desarrollar nuestro programa, es útil mantenerlo en un estado ejecutable. Al hacer esto, y realizar pruebas con frecuencia, podemos detectar errores en las primeras etapas del proceso de desarrollo. Esto hará que los problemas de depuración sean mucho más fáciles. Por ejemplo, si ejecutamos el programa, hacemos un pequeño cambio, volvemos a ejecutar el programa y encontramos un problema, es muy probable que el cambio más reciente sea el origen del problema. Al agregar funciones vacías, llamadas *stubs* en la jerga del programador, podemos verificar el flujo lógico de nuestro programa en una etapa temprana.

Al construir un código auxiliar, es una buena idea incluir algo que proporcione retroalimentación al programador que muestre que se está llevando a cabo el flujo lógico. Si miramos la salida de nuestro script ahora, vemos que hay algunas líneas en blanco en nuestra salida después de la marca de tiempo, pero no podemos estar seguros de la causa.

---

```
[me@linuxbox ~]$ sys_info_page
<HTML>
    <CABEZ
        A>      <TITLE>Informe de información del sistema para twin2</TITLE>
    </CABEZ
        A>      <H1>Informe de información del sistema para linuxbox</H1>
        <CUERPO> <P>Generado el 19/03/2012 04:02:10 PM EDT, por mí</P>
        O>

    </CUERPO>
</HTML>
```

---

Podemos cambiar las funciones para incluir algunos comentarios:

---

```
report_uptime () {
    echo "Función report_uptime ejecutada."
    devolución
}

report_disk_space () {
    echo "Función report_disk_space ejecutada."
    devolución
}

report_home_space () {
    echo "Función report_home_space ejecutada."
    devolución
}
```

---

Y luego volvemos a ejecutar el script:

---

```
[me@linuxbox ~]$ sys_info_page
<HTML>
  <CABEZ
    A>      <TITLE>Informe de información del sistema para linuxbox</TITLE>
  </CABEZ
    A>      <H1>Informe de información del sistema para linuxbox</H1>
    <CUERP <P>Generado el 20/03/2012 05:17:26 AM EDT, por mí</P>
    O>      Función report_uptime ejecutada.
            La función report_disk_space ejecutada. La
            función report_home_space ejecutada.

  </HTML>

  </CUERP
  O>
```

---

Ahora vemos que, de hecho, nuestras tres funciones se están ejecutando.

Con nuestro marco de funciones en su lugar y funcionando, es hora de desarrollar parte del código de funciones. En primer lugar, la función `report_uptime` :

---

```
report_uptime () {
  gato <- _EOF_
    <H2>Tiempo de actividad del sistema </H2>
    <PRE>$uptime</PRE>
    EOF_
  devolución
}
```

---

Es bastante sencillo. Usamos un documento aquí para generar una sección y la salida del comando `uptime`, entre `<PRE>` para conservar el formato del comando. La función `report_disk_space` es similar:

---

```
report_disk_space () {
  gato <- _EOF_
    <H2>Utilización de espacio en disco</H2>
    <PRE>$df -h</PRE>
    _EF_
  devolución
}
```

---

Esta función utiliza el comando `df -h` para determinar la cantidad de disco espacio. Por último, crearemos la función `report_home_space`:

---

```
report_home_space () {
  gato <- _EOF_
    <H2>Utilización del espacio en el hogar</H2>
    <PRE>$du -sh /home/*</PRE>
    _EF_
  devolución
}
```

---

Usamos el comando du con las opciones -sh para realizar esta tarea. Esto, sin embargo, no es una solución completa al problema. Si bien funcionará en algunos sistemas (Ubuntu, por ejemplo), no funcionará en otros. La razón es que muchos sistemas establecen los permisos de los directorios de inicio para evitar que sean legibles por todo el mundo, lo cual es una medida de seguridad razonable. En estos sistemas, la función report\_home\_space, tal como está escrita, funcionará solo si nuestro script se ejecuta con privilegios de superusuario. Una mejor solución sería hacer que el script ajustara su comportamiento de acuerdo con los privilegios del usuario. Abordaremos esto en el capítulo 27.

### SHELL EN SU ARCHIVO .ARCHIVO BASHRC

Las funciones de shell son excelentes sustitutos de los alias, y en realidad son el método preferido para crear pequeños comandos para uso personal. Los alias son muy limitados en el tipo de comandos y características de shell que admiten, mientras que las funciones de shell permiten cualquier cosa que se pueda scriptear. Por ejemplo, si nos gusta la report\_disk\_space función de shell que desarrollamos para nuestro script, podríamos crear una función similar llamada ds para nuestro archivo *.bashrc*:

```
ds () {  
    echo "Utilización del espacio en disco para"  
    $HOSTNAME" df -h  
}
```

## Nota final

En este capítulo, hemos introducido un método común de diseño de programas llamado diseño de arriba hacia abajo, y hemos visto cómo se utilizan las funciones de shell para construir el refinamiento paso a paso que requiere. También hemos visto cómo se pueden usar variables locales para hacer que las funciones de shell sean independientes entre sí y del programa en el que se colocan. Esto hace posible que las funciones de shell se escriban de manera portátil y sean reutilizables al permitir que se coloquen en múltiples programas, un gran ahorro de tiempo.

# 27

## **CONTROL DE FLUJO: BIFURCACIÓN CON IF**

En el último capítulo, se nos presentó un problema. ¿Cómo podemos hacer que nuestro script generador de informes se adapte a los privilegios del usuario que ejecuta el script? La solución a este problema requerirá que encontremos una manera de "cambiar de dirección" dentro de nuestro guión, basándonos en los resultados de una prueba. En términos de programación, necesitamos que el programa se *ramifique*.

Consideremos un ejemplo simple de lógica expresada en *pseudocódigo*, una simulación de un lenguaje informático destinado al consumo humano:

X = 5

Si X = 5, entonces:

    Di "X es igual a

5". De otra manera:

    Di "X no es igual a 5".

Este es un ejemplo de una rama. Basándose en la condición "*¿X = 5?*", haz una cosa: "Di que X es igual a 5". De lo contrario, haz otra cosa: "Di 'X no es igual a 5'".

## El uso de if

Usando el shell, podemos codificar la lógica anterior de la siguiente manera:

---

```
x=5

si [ $x = 5 ]; entonces
    echo "x igual a 5".
else
    echo "x no es igual a 5".
fi
```

---

O podemos introducirlo directamente en la línea de comandos (ligeramente acortada):

---

```
[me@linuxbox ~]$ x=5
[me@linuxbox ~]$ si [ $x = 5 ]; entonces echo "es igual a 5"; else echo "no es igual a
5"; Fi
es igual a 5
[me@linuxbox ~]$ x=0
[me@linuxbox ~]$ si [ $x = 5 ]; entonces echo "es igual a 5"; else echo "no es igual a
5"; Fi
no es igual a 5
```

---

En este ejemplo, ejecutamos el comando dos veces. Una vez, con el valor de x establecido en 5, lo que da como resultado que la cadena sea igual a 5 siendo la salida, y la segunda vez con el valor de x establecido en 0, lo que da como resultado que la cadena no sea igual a 5 siendo la salida.

La instrucción if tiene la siguiente sintaxis:

```
if comandos; entonces
    Comandos
    [comandos elif; luego
        comandos...]
    [else
        comandos]
fi
```

donde *commands* es una lista de comandos. Esto es un poco confuso a primera vista. Pero antes de que podamos aclarar esto, tenemos que ver cómo el shell evalúa el éxito o el fracaso de un comando.

## Estado de salida

Los comandos (incluidos los scripts y las funciones de shell que escribimos) emiten un valor al sistema cuando terminan, llamado *estado de salida*. Este valor, que es un número entero en el intervalo de 0 a 255, indica el éxito o el fracaso de la ejecución del comando. Por convención, un valor de 0 indica éxito y

Cualquier otro valor indica un error. El shell proporciona un parámetro que podemos usar para examinar el estado de salida. Aquí lo vemos en acción:

---

```
[me@linuxbox ~]$ ls -d /usr/bin  
/usr/bin  
[me@linuxbox ~]$ echo $?  
0  
[me@linuxbox ~]$ ls -d /bin/usr  
ls: no se puede acceder a /bin/usr: No existe tal archivo o  
directorio [me@linuxbox ~]$ echo $?  
2
```

---

En este ejemplo, ejecutamos el comando `ls` dos veces. La primera vez, el comando se ejecuta correctamente. Si mostramos el valor del parámetro `$?`, vemos que es 0. Ejecutamos el comando `ls` por segunda vez, produciendo un error, y examinamos el parámetro `$?` otra vez. Esta vez contiene un 2, lo que indica que el comando encontró un error. Algunos comandos utilizan diferentes valores de estado de salida para proporcionar diagnósticos de errores, mientras que muchos comandos simplemente se cierran con un valor de 1 cuando fallan. Las páginas de manual a menudo incluyen una sección titulada "Estado de salida", que describe qué códigos se utilizan. Sin embargo, un 0 siempre indica éxito.

El shell proporciona dos comandos incorporados extremadamente simples que no hacen nada excepto terminar con un estado de salida 0 o 1. El comando `true` siempre se ejecuta con éxito y el comando `false` siempre se ejecuta sin éxito:

---

```
[me@linuxbox ~]$  
verdadero [me@linuxbox  
~]$ echo $? 0  
[me@linuxbox ~]$ false  
[me@linuxbox ~]$ echo $?  
1
```

---

Podemos usar estos comandos para ver cómo funciona la declaración `if`. Lo que realmente hace la instrucción `if` es evaluar el éxito o el fracaso de los comandos:

---

```
[me@linuxbox ~]$ si es verdadero; luego haga clic en "Es cierto"; Fi  
Es verdad.  
[me@linuxbox ~]$ si es falso; luego haga clic en "Es verdad"; Fi  
[me@linuxbox ~]$
```

---

El eco de la orden : "Es verdad". se ejecuta cuando el comando que sigue a `if` se ejecuta con éxito, y no se ejecuta cuando el comando que sigue a `if` no se ejecuta correctamente. Si sigue a una lista de comandos `if`, se evalúa el último comando de la lista:

---

```
[me@linuxbox ~]$ si es falso; verdadero; luego haga eco "Es verdadero"; Fi  
Es verdad.  
[me@linuxbox ~]$ si es verdadero; falso; luego haga clic en "Es verdad"; Fi  
[me@linuxbox ~]$
```

---

## Uso de la prueba

Con mucho, el comando que se usa con más frecuencia con `if` es `test`. El ordenador de pruebas realiza una variedad de comprobaciones y comparaciones. Tiene dos formas equivalentes:

Expresión de prueba

y los más populares

[ *expresión* ]

donde *expresión* es una expresión que se evalúa como verdadera o falsa. El comando `test` devuelve un estado de salida de 0 cuando la expresión es verdadera y un estado de 1 cuando la expresión es falsa.

### Expresiones de archivo

Las expresiones de la Tabla 27-1 se utilizan para evaluar el estado de los archivos.

Tabla 27-1: Expresiones de archivo de prueba

Expresión	Es cierto si . . .
<code>archivo1 -Ef archivo2</code>	<code>archivo1</code> y <code>Archivo2</code> tienen los mismos números de inodo (los dos Los nombres de archivo se refieren al mismo archivo mediante enlaces duros).
<code>archivo1 -Nt Archivo2</code>	<code>archivo1</code> es más reciente que <code>Archivo2</code> .
<code>Archivo2</code>	<code>archivo1</code> es mayor que <code>Archivo2</code> .
<code>-b archivo</code>	<code>archivo</code> existe y es un archivo especial de bloque (dispositivo).
<code>-c archivo</code>	<code>archivo</code> existe y es un archivo de carácter especial (dispositivo).
<code>-d archivo</code>	<code>archivo</code> existe y es un directorio.
<code>-e archivo</code>	<code>archivo</code> Existe.
<code>-f archivo</code>	<code>archivo</code> existe y es un archivo normal.
<code>-g archivo</code>	<code>archivo</code> existe y es set-group-ID.
<code>-G archivo</code>	<code>archivo</code> existe y es propiedad del ID de grupo efectivo.
<code>-k archivo</code>	<code>archivo</code> existe y tiene su "sticky bit" establecido.
<code>-L archivo</code>	<code>archivo</code> existe y es un vínculo simbólico.
<code>-O archivo</code>	<code>archivo</code> existe y es propiedad del ID de usuario efectivo.
<code>-p archivo</code>	<code>archivo</code> existe y es una tubería con nombre.
<code>-r archivo</code>	<code>archivo</code> existe y es legible (tiene permiso legible para

el usuario efectivo).

---

-s *archivo*

*archivo* existe y tiene una longitud mayor que cero.

---

Cuadro 27-1 (continuación )

Expresión	Es cierto si . . .
<code>-S archivo</code>	<code>archivo</code> existe y es un socket de red.
<code>-t fdfd</code>	es un descriptor de archivo dirigido hacia/desde el terminal. Esto se puede utilizar para determinar si se está redirigiendo la entrada/salida/error estándar.
<code>-u archivo</code>	<code>archivo</code> existe y es setuid.
<code>-w archivo</code>	<code>archivo</code> existe y se puede escribir (tiene permiso de escritura para el usuario efectivo).
<code>-x fichero</code>	existe y es ejecutable (tiene permisión para el usuario efectivo).

Aquí tenemos un script que muestra algunas de las expresiones del archivo:

```
#!/bin/bash

# test-file: Evalúa el estado de un fichero

FILE=~/bashrc

si [ -e "$FILE" ]; entonces
    if [ -f "$FILE" ]; entonces
        echo "$FILE es un archivo normal."
    Fi
    si [ -d "$FILE" ]; entonces
        echo "$FILE es un directorio."
    Fi
    si [ -r "$FILE" ]; entonces
        echo "$FILE es legible."
    Fi
    si [ -w "$FILE" ]; entonces
        echo "$FILE se puede escribir."
    Fi
    si [ -x "$FILE" ]; entonces
        echo "$FILE es ejecutable/se puede buscar."
    Fi
más
echo "$FILE no existe" salida 1

Sali
da
```

El script evalúa el archivo asignado a la constante FILE y muestra su

resultados a medida que se realiza la evaluación. Hay dos cosas interesantes a tener en cuenta sobre este script. En primer lugar, observe cómo se entrecilla el parámetro FILE dentro de las expresiones. Esto no es obligatorio, pero es una defensa contra el hecho de que el parámetro esté vacío. Si la expansión del parámetro de FILE diera como resultado un valor vacío, produciría un error (los operadores se interpretarían como cadenas no nulas en lugar de operadores). Uso de las comillas alrededor del parámetro

Garantiza que el operador siempre vaya seguido de una cadena, incluso si la cadena está vacía. En segundo lugar, observe la presencia de los comandos de salida cerca del final del script. El comando exit acepta un único argumento opcional, que se convierte en el estado de salida del script. Cuando no se pasa ningún argumento, el estado de salida se establece de forma predeterminada en 0. El uso de exit de esta manera permite que el script indique un error si \$FILE expande al nombre de un archivo inexistente. El comando de salida que aparece en la última línea del script está ahí como una formalidad. Cuando un script *se ejecuta desde el final* (llega al final del archivo), finaliza con un estado de salida de 0 de forma predeterminada, de todos modos.

De manera similar, las funciones de shell pueden devolver un estado de salida incluyendo un argumento entero en el comando return. Si tuviéramos que convertir el script anterior en una función de shell para incluirlo en un programa más grande, podríamos reemplazar los comandos de salida con declaraciones return y obtener el comportamiento deseado:

---

```
test_file () {  
    # test-file: Evalúa el estado de un fichero  
  
    FILE=~/.bashrc  
  
    si [ -e "$FILE" ]; entonces  
        if [ -f "$FILE" ]; entonces  
            echo "$FILE es un archivo normal."  
        Fi  
        si [ -d "$FILE" ]; entonces  
            echo "$FILE es un directorio."  
        Fi  
        si [ -r "$FILE" ]; entonces  
            echo "$FILE es legible."  
        Fi  
        si [ -w "$FILE" ]; entonces  
            echo "$FILE se puede escribir."  
        Fi  
        si [ -x "$FILE" ]; entonces  
            echo "$FILE es ejecutable/se puede buscar."  
        Fi  
    más  
    echo "$FILE no existe"  
    return 1  
}  
}
```

---

## Expresiones de cadena

Las expresiones de la Tabla 27-2 se utilizan para evaluar cadenas.

Tabla 27-2: Expresiones de cadena de prueba

Expresión	Es cierto si . . .
cuerda	cuerda no es nulo.

-n cuerda

La longitud de *cuerda* es mayor que cero.

---

Cuadro 27-2 (continuación )

Expresión	Es cierto si . . .
<code>-z cuerda</code>	La longitud de <i>cuerda</i> es cero.
<code>cadena1 = cadena2</code> <code>== cadena2</code>	<i>String1</i> y <i>string2</i> son iguales. Se pueden usar signos iguales simples o dobles, pero se prefiere el uso de signos iguales dobles.
<code>cadena1 != cadena2</code>	<i>cadena1</i> y <i>cadena2</i> no son
<code>iguales. cadena1 &gt; cadena2</code>	<i>cadena1</i> Ordena después
<code>cadena2. cadena1 &lt; cadena2</code>	<i>cadena1</i> Ordena antes
<code>cadena2.</code>	<i>cadena2</i> .

**Advertencia:** Los operadores de expresión `>` y `<` deben estar entre comillas (o con escape con una barra diagonal inversa) cuando se usan con `test`. Si no lo son, serán interpretados por el shell como operadores de redirección, con resultados potencialmente destructivos. También tenga en cuenta que, aunque la documentación de bash indica que el orden de clasificación se ajusta al orden de intercalación de la configuración regional actual, no es así. El orden ASCII (POSIX) se utiliza en versiones de bash hasta la 4.0 inclusive.

A continuación, se muestra un script que incorpora expresiones de cadena:

```
#!/bin/bash

# test-string: evalúa el valor de una cadena

ANSWER=maybe

si [ -z "$ANSWER" ]; entonces
    echo "No hay respuesta." >&2
    salida 1
fi

if [ "$ANSWER" = "sí" ]; luego
    haga eco "La respuesta es
    SI".
elif [ "$ANSWER" = "no" ]; luego
    haga eco "La respuesta es
    NO".
elif [ "$ANSWER" = "tal vez" ]; luego
    haga eco "La respuesta es TAL
    VEZ".
else
    echo "La respuesta es DESCONOCIDA".
fi
```

En este script, evaluamos la constante ANSWER. Primero determinamos si el La cadena está vacía. Si es así, terminamos el script y establecemos el estado de salida en 1. Observe la redirección que se aplica al comando echo. Esto redirige el mensaje de error "No hay respuesta" al error estándar, que es lo "correcto" que se debe hacer con los mensajes de error.

Si la cadena no está vacía, evaluamos el valor de la cadena para ver si es igual a "sí", "no" o "quizás". Hacemos esto usando `elif`, que es la abreviatura de `else if`. Al usar `elif`, podemos construir una prueba lógica más compleja.

## Expresiones enteras

Las expresiones de la Tabla 27-3 se utilizan con números enteros.

Tabla 27-3: Prueba de expresiones enteras

Expresión	Es cierto si . . .
<code>entero1 -Eq entero2</code>	<code>entero1</code> es igual a <code>entero2</code> .
<code>entero1 -Ne entero2</code>	<code>entero1</code> no es igual a <code>entero2</code> .
<code>entero1 -le entero2</code>	<code>entero1</code> es menor o igual que <code>entero2</code> . <code>entero1 -lt entero2</code>
<code>entero1 -Ge entero2</code>	<code>entero1</code> es mayor o igual que <code>entero2</code> .
<code>entero1 -Gt entero2</code>	<code>entero1</code> es mayor que <code>entero2</code> .

Aquí hay un script que los demuestra:

```
#!/bin/bash

# test-integer: evalúa el valor de un entero.

INT=-5

si [ -z "$INT" ]; entonces
    echo "INT está vacío." >&2
    salida 1
fi

si [ $INT -eq 0 ]; entonces
    echo "INT es cero".
más
    si [ $INT -lt 0 ]; entonces
        echo "INT es negativo".
    más
        echo "INT es positivo".
fi
if [ $((INT % 2)) -eq 0 ];
    luego haga eco de
    "INT es parejo".
más
    echo "INT es raro".
fi
fi
```

La parte interesante del script es cómo determina si un número entero es par o impar. Al realizar una operación de módulo 2 en el número, que divide el número por 2 y devuelve el resto, puede decir si el número es par o impar.

## Una versión más moderna de la prueba

Las versiones recientes de bash incluyen un comando compuesto que actúa como un reemplazo mejorado para test. Utiliza la siguiente sintaxis:

`[[ expresión ]]`

donde *expression* es una expresión que se evalúa como un resultado verdadero o falso. El comando `[[ ]]` es muy similar a test (admite todas sus expresiones) pero agrega una nueva expresión de cadena importante:

`cadena1 =~ expresión regular`

que devuelve true si String1 coincide con la expresión regular exGive extendida. Esto abre muchas posibilidades para realizar tareas como la validación de datos. En nuestro ejemplo anterior de las expresiones enteras, el script fallaría si la constante INT contenía algo excepto un entero. El script necesita una forma de comprobar que la constante contiene un número entero. Usando `[[ ]]` con el operador de expresión de cadena `=~`, podríamos mejorar el script de esta manera:

---

```
#!/bin/bash

# test-integer2: evalúa el valor de un entero.

INT=-5

if [[ "$INT" =~ ^-?[ 0-9]+$ ]]; entonces
    si [ $INT -eq 0 ]; entonces
        echo "INT es cero".
    más
        si [ $INT -lt 0 ]; entonces
            echo "INT es negativo".
        más
            echo "INT es positivo".
    Fi
    if [ $((INT % 2)) -eq 0 ]; luego
        haga eco de "INT es
        parejo".
    else
        echo "INT es raro".
    fi
F
i
más
    echo "INT no es un número entero."
>&2 salida 1
Fi
```

---

Al aplicar la expresión regular, podemos limitar el valor de INT a Solo cadenas que comiencen con un signo menos opcional, seguido de uno o más números. Esta expresión también elimina la posibilidad de valores vacíos.

Otra característica añadida de [[ ]] es que el operador == admite la coincidencia de patrones de la misma manera que lo hace la expansión de nombre de ruta. Por ejemplo:

```
[me@linuxbox ~]$ ARCHIVO=foo.bar
[me@linuxbox ~]$ if [[ $FILE == foo.* ]]; then
> echo "$FILE coincide con el patrón 'foo.*'"
> Fi
foo.bar coincide con el patrón 'foo.*'
```

Esto hace que [[ ]] sea útil para evaluar nombres de archivos y rutas.

## (( )): diseñado para números enteros

Además del comando compuesto [[ ]], bash también proporciona el comando compuesto (( )), que es útil para operar con números enteros. Es compatible con un conjunto completo de evaluaciones aritméticas, un tema que trataremos a fondo en el capítulo 34.

(( )) se utiliza para realizar *pruebas de verdad aritmética*. Una prueba de verdad aritmética da como resultado verdadero si el resultado de la evaluación aritmética es distinto de cero.

```
[me@linuxbox ~]$ si ((1)); luego echo "Es verdad."; Fi
Es verdad.
[me@linuxbox ~]$ si ((0)); entonces echo "Es verdad."; Fi
[me@linuxbox ~]$
```

Usando (( )), podemos simplificar ligeramente el script test-integer2 de la siguiente manera:

```
#!/bin/bash

# test-integer2a: evalúa el valor de un entero.

INT=-5

if [[ "$INT" =~ ^-?[ 0-9]+$ ]];
entonces si ((INT == 0));
entonces
    echo "INT es cero".
más
    if ((INT < 0)); entonces
        echo "INT es negativo".
más
        echo "INT es positivo".
    Fi
    si (((INT % 2) == 0)); entonces
        echo "INT está parejo".
    else
        echo "INT es raro".
    fi
F
i
más
    echo "INT no es un número
entero." >&2 salida 1
Fi
```

Nótese que usamos signos de menor y mayor que y que == se usa para probar la equivalencia. Se trata de una sintaxis de aspecto más natural para trabajar con números enteros. Nótese también que debido a que el comando compuesto (( )) es parte de la sintaxis del shell en lugar de un comando ordinario, y trata solo con números enteros, es capaz de reconocer variables por su nombre y no requiere que se realice una expansión.

## Combinación de expresiones

También es posible combinar expresiones para crear evaluaciones más complejas. Las expresiones se combinan mediante operadores lógicos. Los vimos en el capítulo 17, cuando aprendimos sobre el comando find. Hay tres operaciones lógicas para test y [[ ]]. Son Y, O y NO. test y [[ ]] utilizan diferentes operadores para representar estas operaciones, como se muestra en la Tabla 27-4.

**Tabla 27-4: Operadores lógicos**

Operación	prueba	[[ ]] y (( ))
Y	-un	&&
O	-o	
NO	!	!

Este es un ejemplo de una operación AND. El siguiente script determina si un número entero está dentro de un rango de valores:

---

```
#!/bin/bash

# test-integer3: determina si un entero está
dentro de un # rango de valores especificado.

MIN_VAL=1
MAX_VAL=100

INT=50

if [[ "$INT" =~ ^-[0-9]+\$ ]]; entonces
    if [[ INT -ge MIN_VAL && INT -le MAX_VAL ]]; luego
        echo "$INT está dentro de $MIN_VAL a
$MAX_VAL".
    else
        echo "$INT está fuera de alcance".
    fi
más
    echo "INT no es un número entero."
>&2 salida 1
fi
```

---

En este script, determinamos si el valor del entero INT se encuentra entre los valores de MIN\_VAL y MAX\_VAL. Esto se realiza mediante un solo uso de [[ ]], que incluye dos expresiones separadas por el operador &&. También podríamos haber codificado esto usando test:

---

```
if [ $INT -ge $MIN_VAL -a $INT -le $MAX_VAL ]; luego
    echo "$INT está dentro de $MIN_VAL a
$MAX_VAL."
else
    echo "$INT está fuera de alcance".
fi
```

---

El ! negation invierte el resultado de una expresión. Devuelve true si una expresión es falsa y devuelve false si una expresión es verdadera. En el siguiente script, modificamos la lógica de nuestra evaluación para encontrar valores de INT que estén fuera del rango especificado:

---

```
#!/bin/bash

# test-integer4: determina si un entero está fuera
de un # rango de valores especificado.

MIN_VAL=1
MAX_VAL=100

INT=50

if [[ "$INT" =~ ^-?[ 0-9]+\$ ]]; entonces
    si [[ ! (INT -ge MIN_VAL && INT -le MAX_VAL) ]];
        luego echo "$INT está fuera de $MIN_VAL a
$MAX_VAL".
    else
        echo "$INT está dentro del alcance".
    fi
más
    echo "INT no es un número entero."
>&2 salida 1
Fi
```

---

También incluimos paréntesis alrededor de la expresión para agrupar. Si De no incluirse, la negación se aplicaría sólo a la primera expresión y no a la combinación de ambas. Codificar esto con test se haría de esta manera:

---

```
if [ ! \($INT -ge $MIN_VAL -a $INT -le $MAX_VAL \) ]; luego
    echo "$INT está fuera de $MIN_VAL a $MAX_VAL".
else
    echo "$INT está dentro del alcance".
fi
```

---

Dado que todas las expresiones y operadores utilizados por test son tratados como argumentos comunes por el shell (a diferencia de [[ ]] y (( ))), los caracteres que tienen un significado especial para bash, como <, >, (, y ), deben ser citados o escapados.

Al ver que la prueba y [[ ]] hacen más o menos lo mismo, ¿cuál es preferible? test es tradicional (y parte de POSIX), mientras que [[ ]] es específico de bash. Es importante saber cómo usar test, ya que es muy utilizado, pero [[ ]] es claramente más útil y es más fácil de codificar.

### POR LA PORTABILIDAD ES EL DUEÑO DE LAS MENTES PEQUEÑAS

Si hablas con gente "real" de Unix, rápidamente descubres que a muchos de ellos no les gusta mucho Linux. Lo consideran impuro e impuro. Un principio de los seguidores de Unix es que todo debe ser *portátil*. Esto significa que cualquier script que escribas debería poder ejecutarse, sin cambios, en cualquier sistema tipo Unix.

La gente de Unix tiene buenas razones para creer esto. Habiendo visto lo que las extensiones propietarias de comandos y shells hicieron al mundo Unix antes de POSIX, naturalmente desconfían del efecto de Linux en su amado sistema operativo.

Pero la portabilidad tiene un serio inconveniente. Impide el progreso. Requiere que las cosas se hagan siempre utilizando técnicas de "mínimo común denominador". En el caso de la programación de shell, significa hacer todo compatible con sh, el shell original de Bourne.

Esta desventaja es la excusa que utilizan los proveedores propietarios para justificar sus extensiones propietarias, solo que las llaman "innovaciones". Pero en realidad son solo dispositivos de bloqueo para sus clientes.

Las herramientas GNU, como bash, no tienen tales restricciones. Fomentan la portabilidad al apoyar las normas y estar disponibles universalmente. Puedes instalar bash y las otras herramientas GNU en casi

## Operadores de control: otra forma de ramificar

bash proporciona dos operadores de control que pueden realizar bifurcaciones. El && (AND) y || Los operadores (OR) funcionan como los operadores lógicos del comando compuesto [[ ]]. Esta es la sintaxis:

*Comando1* y *Comando2*

y

*comando1* || *Comando2*

Es importante entender el comportamiento de estos. Con el operador && Ator, el *comando1* se ejecuta y el *comando2* se ejecuta si, y solo si, el *comando1* se realiza correctamente. Con el || operador, *command1* se ejecuta y *command2* se ejecuta si, y solo si, *command1* no se realiza correctamente.

En términos prácticos, significa que podemos hacer algo como esto:

---

```
[me@linuxbox ~]$ mkdir temp && cd temp
```

---

Esto creará un directorio llamado *temp* y, si tiene éxito, el directorio de trabajo actual se cambiará a *temp*. El segundo comando solo se intenta si el comando *mkdir* tiene éxito. Del mismo modo, un comando como

---

```
[me@linuxbox ~]$ [ -d temp ] || mkdir temp
```

---

probará la existencia del directorio *temp* y, solo si se produce un error en la prueba, se creará el directorio. Este tipo de construcción es muy útil para manejar errores en scripts, un tema que discutiremos más en capítulos posteriores. Por ejemplo, podríamos hacer esto en un script:

---

```
[ -d temp ] || Salida 1
```

---

Si el script requiere el directorio *temp* y no existe, el script finalizará con un estado de salida de 1.

## Nota final

Comenzamos este capítulo con una pregunta. ¿Cómo podríamos hacer que nuestro script *sys\_info\_page* detectara si el usuario tenía o no permiso para leer todos los directorios de inicio? Con nuestro conocimiento de *if*, podemos resolver el problema agregando este código a la función *report\_home\_space*:

---

```
report_home_space () {
    if [[ $(id -u) -eq 0 ]];
        entonces gato <-
        _EOF_
            <H2>Utilización del espacio en el hogar (todos los
            usuarios)</H2>
            <PRE>$(du -sh /home/*)</PRE>
            _EF_
    más
        gato <- _EOF_
            <H2>Utilización del espacio en el hogar ($USER)</H2>
            <PRE>$(du -sh $HOME)</PRE>
            _EF_
    Devo
    lució
    n de
    Fi
}
```

---

Evaluamos la salida del comando *id*. Con la opción *-u*, *id* muestra el número de ID de usuario numérico del usuario efectivo. El superusuario siempre es cero y todos los demás usuarios son un número mayor que cero. Sabiendo esto, podemos construir dos documentos diferentes, uno aprovechando los privilegios de superusuario y el otro restringido al propio directorio de inicio del usuario.

Vamos a tomarnos un descanso del programa *sys\_info\_page*, pero no te

preocupes. Volverá. Mientras tanto, cubriremos algunos temas que necesitaremos cuando reanudemos nuestro trabajo.

# 28

## LECTURA DE LA ENTRADA DEL TECLADO

Los scripts que hemos escrito hasta ahora carecen de una característica común a la mayoría de los programas informáticos: *la interactividad*, la capacidad del programa para interactuar con el usuario. Si bien muchos programas no necesitan ser interactivos, algunos programas se benefician de poder aceptar entradas directamente del usuario. Tomemos, por ejemplo, este script del capítulo anterior:

---

```
#!/bin/bash

# test-integer2: evalúa el valor de un entero.

INT=-5

if [[ "$INT" =~ ^-[0-9]+$ ]]; then
    if [ $INT -eq 0 ]; then
        echo "INT es cero".
    else
        echo "INT es menor que cero".
    fi
fi
```

```
echo "INT es negativo".
```



```

más
    echo "INT es positivo".
Fí
if [ $((INT % 2)) -eq 0 ]; luego
    haga eco de "INT es
    parejo".
else
    echo "INT es raro".
fi
F
i
más
    echo "INT no es un número
entero." >&2 salida 1
Fí

```

---

Cada vez que queramos cambiar el valor de INT, tenemos que editar el script.

El script sería mucho más útil si pudiera pedir al usuario un valor. En este capítulo, comenzaremos a ver cómo podemos agregar interactividad a nuestros programas.

## read: leer valores de la entrada estándar

El comando integrado read se utiliza para leer una sola línea de entrada estándar. Este comando se puede utilizar para leer la entrada del teclado o, cuando se emplea la redirección, una línea de datos de un archivo. El comando tiene la siguiente sintaxis:

`leer [-opciones] [variable...]`

donde *opciones* es una o más de las opciones disponibles enumeradas en la Tabla 28-1 y *variable* es el nombre de una o más variables utilizadas para contener el valor de entrada. Si no se proporciona ningún nombre de variable, la variable de shell REPLY contiene la línea de datos.

Tabla 28-1: Opciones de lectura

Opción	Descripción
-un array	Asignar la entrada a la matriz , empezando por el índice cero. Cubriremos las matrices en el Capítulo 35.
-d <i>delimitadorEl</i>	primer carácter de la cadena <i>delimitador</i> se utiliza para indicar el final de la entrada, en lugar de un carácter de nueva línea.
-e	Utilice Readline para controlar la entrada. Esto permite la edición de entradas de la misma manera que la línea de comandos.
-n <i>Num</i>	Leer <i>Num</i> de entrada, en lugar de una línea completa.
-p <i>pronto</i>	Mostrar una solicitud de entrada mediante la cadena <i>pronto</i> .
-r	Modo RAW. No interprete los caracteres de barra diagonal inversa

como escapes.

Cuadro 28-1 (continuación )

Opción	Descripción
-s	Modo silencioso. No haga eco de los caracteres en la pantalla a medida que se escriben. Esto es útil cuando se ingresan contraseñas y otra información confidencial.
-t <i>secondsTimeout</i>	Finalizar la entrada después de <i>sobras</i> . leer Devuelve un estado de salida distinto de cero si se agota el tiempo de espera de una entrada.
-u <i>Fd</i>	Usar la entrada del descriptor de archivo <i>Fd</i> , en lugar de la entrada estándar.

Básicamente, la lectura asigna campos de la entrada estándar a las variables especificadas. Si modificamos nuestro script de evaluación de enteros para usar read, podría verse así:

```
#!/bin/bash

# read-integer: evalúa el valor de un entero.

echo -n "Por favor, introduzca un
número entero -> " read int

if [[ "$int" =~ ^-[0-9]+$ ]];
    entonces si [ $int -eq 0
    ]; entonces
        eco "$int es cero".
    más
        si [ $int -lt 0 ]; entonces
            "$int es negativo".
        más
            "$int es positivo".
    Fi
    if [ $((int % 2)) -eq 0 ]; luego
        repite "$int es par".
    else
        "$int es extraño".
    fi

    Fi
else
    echo "El valor de entrada no es un número
    entero." >&2 salida 1
fi
```

Usamos echo con la opción -n (que suprime la nueva línea final en output) para mostrar un mensaje y, a continuación, utilice read para introducir un valor para el int variable. La ejecución de este script da como resultado lo siguiente:

```
[me@linuxbox ~]$ read-integer
Por favor, introduzca un
número entero -> 5 5 es
positivo.
```

5 es impar.

---

read puede asignar entrada a varias variables, como se muestra en este script:

---

```
#!/bin/bash

# read-multiple: lee varios valores del teclado echo -n

"Introduce uno o más valores > "
Leer var1 var2 var3 var4 var5

echo "var1 = '$var
1'" echo "var2 =
'$var 2'" echo "var3
= '$var 3'" echo
"var4 = '$var 4'" echo
"var5 = '$var 5"
```

---

En este script, asignamos y mostramos hasta cinco valores. Fíjate en lo leído que se lee  
se comporta cuando se le da un número diferente de valores:

---

```
[me@linuxbox ~]$ read-multiple
Introduzca uno o más valores > a b c
d e var1 = 'a'
var2 = 'b'
var3 =
'c' var4
=
'd'
var5 = 'e'
[me@linuxbox ~]$ read-multiple
Ingrese uno o más valores >
var1 = 'a'
var2 =
var3 =
var4 =
var5 =
[me@linuxbox ~]$ lectura-múltiple
Introduzca uno o más valores > a, b, c, d, e, f, g
var1 = 'a'
var2 = 'b'
var3 = 'c'
var4 = 'd'
var5 = 'e' f
g'
```

---

Si la lectura recibe menos del número esperado, las variables adicionales están vacías, mientras que una cantidad excesiva de entrada da como resultado que la variable final contenga toda la entrada adicional.

Si no se enumera ninguna variable después del comando de lectura, se asignará a una variable de shell, REPLY, todas las entradas:

---

```
#!/bin/bash

# read-single: lee varios valores en la variable
predeterminada echo -n "Ingrese uno o más valores > "
leer

echo "REPLY = '$REPLY'"
```

---

La ejecución de este script da como resultado lo siguiente:

---

```
[me@linuxbox ~]$ leer-sencillo  
Introduzca uno o más valores > a, b, c, d  
RESPONDER = 'a b c d'
```

---

## Opciones

read admite las opciones mostradas anteriormente en la Tabla 28-1.

Usando las diversas opciones, podemos hacer cosas interesantes con la lectura. Por ejemplo, con la opción -p, podemos proporcionar una cadena de solicitud:

---

```
#!/bin/bash  
  
# read-single: lee varios valores en la variable  
  
predeterminada read -p "Ingresa uno o más valores > "  
  
echo "REPLY = '$REPLY'"
```

---

Con las opciones -t y -s podemos escribir un script que lea la entrada "secreta" y se agote el tiempo de espera si la entrada no se completa en un tiempo específico:

---

```
#!/bin/bash  
  
# read-secret: introduce una frase de contraseña secreta  
  
if read -t 10 -sp "Ingrese la frase de contraseña secreta > "  
    secret_pass; then echo -e "\nContraseña secreta =  
    '$secret_pass'"  
else  
    echo -e "\nSe ha agotado el  
    tiempo de espera de la entrada"  
fi >&2 salida 1
```

---

El script solicita al usuario una frase de contraseña secreta y espera 10 segundos para la entrada. Si la entrada no se completa dentro del tiempo especificado, el script se cierra con un error. Dado que se incluye la opción -s, los caracteres de la frase de contraseña no se repiten en la pantalla a medida que se escriben.

## Separación de campos de entrada con IFS

Normalmente, el shell realiza la división de palabras en la entrada proporcionada para leer. Como hemos visto, esto significa que varias palabras separadas por uno o más espacios se convierten en elementos separados en la línea de entrada y se asignan a variables separadas por lectura. Este comportamiento se configura mediante una variable de shell denominada IFS (para Separador de campo interno). El valor predeterminado de IFS contiene un espacio, una tabulación y un carácter de

nueva línea, cada uno de los cuales separará los elementos entre sí.

Podemos ajustar el valor de IFS para controlar la separación de los campos de entrada a leer. Por ejemplo, el archivo `/etc/passwd` contiene líneas de datos que utilizan el carácter de dos puntos como separador de campos. Al cambiar el valor de IFS a dos puntos,

Podemos usar `read` para ingresar el contenido de `/etc/passwd` y separar con éxito los campos en diferentes variables. Aquí tenemos un script que hace precisamente eso:

---

```
#!/bin/bash

# read-ifs: leer campos de un archivo

FILE=/etc/passwd

read -p "Ingrese un nombre de usuario > "

user_name file_info=$(grep "^$user_name:" $FILE) ①

if [ -n "$file_info" ]; then
    IFS=: read usuario pw uid gid name home shell <<< "$file_info" ②
    echo "Usuario = '$user_name'"
    echo "UID = '$uid'"
    echo "GID = '$gid'"
    echo "Lleno Nombre = '$name'" echo "Inicio Dir. = '$home'" echo "Concha = '$shell'"
else
    echo "No existe tal usuario '$user_name'" >&2 salida 1
fi
```

---

Este script solicita al usuario que ingrese el nombre de usuario de una cuenta en el sistema y, a continuación, muestra los diferentes campos que se encuentran en el registro del usuario en el `archivo /etc/passwd`. El guión contiene dos líneas interesantes. El primero, en ①, asigna los resultados de un comando `grep` a la variable `file_info`. La expresión regular utilizada por `grep` garantiza que el nombre de usuario coincidirá solo con una sola línea en el `archivo /etc/passwd`.

La segunda línea interesante, en ②, consta de tres partes: una asignación de variables, un comando de lectura con una lista de nombres de variables como argumentos y un nuevo y extraño operador de redirección. Primero veremos la asignación de variables.

El shell permite que una o más asignaciones de variables tengan lugar inmediatamente antes de un comando. Estas asignaciones alteran el entorno para el comando que sigue. El efecto de la asignación es temporal, solo cambia el entorno mientras dure el comando. En nuestro caso,

el valor de `IFS` se cambia a un carácter de dos puntos. Alternativamente, podríamos haberlo codificado de esta manera:

---

```
OLD_IFS="$IFS"
IFS=:
leer usuario pw uid gid nombre home shell <<< "$file_info"
IFS="$OLD_IFS"
```

---

donde almacenamos el valor de `IFS`, asignamos un nuevo valor, realizamos el

comando de lectura y luego restauramos IFS a su valor original. Claramente, colocar la asignación de variables delante del comando es una forma más concisa de hacer lo mismo.

El operador <<< indica una cadena here. Una *cadena here* es como un documento here, sólo que más corto, que consiste en una sola cadena. En nuestro ejemplo, la línea de datos del *archivo /etc/passwd* se alimenta a la entrada estándar del comando *read*. Podríamos preguntarnos por qué se eligió este método más bien obvio en lugar de

```
echo "$file_info" | IFS=: read user pw uid gid name home shell
```

Bueno, hay una razón...

### TÚ NO PUEDES PIPA RE ANUNCIO

Si bien el comando *read* normalmente toma la entrada de la entrada estándar, no puede hacer lo siguiente:

```
echo "foo" | leer
```

Esperaríamos que esto funcionara, pero no lo hace. El comando parecerá correcto, pero la variable *REPLY* siempre estará vacía. ¿A qué se debe esto?

La explicación tiene que ver con la forma en que el shell maneja las tuberías. En *bash* (y otros shells como *sh*), las canalizaciones crean *subshells*. Estas son copias del shell y su entorno que se utilizan para ejecutar el comando en la línea de tubería. En nuestro ejemplo anterior, la lectura se ejecuta en un subshell.

Los subshells en sistemas tipo Unix crean copias del entorno para que los procesos las utilicen mientras se ejecutan. Cuando finalizan los procesos, se destruye la copia del entorno. Esto significa que *una subcapa nunca puede alterar el entorno de su proceso principal*. *read* asigna variables, que luego se convierten en parte del entorno. En el ejemplo anterior, *read* asigna el valor *foo* a la variable *REPLY* en el entorno de su subshell, pero cuando se cierra el comando, el subshell y su entorno se destruyen y se pierde el efecto de la asignación.

El uso de cadenas here es una forma de evitar este comportamiento. Otro método se discute en el Capítulo 36.

## Validación de la entrada

Con nuestra nueva capacidad de tener entrada de teclado viene un desafío adicional de programación: validar la entrada. Muy a menudo, la diferencia entre un programa bien escrito y uno mal escrito radica en la capacidad del programa para hacer frente a lo inesperado. Con frecuencia, lo inesperado aparece en forma de una mala entrada. Hicimos un poco de esto con nuestros programas de evaluación en el capítulo anterior, donde verificamos los valores de los números enteros y descartamos los valores vacíos y los caracteres no numéricos. Es importante realizar este tipo de comprobaciones de programación cada vez que un programa recibe una entrada para protegerse contra datos no válidos. Esto es especialmente importante para los programas que son compartidos por varios usuarios. La omisión de estas salvaguardias en interés de la economía podría ser excusada si un programa va a ser utilizado una sola vez y sólo por el autor

para

realizar alguna tarea especial. Incluso entonces, si el programa realiza tareas peligrosas como eliminar archivos, sería prudente incluir la validación de datos, por si acaso.

Aquí tenemos un programa de ejemplo que valida varios tipos de entrada:

```
#!/bin/bash

# read-validate: validar la entrada

invalid_input () {
    echo "Entrada '$REPLY'" no válida
    >&2 salida 1
}

read -p "Ingrese un solo

elemento > " # la entrada está

vacía (no válida)
[[ -z $REPLY ]] && invalid_input

# la entrada es varios elementos (no válido)
(( $(echo $REPLY | wc -w) > 1 )) && invalid_input

# ¿Es la entrada un nombre de archivo válido?
si [[ $REPLY =~ ^[-[:alnum:]\._]+$ ]]; A
    continuación, haga clic en
    "$REPLY" es un nombre de archivo
    válido". si [[ -e $REPLY ]];
    entonces
        echo "Y el archivo '$REPLY' existe".
    else
        echo "Sin embargo, el archivo '$REPLY' no existe".
    fi

# ¿La entrada es un número de coma flotante?
si [[ $REPLY =~ ^-?[[:d igit:]]*\. [[:d igit:]]+$ ]];
    luego echo "'$REPLY' es un número de coma
    flotante".
else
    echo "'$REPLY' no es un número de coma flotante".
fi

# ¿La entrada es un número entero?
si [[ $REPLY =~ ^-?[[:d igit:]]+$ ]]; luego
    echo "'$REPLY' es un número
    entero".
else
    echo "'$REPLY' no es un número entero".
fi
más
echo "La cadena '$REPLY' no es un nombre de archivo válido."
Fí
```

Este script solicita al usuario que introduzca un elemento. Posteriormente, el artículo se analizados para determinar su contenido. Como podemos ver, el script hace uso de muchos de los conceptos que hemos cubierto hasta ahora,

incluyendo las funciones de shell , [[ ]], (( )), el operador de control &&, y if, así como una buena dosis de expresiones regulares.

# Menús

Un tipo común de interactividad se denomina *impulsada por menús*. En los programas basados en menús, se presenta al usuario una lista de opciones y se le pide que elija una. Por ejemplo, podríamos imaginar un programa que presentara lo siguiente:

---

Por favor, seleccione:

1. Mostrar información del sistema
2. Mostrar espacio en disco
3. Mostrar la utilización del espacio en el hogar
0. Renunciar

---

Introduzca la selección [0-3] >

Usando lo que aprendimos al escribir nuestro `programa sys_info_page`, podemos construir un programa basado en menús para realizar las tareas en el menú anterior:

---

```
#!/bin/bash

# Read-Menu: Un programa de información del sistema basado
en menús claro
echo "
Por favor, seleccione:

1. Mostrar información del sistema
2. Mostrar espacio en disco
3. Mostrar la utilización del espacio en el hogar
0. Salir
"
read -p "Entrar en la selección [0-3] > "

si [[ $REPLY =~ ^[0-3]$ ]]; entonces
    si [[ $REPLY == 0 ]];
    entonces
        echo "Programa terminado".
        salida
    Fi
    si [[ $REPLY == 1 ]];
    entonces
        echo "Nombre de host:
$HOSTNAME" de tiempo de
actividad
        salida
    Fi
    si [[ $REPLY == 2 ]];
    entonces
        Salid
        a df
        -h
    Fi
    si [[ $REPLY == 3 ]];
    entonces
        if [[ $(id -u) -eq 0 ]];
        entonces
            echo "Utilización del espacio doméstico
(todos los usuarios)" du -sh /home/*
        más
        Salida
    Fi
```

```
echo "Home Space  
Utilization ($USER)"  
du -sh $HOME
```

```
más  
echo "Entrada no válida."  
>&2 salida 1  
Fí
```

---

Este guión se divide lógicamente en dos partes. La primera parte muestra el menú e ingresa la respuesta del usuario. La segunda parte identifica la respuesta y lleva a cabo la acción seleccionada. Observe el uso del comando `exit` en este script. Se utiliza aquí para evitar que el script ejecute código innecesario después de que se haya llevado a cabo una acción. La presencia de múltiples puntos de salida en un programa es generalmente una mala idea (hace que la lógica del programa sea más difícil de entender), pero funciona en este script.

## Nota final

En este capítulo, dimos nuestros primeros pasos hacia la interactividad, permitiendo a los usuarios ingresar datos en nuestros programas a través del teclado. Utilizando las técnicas presentadas hasta ahora, es posible escribir muchos programas útiles, como programas de cálculo especializados y interfaces fáciles de usar para herramientas de línea de comandos arcanas. En el próximo capítulo, nos basaremos en el concepto de programa basado en menús para hacerlo aún mejor.

## Crédito Extra

Es importante estudiar detenidamente los programas de este capítulo y tener una comprensión completa de la forma en que están estructurados lógicamente, ya que los programas que vendrán serán cada vez más complejos. A modo de ejercicio, reescriba los programas de este capítulo utilizando el comando `test` en lugar del comando compuesto `[[ ]]`. Sugerencia: Utilice `grep` para evaluar las expresiones regulares y, a continuación, evalúe su estado de salida. Esta será una buena práctica.

# 29

## **CONTROL DE FLUJO: BUCLE CON WHILE Y UNTIL**

En el capítulo anterior, desarrollamos un programa basado en menús para producir varios tipos de información del sistema. El programa funciona, pero todavía tiene un problema de usabilidad importante.

**Ejecuta una sola opción**

y luego termina. Peor aún, si se realiza una selección no válida, el programa termina con un error, sin dar al usuario la oportunidad de volver a intentarlo. Sería mejor si de alguna manera pudiéramos construir el programa de manera que pudiera repetir la visualización del menú y la selección una y otra vez, hasta que el usuario elija salir del programa.

En este capítulo, veremos un concepto de programación llamado *bucle*, que se puede usar para hacer que se repitan partes de los programas. El shell proporciona tres comandos compuestos para el bucle. Veremos dos de ellos en este capítulo y el tercero en el capítulo 33.



## Bucle

La vida diaria está llena de actividades repetidas. Ir a trabajar todos los días, pasear al perro y cortar una zanahoria son tareas que implican repetir una serie de pasos. Consideremos cortar una zanahoria en rodajas. Si expresamos esta actividad en pseudocódigo, podría tener un aspecto similar al siguiente:

1. Consigue una tabla de cortar.
2. Consigue un cuchillo.
3. Coloque la zanahoria en la tabla de cortar.
4. Cuchillo de elevación.
5. Avance la zanahoria.
6. Corta la zanahoria en rodajas.
7. Si la zanahoria entera está en rodajas, entonces déjelo, de lo contrario, vaya al paso 4.

Los pasos 4 a 7 forman un *bucle*. Las acciones dentro del bucle se repiten hasta que se alcanza la condición "zanahoria entera en rodajas".

## mien

### tras

bash puede expresar una idea similar. Supongamos que queremos mostrar cinco números en orden secuencial del 1 al 5. Un script bash podría construirse de la siguiente manera:

---

```
#!/bin/bash

# while-count: muestra una serie de números

count=1

mientras que [ $count -le 5
    ]; do echo $count
        count=$((count + 1))
hecho
echo "Terminado".
```

---

Cuando se ejecuta, este script muestra lo siguiente:

---

```
[me@linuxbox ~]$ while-count
1
2
3
4
5
Terminado.
```

---

La sintaxis del comando while es:

while *commands*; do *commands*; done

Como si, while evalúa el estado de salida de una lista de comandos. Siempre que el estado de salida sea 0, realiza los comandos dentro del bucle. En el script anterior, se crea el recuento de variables y se le asigna un valor inicial de 1. El comando while evalúa el estado de salida del comando de prueba. Siempre que el comando de prueba devuelva un estado de salida de 0, se ejecutan los comandos dentro del bucle. Al final de cada ciclo, se repite el comando de prueba. Después de seis iteraciones del bucle, el valor de count ha aumentado a 6, el comando de prueba ya no devuelve un estado de salida de 0 y el bucle termina. El programa continúa con la siguiente instrucción después del bucle.

Podemos usar un *bucle while* para mejorar el programa de menú de lectura del Capítulo 28:

```
#!/bin/bash

# while-menu: un programa de información del sistema controlado por menús

DELAY=3 # Número de segundos para mostrar los resultados

mientras que [[ $REPLY != 0 ]]; hacer
    claro
    gato <<- _EOF_
        Por favor, seleccione:

        1. Mostrar información del sistema
        2. Mostrar espacio en disco
        3. Mostrar la utilización del espacio en el hogar
        0. Renunciar

    _EF_
    read -p "Entrar en la selección [0-3] > "

    si [[ $REPLY =~ ^[0-3]$ ]];
        entonces si [[ $REPLY == 1
    ]]; entonces
        echo "Nombre de host:
$HOSTNAME" de tiempo de
actividad
Sueño $DELAY
    Fi
    si [[ $REPLY == 2 ]]; entonces
        df -h
        Sueño $DELAY
    Fi
    si [[ $REPLY == 3 ]]; entonces
        if [[ $(id -u) -eq 0 ]]; entonces
            echo "Utilización del espacio doméstico
(todos los usuarios)" du -sh /home/*
        más
            echo "Home Space Utilization ($USER)"
            du -sh $HOME
        Fi
        Sueño $DELAY
    Fi
    más
    i
hecho
F
```

```
echo "Entrada no  
válida".
```

```
Sueño $DELAY
```

---

```
echo "Programa terminado".
```

Al encerrar el menú en un bucle while, podemos hacer que el programa repita la visualización del menú después de cada selección. El bucle continúa mientras REPLY no sea igual a 0 y el menú se muestre de nuevo, dando al usuario la oportunidad de realizar otra selección. Al final de cada acción, se ejecuta un comando de suspensión para que el programa se detenga durante unos segundos para permitir que se vean los resultados de la selección antes de que se borre la pantalla y se vuelva a mostrar el menú. Una vez que REPLY es igual a 0, lo que indica la selección de "salir", el bucle termina y la ejecución continúa con la línea siguiente hecha.

## Salir de un bucle

bash proporciona dos comandos incorporados que se pueden usar para controlar el flujo del programa dentro de los bucles. El comando break termina inmediatamente un bucle y el control de programa se reanuda con la siguiente instrucción después del bucle. El comando continue hace que se omita el resto del bucle y el control del programa se reanude con la siguiente iteración del bucle. Aquí vemos una versión del programa del menú while que incorpora tanto la pausa como la continuación:

```
#!/bin/bash

# while-menu2: un programa de información del sistema controlado
por menús DELAY=3 # Número de segundos para mostrar los
resultados
```

```
Si bien es cierto; hacer
claro
gato <<- _EOF_
Por favor, seleccione:
```

1. Mostrar información del sistema
2. Mostrar espacio en disco
3. Mostrar la utilización del espacio en el hogar
0. Renunciar

```
_EF_
read -p "Entrar en la selección [0-3] > "
```

```
si [[ $REPLY =~ ^[0-3]$ ]]; entonces si [[ $REPLY == 1
]]; entonces
echo "Nombre de host:
$HOSTNAME" de tiempo de
actividad
Sueño $DELAY
continuar
Fí
si [[ $REPLY == 2 ]]; entonces
df -h
Sueño $DELAY
continuar
Fí
si [[ $REPLY == 3 ]]; entonces
if [[ $(id -u) -eq 0 ]]; entonces
```

```
echo "Utilización del espacio doméstico  
(todos los usuarios)" du -sh /home/*
```

```

más
    echo "Home Space Utilization ($USER)"
    du -sh $HOME
Fi
Sueño $DELAY
continuar
Fi
si [[ $REPLY == 0 ]]; entonces
quebrar
Fi
más
    echo "Entrada no válida."
    sueño $DELAY
Fi
hecho
echo "Programa terminado".

```

---

En esta versión del script, configuramos un *bucle sin fin* (uno que nunca termina por sí solo) usando el comando true para proporcionar un estado de salida a while. Dado que true siempre se cerrará con un estado de salida de 0, el bucle nunca terminará. Esta es una técnica de scripting sorprendentemente común. Dado que el bucle nunca terminará por sí solo, depende del programador proporcionar alguna forma de salir del bucle cuando sea el momento adecuado. En este script, el comando break se utiliza para salir del bucle cuando se elige la selección 0. El comando continue se ha incluido al final de las otras opciones de script para permitir una ejecución más eficiente. Mediante el uso de continue, el script omitirá el código que no sea necesario cuando se identifique una selección. Por ejemplo, si se elige e identifica la selección 1, no hay razón para realizar pruebas para las otras selecciones.

## hast

### a

El comando until es muy parecido a while, excepto que en lugar de salir de un bucle cuando se encuentra un estado de salida distinto de cero, hace lo contrario. Un *bucle until* continúa hasta que recibe un estado de salida 0. En nuestro script while-count, continuamos el bucle siempre que el valor de la variable count sea menor o igual que 5. Podríamos obtener el mismo resultado codificando el script con hasta:

```

#!/bin/bash

# until-count: muestra una serie de números

count=1

hasta [ $count -gt 5 ]; do
    echo $count
    count=$((count +
    1))
hecho
echo "Terminado".

```

---

Al cambiar la expresión de prueba a \$count -gt 5, until finalizará el bucle en el momento correcto. Decidir si usar el bucle while o until suele ser una cuestión de elegir el que permita escribir la prueba más

clara.

## Lectura de archivos con bucles

while y until puede procesar la entrada estándar. Esto permite que los archivos se procesen con bucles while y gota. En el siguiente ejemplo, mostraremos los contenidos del archivo *distros.txt* utilizado en los capítulos anteriores:

---

```
#!/bin/bash

# while-read: lee líneas de un archivo

mientras lee el lanzamiento de la

versión de la distribución; hacer
    printf "Distro: %s\tVersión: %s\tLiberado: %s\n" \
        $distro \
        $version \
        $release
hecho < distros.txt
```

---

Para redirigir un archivo al bucle, colocamos el operador de redirección después de la instrucción done. El bucle usará read para ingresar los campos del archivo redirigido. El comando read se cerrará después de que se lea cada línea, con un estado de salida 0 hasta que se alcance el final del archivo. En ese momento, saldrá con un estado de salida distinto de cero, terminando así el bucle. También es posible canalizar la entrada estándar en un bucle:

---

```
#!/bin/bash

# while-read2: leer líneas de un archivo

sort -k 1,1 -k 2n distros.txt | mientras lee el lanzamiento de la
    versión de la distribución; do printf "Distro: %s\tVersion:
        %s\tReleased: %s\n" \
        $distro \
        $version \
        $release
hecho
```

---

Aquí tomamos la salida del comando sort y mostramos el flujo de Mensaje de texto. Sin embargo, es importante recordar que, dado que una tubería ejecutará el bucle en un subshell, cualquier variable creada o asignada dentro del bucle se perderá cuando termine el bucle.

## Nota final

Con la introducción de los bucles y nuestros encuentros previos con ramificaciones, subrutinas y secuencias, hemos cubierto los principales tipos de control de flujo utilizados en los programas. Bash tiene algunos trucos más bajo la manga, pero son refinamientos de estos conceptos básicos.

# 30

## SOLUCIÓN DE PROBLEMAS

A medida que nuestros guiones se vuelven más complejos, es hora de echar un vistazo a lo que sucede cuando las cosas salen mal y no hacen lo que queremos. En este capítulo, veremos algunos de los tipos comunes de errores que ocurren en los scripts y describiremos algunas técnicas que se pueden usar para rastrear y erradicar problemas.

### Errores sintácticos

Una clase general de errores es *la sintáctica*. Los errores sintácticos implican escribir mal algún elemento de la sintaxis del shell. En la mayoría de los casos, este tipo de errores harán que el shell se niegue a ejecutar el script.

En las siguientes discusiones, usaremos este script para demostrar los tipos comunes de errores:

---

```
#!/bin/bash
```

```
# problema: script para demostrar errores comunes
```

```
número=1

si [ $number = 1 ]; entonces
    echo "El número es igual a 1".
else
    echo "El número no es igual a 1".
fi
```

---

Tal como está escrito, este script se ejecuta correctamente:

---

```
[me@linuxbox ~]$ problemas
El número es igual a 1.
```

---

### **Citas faltantes**

Editemos nuestro script y eliminemos la comilla final del argumento que sigue al primer comando echo:

---

```
#!/bin/bash

# problema: script para demostrar errores comunes

número=1

si [ $number = 1 ]; entonces
    echo "El número es igual a 1.
else
    echo "El número no es igual a 1".
fi
```

---

Mira lo que sucede:

---

```
[me@linuxbox ~]$ problemas
/home/me/bin/trouble: línea 10: EOF inesperado mientras se busca la coincidencia """
/home/me/bin/trouble: línea 13: error de sintaxis: fin inesperado del archivo
```

---

Genera dos errores. Curiosamente, los números de línea reportados no son donde se eliminó la cita faltante, sino mucho más tarde en el programa. Podemos ver por qué si seguimos el programa después de la cita que falta. bash continuará buscando la cita de cierre hasta que encuentre una, lo que hace inmediatamente después del segundo comando echo. bash se vuelve muy confuso después de eso, y la sintaxis del comando if se rompe porque el estado final ahora está dentro de una cadena entrecomillada (pero abierta).

En scripts largos, este tipo de error puede ser bastante difícil de encontrar. Usar un editor con resaltado de sintaxis ayudará. Si se instala una versión completa de vim , el resaltado de sintaxis se puede habilitar ingresando el comando:

```
:sintaxis en
```

## Tokens faltantes o inesperados

Otro error común es olvidarse de completar un comando compuesto, como if o while. Veamos qué sucede si eliminamos el punto y coma después de la prueba en el comando if.

---

```
#!/bin/bash

# problema: script para demostrar errores comunes

número=1

Si [ $number = 1 ] entonces
    echo "El número es igual a 1".
else
    echo "El número no es igual a 1".
fi
```

---

El resultado es el siguiente:

---

```
[me@linuxbox ~]$ problemas
/home/me/bin/trouble: línea 9: error de sintaxis cerca del token inesperado 'else'
/home/me/bin/trouble: línea 9: 'else'
```

---

De nuevo, el mensaje de error apunta a un error que se produce después del problema real. Lo que sucede es realmente muy interesante. Como recordamos, si acepta una lista de comandos y evalúa el código de salida del último comando de la lista. En nuestro programa, pretendemos que esta lista conste de un solo comando, [ , un sinónimo de prueba. El comando [ toma lo que sigue como una lista de argumentos, en nuestro caso, cuatro argumentos: \$number, =, 1 y ]. Con el punto y coma eliminado, la palabra then se agrega a la lista de argumentos, lo cual es sintácticamente legal. El siguiente comando echo también es legal. Se interpreta como otro comando en la lista de comandos que se evaluarán para un código de salida. El else se encuentra a continuación, pero está fuera de lugar, ya que el shell lo reconoce como una *palabra reservada* (una palabra que tiene un significado especial para el shell) y no como el nombre de un comando. De ahí el mensaje de error.

## Expansiones imprevistas

Es posible que haya errores que solo se produzcan de forma intermitente en un script. A veces el script se ejecutará bien y otras veces fallará debido a los resultados de una expansión. Si devolvemos el punto y coma que falta y cambiamos el valor de number a una variable vacía, podemos demostrar:

---

```
#!/bin/bash

# problema: script para demostrar errores comunes

número=
```

```
si [ $number = 1 ]; entonces
    echo "El número es igual a 1".
else
    echo "El número no es igual a 1".
fi
```

---

Al ejecutar el script con este cambio, se obtiene el siguiente resultado:

```
[me@linuxbox -]$ problemas
/home/me/bin/trouble: line 7: [: =: operador unario esperado El
número no es igual a 1.
```

---

Obtenemos este mensaje de error bastante críptico, seguido de la salida del segundo comando echo. El problema es la expansión de la variable numérica dentro del comando de prueba. Cuando el comando

---

```
[ $number = 1 ]
```

---

se expande con el número vacío, el resultado es este:

---

```
[ = 1 ]
```

---

que no es válido y se genera el error. El operador = es un operador binario (requiere un valor en cada lado), pero falta el primer valor, por lo que el comando de prueba espera un operador unario (como -z) en su lugar. Además, dado que la prueba falló (debido al error), el comando if recibe un código de salida distinto de cero y actúa en consecuencia, y se ejecuta el segundo comando echo.

Este problema se puede corregir agregando comillas alrededor del primer argumento en el comando de prueba:

---

```
[ "$number" = 1 ]
```

---

Luego, cuando se produzca la expansión, el resultado será este:

---

```
[ "" = 1 ]
```

---

lo que produce el número correcto de argumentos. Además de usarse con cadenas vacías, las comillas deben usarse en los casos en que un valor podría expandirse en cadenas de varias palabras, como con los nombres de archivo que contienen espacios incrustados.

## Errores lógicos

A diferencia de los errores sintácticos, *los errores lógicos* no impiden que se ejecute un script. El script se ejecutará, pero no producirá el resultado deseado debido a un problema con su lógica. Hay un sinnúmero de posibles errores lógicos, pero estos son algunos de los tipos más comunes que se encuentran en los scripts:

- **Expresiones condicionales incorrectas.** Es fácil codificar incorrectamente una instrucción if/then/else y llevar a cabo la lógica incorrecta. A veces la lógica se invertirá, o será incompleta.

- **Errores de "Off by one".** Al codificar bucles que emplean contadores, es posible pasar por alto que el bucle puede requerir que el conteo comience con 0, en lugar de 1, para que el conteo concluya en el punto correcto. Estos tipos de errores dan como resultado que un bucle "se desvíe del final" al contar demasiado lejos, o bien se pierda la última iteración del bucle al terminar una iteración demasiado pronto.
- **Situaciones imprevistas.** La mayoría de los errores lógicos son el resultado de que un programa encuentre datos o situaciones que no fueron previstos por el programador. También pueden incluir expansiones imprevistas, como un nombre de archivo que contiene espacios incrustados que se expanden en varios argumentos de comando en lugar de un solo nombre de archivo.

## Programación Defensiva

Es importante verificar las suposiciones al programar. Esto significa una evaluación cuidadosa del estado de salida de los programas y comandos que son utilizados por un script. He aquí un ejemplo, basado en una historia real. Un desafortunado administrador del sistema escribió un script para realizar una tarea de mantenimiento en un servidor importante. El script contenía las dos líneas de código siguientes:

---

```
cd  
$dir_nombre  
rm *
```

---

No hay nada intrínsecamente malo con estas dos líneas, siempre y cuando el directorio nombrado en la variable, `dir_name`, exista. Pero, ¿qué pasa si no es así? En ese caso, el comando `cd` falla y el script continúa a la siguiente línea y elimina los archivos en el directorio de trabajo actual. ¡No es el resultado deseado en absoluto! El desafortunado administrador destruyó una parte importante del servidor debido a esta decisión de diseño.

Veamos algunas formas en que se podría mejorar este diseño. En primer lugar, podría ser prudente hacer que la ejecución de `rm` dependa del éxito de `cd`:

---

```
cd $dir_nombre && rm *
```

---

De esta manera, si el comando `cd` falla, el comando `rm` no se lleva a cabo. Esto es mejor, pero aún deja abierta la posibilidad de que la variable, `dir_name`, no esté establecida o esté vacía, lo que resultaría en la eliminación de los archivos en la dirección de inicio del usuario. Esto también podría evitarse comprobando que `dir_name` realmente contiene el nombre de un directorio existente:

---

```
[[ -d $dir_nombre ]] && cd $dir_nombre && rm *
```

---

A menudo, es mejor terminar el script con un error cuando se produce una situación como la anterior:

---

```
if [[ -d $dir_nombre ]]; entonces
```

```
if cd $dir_name; then rm  
*
```

```
más
    echo "no se puede enviar un CD a
          '$dir_name'" >&2 salida 1
Fi
más
    echo "no existe tal directorio: '$dir_name'"
    >&2 salida 1
Fi
```

---

Aquí, comprobamos tanto el nombre, para ver que es el de una dirección existente. y el éxito del comando CD. Si se produce un error, se envía un mensaje de error descriptivo a error estándar y el script finaliza con un estado de salida de 1 para indicar un error.

### **Verificación de la entrada**

Una regla general de una buena programación es que si un programa acepta entradas, debe ser capaz de lidiar con cualquier cosa que reciba. Por lo general, esto significa que la entrada debe examinarse cuidadosamente para garantizar que solo se acepte la entrada válida para su posterior procesamiento. Vimos un ejemplo de esto en el capítulo anterior cuando estudiamos el comando de lectura. Un script contenía la siguiente prueba para verificar una selección de menú:

---

```
[[ $REPLY =~ ^[0-3]$ ]]
```

---

Esta prueba es muy específica. Devolverá un estado de salida 0 solo si la cadena devuelta por el usuario es un número en el intervalo de 0 a 3. No se aceptará nada más. A veces, este tipo de pruebas pueden ser muy difíciles de escribir, pero el esfuerzo es necesario para producir un script de alta calidad.

## **EL DISEÑO ES UNA FUNCIÓN DEL TIEMPO**

Cuando era un estudiante universitario que estudiaba diseño industrial, un sabio profesor afirmó que el grado de diseño de un proyecto estaba determinado por la cantidad de tiempo que se le daba al diseñador. Si te dieron 5 minutos para diseñar un dispositivo que mata moscas, diseñaeste un matamoscas. Si le dieran 5 meses, podría idear un "sistema antimoscas" guiado por láser.

El mismo principio se aplica a la programación. A veces, un script "rápido y sucio" servirá si va a ser utilizado sólo una vez y sólo por el programador. Ese tipo de script es común y debe desarrollarse rápidamente para que el esfuerzo sea económico. Tales scripts no necesitan muchos comentarios y controles defensivos. Por otro lado, si un script está destinado a uso en *producción*, es decir, un script que se utilizará una y otra vez para una tarea importante o por varios usuarios, necesita un desarrollo mucho más cuidadoso.

## Ensayo

Las pruebas son un paso importante en todo tipo de desarrollo de software, incluidos los scripts. Hay un dicho en el mundo del código abierto, "publica pronto, publica a menudo", que refleja este hecho. Al lanzarse temprano y con frecuencia, el software obtiene más exposición para su uso y pruebas. La experiencia ha demostrado que los errores son mucho más fáciles de encontrar, y mucho menos costosos de corregir, si se encuentran en una etapa temprana del ciclo de desarrollo.

### Talones

En una discusión anterior, vimos cómo se pueden usar los códigos auxiliares para verificar el flujo del programa. Desde las primeras etapas del desarrollo de guiones, son una técnica valiosa para verificar el progreso de nuestro trabajo.

Echemos un vistazo al problema anterior de eliminación de archivos y veamos cómo se podría codificar para facilitar las pruebas. Probar el fragmento de código original sería peligroso, ya que su propósito es eliminar archivos, pero podríamos modificar el código para que la prueba sea segura:

---

```
if [[ -d $dir_nombre ]]; entonces
    si cd $dir_nombre; entonces
        echo rm * # PRUEBAS
    más
        echo "no se puede enviar un CD a
        '$dir_name'" >&2 salida 1
    Fi
más
    echo "no existe tal directorio: '$dir_name'"
    >&2 salida 1
Fi
exit # TESTING
```

---

Dado que las condiciones de error ya generan mensajes útiles, no tenemos que agregar ninguno. El cambio más importante es colocar un comando echo justo antes del comando rm para permitir que el comando y su lista de argumentos expandida se muestren, en lugar de ejecutarse. Este cambio permite la ejecución segura del código. Al final del fragmento de código, colocamos un comando de salida para concluir la prueba y evitar que se lleve a cabo cualquier otra parte del script. La necesidad de esto variará según el diseño de el guión.

También incluimos algunos comentarios que actúan como "marcadores" para nuestros cambios relacionados con la prueba. Se pueden usar para ayudar a encontrar y eliminar los cambios una vez finalizadas las pruebas.

### Casos de prueba

Para realizar pruebas útiles, es importante desarrollar y aplicar buenos casos de prueba. Esto se hace eligiendo cuidadosamente los datos de entrada o las

condiciones de funcionamiento que

Refleja los casos *de borde* y *esquina*. En nuestro fragmento de código (que es muy simple), queremos saber cómo se comporta el código bajo tres condiciones específicas:

- `dir_name` contiene el nombre de un directorio existente.
- `dir_name` contiene el nombre de un directorio inexistente.
- `dir_name` está vacío.

Al realizar la prueba con cada una de estas condiciones, se logra una buena cobertura de la *prueba*.

Al igual que con el diseño, las pruebas también son una función del tiempo. No todas las funciones del script necesitan ser probadas exhaustivamente. Realmente se trata de determinar qué es lo más importante. Dado que podría ser muy destructivo si funcionara mal, nuestro fragmento de código merece una consideración cuidadosa tanto durante su diseño como durante sus pruebas.

## Depuración

Si las pruebas revelan un problema con un script, el siguiente paso es la depuración. "Un problema" generalmente significa que el script, de alguna manera, no está funcionando según las expectativas del programador. Si este es el caso, debemos determinar cuidadosamente qué es exactamente lo que el script está haciendo realmente y por qué. Encontrar errores a veces puede implicar mucho trabajo de detective.

Un guión bien diseñado intentará ayudar. Debe programarse a la defensiva para detectar condiciones anormales y proporcionar información útil al usuario. A veces, sin embargo, los problemas son extraños e inesperados, y se requieren técnicas más complejas.

### Encontrar el área problemática

En algunos guiones, particularmente los largos, a veces es útil aislar el área del alfabeto que está relacionada con el problema. Este no siempre será el error real, pero el aislamiento a menudo proporcionará información sobre la causa real. Una técnica que se puede utilizar para aislar el código es "comentar" secciones de un script. Por ejemplo, nuestro fragmento de eliminación de archivos podría modificarse para determinar si la sección eliminada estaba relacionada con un error:

---

```
if [[ -d $dir_nombre ]]; entonces
    if cd $dir_name; then rm
        *
    más
        echo "no se puede enviar un CD a
        '$dir_name'" >&2 salida 1
    Fi
# else
#     echo "No existe tal directorio: '$dir_name'"
>&2 #     Salida 1
Fi
```

---

Al colocar símbolos de comentario al principio de cada línea en una sección lógica de un script, evitamos que esa sección se ejecute. A continuación, se pueden volver a realizar pruebas para ver si la eliminación del código tiene algún impacto en el comportamiento del error.

## Trazado

Los errores suelen ser casos de flujo lógico inesperado dentro de un script. Es decir, partes del script nunca se ejecutan o se ejecutan en el orden equivocado o en el momento equivocado. Para ver el flujo real del programa, utilizamos una técnica llamada *rastreo*.

Un método de rastreo consiste en colocar mensajes informativos en un script que muestren la ubicación de la ejecución. Podemos agregar mensajes a nuestro fragmento de código:

---

```
Echo "Preparándose para eliminar archivos" >&2
if [[ -d $dir_nombre ]]; entonces
    si cd $dir_nombre; entonces
        echo "Eliminación de archivos" >&2
        rm *
    más
        echo "no se puede enviar un CD a
        '$dir_name'" >&2 salida 1
    Fi
más
    echo "no existe tal directorio: '$dir_name"
    >&2 salida 1
Fi
echo " Eliminación de archivos completada" >&2
```

---

Enviamos los mensajes a error estándar para separarlos de la salida normal. Tampoco aplicamos sangría a las líneas que contienen los mensajes, por lo que es más fácil encontrar cuándo es el momento de eliminarlos.

Ahora, cuando se ejecuta el script, es posible ver que se ha realizado la eliminación del archivo:

---

```
[me@linuxbox ~]$ script-de-
eliminación preparándose para
eliminar archivos eliminando
archivos
Eliminación de archivos
completa [me@linuxbox
~]$
```

---

bash también proporciona un método de rastreo, implementado por la opción `-x` y el comando `set` con la opción `-x`. Usando nuestro script de problemas anterior, podemos activar el rastreo para todo el script agregando la opción `-x` a la primera línea:

---

```
#!/bin/bash -x
# problema: script para demostrar errores comunes
```

---

número=1

```
si [ $number = 1 ]; entonces
    echo "El número es igual a 1".
else
    echo "El número no es igual a 1".
fi
```

---

Cuando se ejecuta, los resultados se ven así:

```
[me@linuxbox -]$ problemas
+ número=1
+ '[' 1 = 1 ']'
+ echo 'El número es igual a 1.' El
número es igual a 1.
```

---

Con el trazado habilitado, vemos los comandos realizados con las expansiones aplicadas. Los signos más iniciales indican la visualización de la traza para distinguirlos de las líneas de salida regular. El signo más es el carácter predeterminado para la salida de seguimiento. Está contenido en la variable de shell PS4 (prompt string 4). El contenido de esta variable se puede ajustar para que el mensaje sea más útil.

Aquí, lo modificamos para incluir el número de línea actual en el script donde se realiza el seguimiento. Tenga en cuenta que se requieren comillas simples para evitar la expansión hasta que se use realmente el mensaje:

```
[me@linuxbox -]$ exportar PS4='$LINENO + '
[me@linuxbox -]$ problemas
5 + número=1
7 + '[' 1 = 1 ']'
8 + echo 'El número es igual a 1.' El
número es igual a 1.
```

---

Para realizar un seguimiento en una parte seleccionada de un script, en lugar de todo el script, podemos usar el comando set con la opción -x:

```
#!/bin/bash

# problema: script para demostrar errores comunes

número=1

set -x # Activar el rastreo
si [ $number = 1 ]; entonces
    echo "El número es igual a 1".
más
    echo "El número no es igual a 1".
Fi
set +x # Desactivar el rastreo
```

---

Usamos el comando set con la opción -x para activar el rastreo y el comando +x para desactivar el seguimiento. Esta técnica se puede utilizar para examinar varias partes de un guión problemático.

## *Examen de valores durante la ejecución*

A menudo es útil, junto con el seguimiento, mostrar el contenido de las variables para ver el funcionamiento interno de un script mientras se está ejecutando. La aplicación de instrucciones de eco adicionales generalmente hará el truco:

---

```
#!/bin/bash

# problema: script para demostrar errores comunes

número=1

Echo "número=$número" #debug
set -x # Activa el trazado
si [ $número = 1 ];
entonces
    echo "El número es igual a 1".
más
    echo "El número no es igual a 1".
Fi
set +x # Desactivar el rastreo
```

---

En este ejemplo trivial, simplemente mostramos el valor de la variable número y marcamos la línea agregada con un comentario para facilitar su posterior identificación y eliminación. Esta técnica es particularmente útil cuando se observa el comportamiento de los bucles y la aritmética dentro de los scripts.

## **Nota final**

En este capítulo, analizamos solo algunos de los problemas que pueden surgir durante el desarrollo del script. Por supuesto, hay muchos más. Las técnicas descritas aquí permitirán encontrar los errores más comunes. La depuración es un arte que se puede desarrollar a través de la experiencia, tanto en evitar errores (pruebas constantes durante todo el desarrollo) como en encontrar errores (uso efectivo del rastreo).



# 31

## **CONTROL DE FLUJO: RAMIFICACIÓN CON ESTUCHE**

En este capítulo, continuaremos examinando el control del flujo. En el Capítulo 28, construimos algunos menús simples y construimos la lógica utilizada para actuar sobre la selección de un usuario. Para ello, utilizamos una serie de **comandos if** para identificar cuál de las opciones posibles se había seleccionado. Este tipo de constructo aparece con frecuencia en los programas, hasta el punto de que muchos lenguajes de programación (incluido el shell) proporcionan un mecanismo de control de flujo para decisiones de opción múltiple.



## caso

El comando compuesto de opción múltiple bash se llama case. Tiene la siguiente sintaxis:

```
palabra de caso en  
    [patrón [| patrón]...) comandos ;;...  
Esac
```

Si nos fijamos en el programa de menú de lectura del capítulo 28, vemos la lógica utilizada para actuar sobre la selección de un usuario:

---

```
#!/bin/bash

# Read-Menu: Un programa de información del sistema

basado en menús claros
echo "
Por favor, seleccione:

1. Mostrar información del sistema
2. Mostrar espacio en disco
3. Mostrar la utilización del espacio en el hogar
0. Salir
"
read -p "Entrar en la selección [0-3] > "

si [[ $REPLY =~ ^[0-3]$ ]]; then
    entonces si [[ $REPLY == 0 ]]; entonces
        echo "Programa terminado."
        exit
    Fi
    si [[ $REPLY == 1 ]]; entonces
        echo "Nombre de host:
$HOSTNAME" de tiempo de
actividad
salida
    Fi
    si [[ $REPLY == 2 ]]; entonces
        Salida
        a df
        -h
    Fi
    si [[ $REPLY == 3 ]]; entonces
        if [[ $(id -u) -eq 0 ]]; entonces
            echo "Utilización del espacio doméstico
(todos los usuarios)" du -sh /home/*
        más
            echo "Home Space Utilization ($USER)"
            du -sh $HOME
        Salida
        más
    Fi
    más
        echo "Entrada no válida."
        >&2 salida 1
Fi
```

---

Usando case, podemos reemplazar esta lógica con algo más simple:

---

```
#!/bin/bash

# case-menu: un programa de información del sistema

basado en menús claros
echo "
Por favor, seleccione:

1. Mostrar información del sistema
2. Mostrar espacio en disco
3. Mostrar la utilización del espacio en el hogar
0. Salir
"
read -p "Entrar en la selección [0-3] > "

Caso $REPLY en
    0)      echo "Programa terminado."
            exit
            ;;
    1)      echo "Nombre de host:
$HOSTNAME" de tiempo de
actividad
            ;;
    2)      df -h
            ;;
    3)      if [[ $(id -u) -eq 0 ]]; entonces
            echo "Utilización del espacio doméstico
(todos los usuarios)" du -sh /home/*
más
            echo "Home Space Utilization ($USER)"
du -sh $HOME
        Fi
        ;;
    *)      echo "Entrada no válida"
        >&2 salida 1
        ;;
Esac
```

---

El comando case examina el valor de *word*: en nuestro ejemplo, el valor de la variable REPLY— y luego intenta compararla con uno de los *patrones especificados*. Cuando se encuentra una coincidencia, *se ejecutan los comandos* asociados con el patrón especificado. Una vez que se encuentra una coincidencia, no se intentan más coincidencias.

## Patrones

Los patrones utilizados por caso son los mismos que los utilizados por la expansión del nombre de la ruta. Los patrones terminan con un carácter ). La Tabla 31-1 muestra algunos patrones válidos.

Tabla 31-1: Ejemplos de patrones de casos

Patrón	Descripción
a)	Coincide si la <i>palabra</i> es igual a a.
[[:alfa:]]	Coincide si <i>palabra</i> es un solo carácter alfabético.
???)	Coincide si <i>palabra</i> tiene exactamente tres caracteres.
*.txt)	Coincide si <i>palabra</i> termina con los caracteres .Txt.
*)	Coincide con cualquier valor de <i>palabra</i> . Es una buena práctica incluir esto como el último patrón en un caso para capturar cualquier valor de <i>palabra</i> que no coincidía con un patrón anterior; es decir, para detectar cualquier posible valor no válido.

He aquí un ejemplo de patrones en funcionamiento:

---

```
#!/bin/bash

read -p "enter word > "

caso $REPLY en
    [[:alfa:]]
    [ABC][0-9]
    ???
    *.txt)
    *)
Esac

    echo "Es un carácter alfabético único." ;;
    echo "Es A, B o C Seguido por un dígito." ;;
    echo "Es tres caracteres." ;;
    echo "Es un Terminación de palabra en '.txt'" ;;
    echo "es otra cosa." ;;
```

---

### Combinación de múltiples patrones

También es posible combinar varios patrones utilizando el carácter de la tubería vertical como separador. Esto crea un patrón condicional "o". Esto es útil para cosas como el manejo de caracteres en mayúsculas y minúsculas. Por ejemplo:

---

```
#!/bin/bash

# case-menu: un programa de información del sistema

basado en menús claro
echo "
Por favor, seleccione:

A. Mostrar información del sistema
B. Mostrar espacio en disco
C. Mostrar la utilización del espacio en el hogar
Pregun
ta.
Salir "
read -p "Ingrese la selección [A, B, C o Q] > "

Caso $REPLY en
    Q|Q)    echo "Programa terminado".
```

---

salida  
;;

```

una|A) echo "Nombre de host:
$HOSTNAME" de tiempo de
actividad
;;
b|B) df -h
;;
c|C) si [[ $(id -u) -eq 0 ]]; entonces
      echo "Utilización del espacio doméstico
(todos los usuarios)" du -sh /home/*
más
      echo "Home Space Utilization ($USER)" du
      -sh $HOME
    Fi
;;
*) echo "Entrada no válida"
>&2 salida 1
;;
Esac

```

---

Aquí, modificamos el programa case-menu para usar letras en lugar de dígitos para Selección de menú. Tenga en cuenta que los nuevos patrones permiten la entrada de letras mayúsculas y minúsculas.

## Nota final

El comando case es una adición útil a nuestra bolsa de trucos de programación. Como veremos en el próximo capítulo, es la herramienta perfecta para manejar ciertos tipos de problemas.



# 32

## PARÁMETROS POSICIONALES

Una característica que ha faltado en nuestros programas es la capacidad de aceptar y procesar opciones y argumentos de la línea de comandos. En este capítulo, examinaremos las características del shell que permiten a nuestros programas acceder al contenido de la línea de comandos.

### Acceso a la línea de comandos

El shell proporciona un conjunto de variables denominadas *parámetros posicionales* que contienen las palabras individuales en la línea de comandos. Las variables se denominan del 0 al 9. Se pueden demostrar de esta manera:

---

```
#!/bin/bash

# posit-param: script para ver los parámetros de la línea de
# comandos echo "
\$0 = \$0
\$1 = \$1
\$2 = \$2
```



```
\$3 = $3
\$4 = $4
\$5 = $5
\$6 = $6
\$7 = $7
\$8 = $8
\$9 = $9
"
```

---

Este script muy simple muestra los valores de las variables \$0 a \$9.  
Cuando se ejecuta sin argumentos de línea de comandos:

```
[me@linuxbox ~]$ posit-param
```

```
$0 = /home/me/bin/posit-param
$1 =
$2 =
$3 =
$4 =
$5 =
$6 =
$7 =
$8 =
$9 =
```

---

Incluso cuando no se proporcionan argumentos, \$0 siempre contendrá el primer elemento que aparece en la línea de comandos, que es el nombre de la ruta del programa que se está ejecutando. Cuando se proporcionan argumentos, vemos los resultados:

```
[me@linuxbox ~]$ posit-param a b c d
```

```
$0 = /home/me/bin/posit-param
$1 = un
$2 = b
$3 = c
$4 = d
$5 =
$6 =
$7 =
$8 =
$9 =
```

---

**Nota:** De hecho, puede acceder a más de nueve parámetros mediante la expansión de parámetros. Para especificar un número mayor que nueve, rodee el número entre llaves; por ejemplo  
\$\{10\}, \$\{55\}, \$\{211\}, etc.

### Determinación del número de argumentos

El shell también proporciona una variable, \$# , que produce el número de argumentos en la línea de comandos:

```
#!/bin/bash
```

```
# posit-param: script para ver los parámetros de la línea de comandos
```

```
eco "
Número de argumentos: $#
$0 = $0
$1 = $1
$2 = $2
$3 = $3
$4 = $4
$5 = $5
$6 = $6
$7 = $7
$8 = $8
$9 = $9
"
```

---

El resultado:

```
[me@linuxbox ~]$ posit-param a b c d
```

```
Número de argumentos: 4
$0 = /home/me/bin/posit-param
$1 = un
$2 = b
$3 = c
$4 = d
$5 =
$6 =
$7 =
$8 =
$9 =
```

---

### *shift: obtener acceso a muchos argumentos*

Pero, ¿qué ocurre cuando le damos al programa una gran cantidad de argumentos como este:

```
[me@linuxbox ~]$ posit-param *
```

```
Número de argumentos: 82
$0 = /home/me/bin/posit-param
$1 = direcciones.ldif
$2 = contenedor
$3 = bookmarks.html
$4 = debian-500-i386-netinst.iso
$5 = debian-500-i386-netinst.jigdo
$6 = debian-500-i386-netinst.template
$7 = debian-cd_info.tar.gz
$8 = Escritorio
$9 = dirlist-bin.txt
```

---

En este sistema de ejemplo, el comodín \* se expande en 82 argumentos. ¿Cómo podemos procesar tantos? El shell proporciona un método, aunque torpe, para hacer esto. El comando shift hace que cada parámetro "se mueva hacia abajo uno" cada vez que se ejecuta. De hecho, mediante el uso de shift, es posible arreglárselas con un solo parámetro (además de \$0, que nunca cambia).

---

```
#!/bin/bash

# posit-param2: script para mostrar todos los
# argumentos count=1

mientras que [[ $# -gt 0 ]]; hacer
    echo "Argumento $count = $1"
    count=$((count + 1))
    turno
hecho
```

---

Cada turno de tiempo que se ejecuta, el valor de \$2 se mueve a \$1, el valor de

\$3 se mueve a \$2, y así sucesivamente. El valor de \$# también se reduce en 1.

En el programa posit-param2, creamos un bucle que evalúa el número de argumentos restantes y continúa mientras haya al menos uno.

Mostramos el argumento actual, incrementamos el recuento de variables con cada iteración del bucle para proporcionar un recuento continuo del número de argumentos procesados y, finalmente, ejecutamos un turno para cargar \$1 con el siguiente argumento.

Este es el programa en funcionamiento:

---

```
[me@linuxbox ~]$ posit-param2 a b c d
Argumento 1 =
a Argumento 2
= b Argumento
3      =      c
Argumento 4 =
d
```

---

### *Aplicaciones sencillas*

Incluso sin desplazamiento, es posible escribir aplicaciones útiles utilizando parámetros posicionales. A modo de ejemplo, aquí hay un programa simple de información de archivos:

---

```
#!/bin/bash

# file_info: programa simple de información de archivos

PROGNAME=$(basename $0)

si [[ -e $1 ]]; entonces
    echo -e "\nTipo de
archivo:" fichero $1
    echo -e "\nEstado del
archivo:" stat $1
else
    echo "$PROGNAME: uso: $PROGNAME archivo"
    >&2 salida 1
fi
```

---

Este programa muestra el tipo de archivo (determinado por el comando `file`) y el estado del archivo (del comando `stat`) de un archivo especificado. Una característica interesante de este programa es la variable `PROGNAME`. Se le da el

valor que resulta del comando basename \$0. El comando basename elimina el atributo

parte inicial de un nombre de ruta, dejando solo el nombre base de un archivo. En nuestro ejemplo, basename elimina la parte inicial del nombre de ruta contenida en el parámetro \$0, el nombre de ruta completo de nuestro programa de ejemplo. Este valor es útil cuando se construyen mensajes, como el mensaje de uso al final del programa. Cuando se codifica de esta manera, se puede cambiar el nombre del script y el mensaje se ajusta automáticamente para contener el nombre del programa.

### ***Uso de parámetros posicionales con funciones de shell***

Al igual que los parámetros posicionales se utilizan para pasar argumentos a los scripts de shell, también se pueden utilizar para pasar argumentos a las funciones de shell. Para demostrarlo, convertiremos el script file\_info en una función de shell:

---

```
file_info () {  
    # file_info: función para mostrar la  
    # información del archivo si [[ -e $1 ]]; then  
    #     echo -e "\nTipo de  
    #     archivo:" fichero $1  
    #     echo -e "\nEstado del  
    #     archivo:" stat $1  
    # más  
    #     echo "$FUNCNAME: uso: $FUNCNAME archivo" >&2  
    #     return 1  
    # Fí  
}
```

---

Ahora, si un script que incorpora la función de shell file\_info llama a la función con un argumento de nombre de archivo, el argumento se pasará a la función.

Con esta capacidad, podemos escribir muchas funciones de shell útiles que se pueden usar no solo en scripts sino también dentro del archivo *.bashrc*.

Observe que la variable PROGNAME se cambió a la variable de shell FUNCNAME. El shell actualiza automáticamente esta variable para realizar un seguimiento de la función de shell ejecutada actualmente. Tenga en cuenta que \$0 siempre contiene el nombre completo de la ruta del primer elemento en la línea de comandos (es decir, el nombre del programa) y no contiene el nombre de la función de shell como podríamos esperar.

## **Manejo masivo de los parámetros posicionales**

A veces es útil gestionar todos los parámetros posicionales como un grupo. Por ejemplo, es posible que queramos escribir un *envoltorio* alrededor de otro programa. Esto significa que creamos un script o una función de shell que simplifica la ejecución de otro programa. El contenedor proporciona una lista de opciones arcanas de línea de comandos y, a continuación, pasa una lista de argumentos al programa de nivel inferior.

El shell proporciona dos parámetros especiales para este propósito. Ambos se expanden en la lista completa de parámetros posicionales,

pero difieren en formas bastante sutiles. En la tabla 32-1 se describen estos parámetros.

**Tabla 32-1: Los parámetros especiales \* y @**

Parámetro	Descripción
\$*	Se expande a la lista de parámetros posicionales, empezando por 1. Cuando está entre comillas dobles, se expande en una cadena entre comillas dobles que contiene todos los parámetros posicionales, cada uno separado por el primer carácter de la variable de shell IFS (por defecto, un carácter de espacio).
\$@	Se expande a la lista de parámetros posicionales, empezando por 1. Cuando está entre comillas dobles, expande cada parámetro posicional en una palabra separada rodeada de comillas dobles.

A continuación, se muestra un script que muestra estos parámetros especiales en acción:

```
#!/bin/bash

# posit-params3 : script para demostrar $* y $@

print_params () {
    echo "$1 = $1"
    echo "$2 = $2"
    echo "$3 = $3"
    echo "$4 = $4"
}

pass_params () {
    echo -e "\n" '$*' ; print_params $*
    echo -e "\n" "$*" ; print_params "$*"
    echo -e "\n" '$@' ; print_params $@
    echo -e "\n" "$@" ; print_params "$@"
}

pass_params "palabra" "palabras con espacios"
```

En este programa bastante enrevesado, creamos dos argumentos, palabra y palabras con espacios, y los pasamos a la función pass\_params. Esta función, a su vez, los pasa a la función print\_params, utilizando cada uno de los cuatro métodos disponibles con los parámetros especiales \$\* y \$@. Cuando se ejecuta, el script revela las diferencias:

```
[me@linuxbox ~]$ posit-params3
```

```
$* :
$1 = palabra
$2 = palabras
$3 = con
$4 = espacios

"$*" :
$1 = palabras con espacios
$2 =
```

```
$3 =  
$4 =  
  
$@ :  
$1 = palabra  
$2 = palabras  
$3 = con  
$4 = espacios
```

```
"$@" :  
$1 = palabra  
$2 = palabras con espacios  
$3 =  
$4 =
```

---

Con nuestros argumentos, tanto \$\* como \$@ producen un resultado de cuatro palabras: palabra, palabras, con y espacios. "\$\*" produce un resultado de una sola palabra: palabras con espacios. "\$@" produce un resultado de dos palabras: palabra y palabras con espacios.

Esto coincide con nuestra intención real. La lección que se puede extraer de esto es que, a pesar de que el shell proporciona cuatro formas diferentes de obtener la lista de parámetros posicionales, "\$@" es, con mucho, la más útil para la mayoría de las situaciones, porque preserva la integridad de cada parámetro posicional.

## Una aplicación más completa

Después de un largo paréntesis, vamos a reanudar los trabajos en nuestro programa sys\_info\_page. Nuestra próxima adición agregará varias opciones de línea de comandos al programa de la siguiente manera:

- **Archivo de salida.** Agregaremos una opción para especificar un nombre para un archivo que contendrá la salida del programa. Se especificará como archivo -f o archivo --file.
- **Modo interactivo.** Esta opción solicitará al usuario un nombre de archivo de salida y determinará si el archivo especificado ya existe. Si es así, se preguntará al usuario antes de que se sobrescriba el archivo existente. Esta opción se especificará mediante -i o --interactive.
- **Ayuda.** Se puede especificar -h o --help para que el programa emita un mensaje de uso informativo.

Este es el código necesario para implementar el procesamiento de la línea de comandos:

---

```
uso () {  
    echo "$PROGNAME: uso: $PROGNAME [archivo -f | -i]"  
    return  
}  
  
# opciones de línea de comandos de  
  
proceso interactivo=  
nombre de archivo=
```

mientras que [[ -n \$1 ]]; hacer  
Caso \$1 en

```

-f | --archivo) turno
                nombre de
                archivo=$1
;;
-i | --interactivo) interactivo=1
;;
-h | --ayuda) uso
salid
a
;;
*) Uso >&2
      Salida 1
;;
hecho Cam
      bio
      de
      ESAC

```

---

Primero, agregamos una función de shell llamada `usage` para mostrar un mensaje cuando se invoca la opción de ayuda o se intenta una opción desconocida.

A continuación, comenzamos el ciclo de procesamiento. Este bucle continúa mientras el parámetro positivo `$1` no está vacío. En la parte inferior del bucle, tenemos un comando `shift` para avanzar los parámetros posicionales y garantizar que el bucle termine finalmente.

Dentro del bucle, tenemos una instrucción `case` que examina el parámetro positional actual para ver si coincide con alguna de las opciones admitidas. Si se encuentra un parámetro compatible, se actúa en consecuencia. De lo contrario, se muestra el mensaje de uso y el script finaliza con un error.

El parámetro `-f` se maneja de una manera interesante. Cuando se detecta, hace que ocurra un desplazamiento adicional, que avanza el parámetro positional `$1` al argumento de nombre de archivo proporcionado a la opción `-f`.

A continuación, añadimos el código para implementar el modo interactivo:

---

```

# Modo interactivo

si [[ -n $interactive ]];
entonces mientras sea
verdadero; haga
    read -p "Ingrese el nombre del archivo de salida
: " nombre de archivo if [[ -e $filename ]];
entonces
    read -p "'$filename' existe. ¿Sobrescribir? [s/n/q] > "
caso $REPLY en
    Y|y)   quebrar
    ;;
    Q|q)   echo "Programa terminado".
salida
    ;;
    *)    continuar
    ;;
Esac
elif [[ -z $filename ]]; A
continuación, continúe
h      ho
  F
c

```

else fi

que  
brar

---

Si la variable interactiva no está vacía, se inicia un bucle sin fin, que contiene la solicitud de nombre de archivo y el código de manejo de archivos existente posterior. Si el archivo de salida deseado ya existe, se le pide al usuario que sobrescriba, elija otro nombre de archivo o salga del programa. Si el usuario elige sobrescribir un archivo existente, se ejecuta una interrupción para terminar el bucle. Tenga en cuenta que la instrucción case solo se detecta si el usuario decide sobrescribir o salir. Cualquier otra opción hace que el bucle continúe y vuelva a preguntar al usuario.

Con el fin de implementar la característica de nombre de archivo de salida, primero debemos convertir el código de escritura de página existente en una función de shell, por razones que se aclararán en un momento:

---

```
write_html_page () {
    gato <<- _EOF_
    <HTML>
        <CABEZ
            A>      <TÍTULO>$TITLE</TÍTULO>
        </CABEZ
            A>      <H1>$TITLE</H1>
            <CUERP
                O>      $(report_uptime)
                        $(report_disk_space)
                        $(report_home_space)

        </HTML>
        Devoluc
        ión de   </CUERP
        _EOF_   O>
    }

# página html de salida

si [[ -n $filename ]]; entonces
    si toque $filename && [[ -f $filename ]]; entonces
        write_html_page > $filename
    más
        echo "$PROGNAME: No se puede escribir el archivo
        '$filename'" >&2 salida 1
    Fi
más
    write_html_page
Fi
```

---

El código que maneja la lógica de la opción -f aparece al final de el listado que se muestra arriba. En él, probamos la existencia de un nombre de archivo y, si se encuentra uno, se realiza una prueba para ver si el archivo es realmente escribible. Para ello, se realiza un toque, seguido de una prueba para determinar si el archivo resultante es un archivo normal. Estas dos pruebas se encargan de las situaciones en las que se introduce un nombre de ruta no válido ( se producirá un error táctil) y, si el archivo ya existe, que es un archivo regular.

Como podemos ver, se llama a la función write\_html\_page para realizar la generación real de la página. Su salida se dirige a la salida

estándar (si el nombre de archivo de la variable `está vacío`) o se redirige al archivo especificado.

## Nota final

Con la adición de parámetros posicionales, ahora podemos escribir scripts bastante funcionales. Para tareas simples y repetitivas, los parámetros posicionales permiten escribir funciones de shell muy útiles que se pueden colocar en el *archivo .bashrc* de un usuario.

Nuestro programa `sys_info_page` ha crecido en complejidad y sofisticación. Aquí hay una lista completa, con los cambios más recientes destacados:

---

```
#!/bin/bash

# sys_info_page: programa para generar una página de información del sistema

PROGNAME=$(nombre base $0)
TITLE="Informe de información del sistema para $HOSTNAME"
CURRENT_TIME=$(date +"%x %r %Z")
TIME_STAMP="Generado $CURRENT_TIME, por $USER"

report_uptime () {
    gato <<- _EOF_
        <H2>Tiempo de actividad del sistema</H2>
        <PRE>$tiempo de actividad</PRE>
    _EF_
    devolución
}

report_disk_space () {
    gato <<- _EOF_
        <H2>Utilización de espacio en disco</H2>
        <PRE>$(df -h)</PRE>
    _EF_
    devolución
}

report_home_space () {
    if [[ $(id -u) -eq 0 ]];
        entonces gato <<-
    _EOF_
        <H2>Utilización del espacio en el hogar (todos los
    usuarios)</H2>
        <PRE>$(du -sh /home/*)</PRE>
    _EF_
    más
    gato <<- _EOF_
        <H2>Utilización del espacio en el hogar ($USER)</H2>
        <PRE>$(du -sh $HOME)</PRE>
    _EF_
    Deva
    lució
    n de
    Fi
}

uso () {
    echo "$PROGNAME: uso: $PROGNAME [archivo -f | -i]"
    return
}

write_html_page () {
    gato <<- _EOF_
```

<HTML>  
<CABEZA>

```

        <TÍTULO>$TITLE</TÍTULO>
    </CABEZ
    A>
    <CUERP  <H1>$TITLE</H1>
    O>      <P>$TIME_SELLO</P>
            $(report_uptime)
            $(report_disk_space)
            $(report_home_space)

    </HTML>
Devolu  </CUERP
ción     O>
de
_EOF_
}

# Opciones de la línea de comandos del proceso

interactivo=
nombre de
archivo=

mientras que [[ -n $1 ]]; hacer
    Caso $1 en
        -f | --archivo)      turno
                            nombre de
                            archivo=$1
                            ;;
        -Yo | --interactivo) interactivo=1
                            ;;
        -h | --help)usage
                            salid
                            a
                            ;;
        *)                  Uso >&2
                            Salida 1
                            ;;
Cambio
de ESAC
hecho

# Modo interactivo

si [[ -n $interactive ]];
    entonces mientras sea
    verdadero; haga
        read -p "Ingrese el nombre del archivo de salida:
        " nombre del archivo si [[ -e $filename ]];
    entonces
        read -p "'$filename' existe. ¿Sobrescribir? [s/n/q] > "
        caso $REPLY en
            Y|y)    quebrar
                    ;;
            Q|q)    echo "Programa terminado."
                    exit
                    ;;
            *)      continuar
                    ;;
Esac
Fi
hecho

```

**Fi**

# página html de salida

```
si [[ -n $filename ]]; entonces
    si toque $filename && [[ -f $filename ]]; entonces
        write_html_page > $filename
    más
        echo "$PROGNAME: No se puede escribir el archivo
'$filename'" >&2 salida 1
    Fi
más
    write_html_page
Fi
```

---

Nuestro guión es bastante bueno ahora, pero aún no hemos terminado. En el siguiente , añadiremos una última mejora a nuestro guión.

# 33

## CONTROL DE FLUJO: BUCLE CON FOR

En este último capítulo sobre el control de flujo, veremos otra de las construcciones de bucle del shell. El *bucle for* difiere de los bucles while y until en que proporciona un medio para procesar secuencias durante un bucle. Esto resulta ser muy útil a la hora de programar. De acuerdo con esto, el bucle for es una construcción muy popular en las secuencias de comandos bash.

Un bucle for se implementa, naturalmente, con el comando for. En las versiones modernas de bash, for está disponible en dos formas.

### para: Forma de concha tradicional

La sintaxis original del comando for es la siguiente:

```
para la variable [en palabras]; do  
    Comandos  
hecho
```



donde *variable* es el nombre de una variable que se incrementará durante la ejecución del bucle, *words* es una lista opcional de elementos que se asignarán secuencialmente a *la variable*, y *commands* son los comandos que se ejecutarán en cada iteración del bucle.

El comando `for` es útil en la línea de comandos. Podemos demostrar fácilmente cómo funciona:

---

```
[me@linuxbox ~]$ for i in A B C D; hacer echo $i; hecho
A
B
C
D
```

---

En este ejemplo, `for` se da una lista de cuatro palabras: *A*, *B*, *C* y *D*. Con una lista de cuatro palabras, el bucle se ejecuta cuatro veces. Cada vez que se ejecuta el bucle, se asigna una palabra a la variable *i*. Dentro del bucle, tenemos un comando `echo` que muestra el valor de *i* para mostrar la asignación. Al igual que con los bucles `while` y `gota`, la palabra clave `done` cierra el bucle.

La característica realmente poderosa de `for` es la cantidad de formas interesantes en las que podemos crear la lista de palabras. Por ejemplo, podemos usar la expansión de llaves:

---

```
[me@linuxbox ~]$ for i in {A.. D}; hacer echo $i; hecho
A
B
C
D
```

---

o expansión del nombre de la ruta:

---

```
[me@linuxbox ~]$ for i in distros*.txt; do echo $i; done
distros-by-date.txt
distros-dates.txt
distros-key-names.txt
distros-key-vernums.txt
distros-names.txt
distros.txt
distros-vernums.txt
distros-versions.txt
```

---

o sustitución de comandos:

---

```
#!/bin/bash

# longest-word : encuentra la cadena más
# larga en un archivo mientras [[ -n $1 ]]; hacer
# si [[ -r $1 ]]; entonces
#   max_word=
#   max_len=0
#   for i en $(strings $1); do
#     len=$(echo $i | wc -c)
#     si ( len > max_len );
#       entonces
#         max_len=$len
#         max_word=$i
```

**Fi**

```

hecho
echo "$1: '$max_palabra' ($max_len caracteres)"
Ca
mbi
o de
fi

```

---

En este ejemplo, buscamos la cadena más larga que se encuentra dentro de un archivo. Cuando se le asignan uno o más nombres de archivo en la línea de comandos, este programa utiliza el programa strings (que se incluye en el paquete binutils de GNU) para generar una lista de "palabras" de texto legibles en cada archivo. El bucle for procesa cada palabra a su vez y determina si la palabra actual es la más larga encontrada hasta ahora. Cuando concluye el bucle, se muestra la palabra más larga.

Si se omite la parte opcional *en* palabras del comando for, for procesa de forma predeterminada los parámetros posicionales. Modificaremos nuestro script de palabras más largas para usar este método:

```

#!/bin/bash

# longest-word2 : encuentra la cadena más

larga en un archivo para i; hacer
    si [[ -r $i ]]; entonces
        max_word=
        max_len=0
        para j en $(cadenas $i); hacer
            len=$(echo $j | wc -c)
            si ( len > max_len ); entonces
                max_len=$len
                max_word=$j
            Fi
        hecho
        echo "$i: '$max_palabra' ($max_len caracteres)"
    Fi
hecho

```

---

Como podemos ver, hemos cambiado el bucle más externo para usar en lugar de while. Debido a que omitimos la lista de palabras en el comando for, se utilizan los parámetros positivos en su lugar. Dentro del bucle, las instancias anteriores de la variable *i* se han cambiado a la variable *j*. También se ha eliminado el uso de turnos.

### ¿POR QUÉ YO?

Es posible que haya notado que la variable *i* se eligió para cada uno de los ejemplos de bucles for anteriores. ¿Por qué? En realidad, no hay una razón específica, aparte de la tradición. La variable utilizada con for puede ser cualquier variable válida, pero *i* es la más común, seguida de *j* y *k*.

La base de esta tradición proviene del lenguaje de programación Fortran. En Fortran, las variables no declaradas que comienzan con las letras *I*, *J*, *K*, *L* y *M* se escriben automáticamente como números enteros, mientras que las variables que comienzan con cualquier otra letra se escriben como reales (números con fracciones decimales). Este comportamiento llevó a los

para usar las variables I, J y K para las variables de bucle, ya que era menos trabajo usarlas cuando se necesitaba una variable temporal (como a menudo lo era una variable de bucle).

También condujo a la siguiente ocurrencia basada en Fortran: "DIOS es real, a menos que se declare entero".

## para: Formulario de Lenguaje C

Las versiones recientes de bash han agregado una segunda forma de sintaxis de comando for, una que se asemeja a la forma que se encuentra en el lenguaje de programación C. Muchos otros idiomas también admiten esta forma.

```
para (( expresión1; expresión2; expresión3 )); hacer
    Comandos
hecho
```

donde *expression1*, *expression2* y *expression3* son expresiones aritméticas y *commands* son los comandos que se van a realizar durante cada iteración del bucle.

En términos de comportamiento, esta forma es equivalente al siguiente constructo:

```
(( expresión1 ))
while (( expresión2 )); hacer
    Comandos
    (( expresión3 ))
hecho
```

*Expression1* se usa para inicializar las condiciones del bucle, *Expression2* se usa para determinar cuándo finaliza el bucle y *Expression3* se lleva a cabo al final de cada iteración del bucle.

Esta es una aplicación típica:

---

```
#!/bin/bash

# simple_counter : demostración del estilo C para el
comando (( i = 0; i<5; i = i + 1 )); hacer
    Eco $i
hecho
```

---

Cuando se ejecuta, produce la siguiente salida:

---

```
[me@linuxbox ~]$ simple_counter
0
1
2
3
4
```

---

En este ejemplo, *expression1* inicializa la variable *i* con el valor de 0, *expression2* permite que el bucle continúe siempre que el valor de *i* sea menor que 5 y *expression3* incrementa el valor de *i* en 1 cada vez que se repite el bucle.

La forma de lenguaje C de for es útil siempre que se necesite una secuencia numérica . Veremos varias aplicaciones de esto en los próximos dos capítulos.

## Nota final

Con nuestro conocimiento del comando for, ahora aplicaremos las mejoras finales a nuestro script sys\_info\_page. Actualmente, la función report\_home\_space tiene el siguiente aspecto:

```
report_home_space () {
    if [[ $id -u ]];
        entonces gato <<
        _EOF_
            <H2>Utilización del espacio en el hogar (todos los
            usuarios)</H2>
            <PRE>$du -sh /home/*</PRE>
            _EF_
    más
        gato << _EOF_
            <H2>Utilización del espacio en el hogar ($USER)</H2>
            <PRE>$du -sh $HOME</PRE>
            _EF_
    Devo
    lució
    n de
    Fi
}
```

A continuación, lo reescribiremos para proporcionar más detalles sobre el directorio principal de cada usuario e incluiremos el número total de archivos y subdirectorios en cada uno:

```
report_home_space () {

    local format="%8s%10s%10s\n"
    Local I dir_list total_files total_dirs total_size user_name

    if [[ $id -u ]]; then
        dir_list=/home/*
        user_name="Todos los
        usuarios"
    más
        dir_list=$HOME
        user_name=$USER
    Fi

    echo "<H2>Home Space Utilization ($user_name)</H2>

    for i in $dir_list; do

        total_files=$(encontrar $i tipo f | 
        wc -l) total_dirs=$(encontrar $i -
        tipo d | wc -l) total_size=$(du -sh $i |
        cut -f 1) echo "<H3>$i</H3>
        echo "<PRE>
        printf "$format" "Dirs" "Archivos"
        "Tamaño" printf "$format" "----" "--
```

```
---" " -----"  
printf "$format" $total_dirs $total_archivos  
$total_tamaño echo "</PRE>"  
Hecho  
Volver  
}
```

---

Esta reescritura aplica gran parte de lo que hemos aprendido hasta ahora. Todavía probamos para el superusuario, pero en lugar de realizar el conjunto completo de acciones como parte del if, establecemos algunas variables utilizadas más adelante en un bucle for. Hemos añadido varias variables locales a la función y hemos hecho uso de printf para formatear parte de la salida.

# 34

## CADERAS Y NÚMEROS

Los programas informáticos tienen que ver con trabajar con datos. En capítulos anteriores, nos hemos centrado en el procesamiento de datos a nivel de archivo. Sin embargo, muchos problemas de programación deben resolverse utilizando unidades de datos más pequeñas, como cadenas y números.

En este capítulo, veremos varias características de shell que se utilizan para manipular cadenas y números. El shell proporciona una variedad de expansiones de parámetros que realizan operaciones de cadena. Además de la expansión aritmética (de la que hablamos en el capítulo 7), existe un programa de línea de comandos común llamado `bc`, que realiza matemáticas de nivel superior.

### Expansión de parámetros

Aunque la expansión de parámetros surgió en el Capítulo 7, no la cubrimos en detalle porque la mayoría de las expansiones de parámetros se usan en scripts en lugar de en la línea de comandos. Ya hemos trabajado con algunas formas de expansión de parámetros; Por ejemplo, las variables de shell. El caparazón proporciona muchos más.

## Parámetros básicos

La forma más simple de expansión de parámetros se refleja en el uso ordinario de variables. Por ejemplo, \$a, cuando se expande, se convierte en lo que contiene la variable a . Los parámetros simples también pueden estar entre llaves, como \${a}. Esto no tiene ningún efecto en la expansión, pero es necesario si la variable está adyacente a otro texto, lo que puede confundir al shell. En este ejemplo, intentamos crear un nombre de archivo anexando la cadena \_file al contenido de la variable a.

---

```
[me@linuxbox ~]$ a="foo"  
[me@linuxbox ~]$ echo "$a_archivo"
```

---

Si realizamos esta secuencia, el resultado será nada, porque el shell intentará expandir una variable llamada a\_file en lugar de a. Este problema se puede resolver agregando llaves:

---

```
[me@linuxbox ~]$ echo "${a}_file"  
foo_file
```

---

También hemos visto que se puede acceder a parámetros posicionales mayores que 9 rodeando el número entre llaves. Por ejemplo, para acceder al parámetro posicional 11, podemos hacer esto: \${11}.

## Expansiones para gestionar variables vacías

Varias expansiones de parámetros se ocupan de variables inexistentes y vacías. Estas expansiones son útiles para manejar los parámetros posicionales que faltan y asignar valores predeterminados a los parámetros. Esta es una de esas expansiones:

`${parámetro:-palabra}`

Si el *parámetro* no está establecido (es decir, no existe) o está vacío, esta expansión da como resultado el valor de *word*. Si el *parámetro* no está vacío, la expansión da como resultado el valor del *parámetro*.

---

```
[me@linuxbox ~]$ foo=  
[me@linuxbox ~]$ echo ${foo:-"Sustituir valor si no se establece"}  
valor de sustitución si no  
está establecido  
[me@linuxbox ~]$ echo $foo  
  
[me@linuxbox ~]$ foo=bar  
[me@linuxbox ~]$ echo ${foo:-"Sustituir valor si no se establece"}  
barra  
[me@linuxbox ~]$ echo $foo  
barra
```

---

Aquí hay otra expansión, en la que usamos el signo igual en lugar de un guión:

`${parámetro:=palabra}`

Si el *parámetro* no está establecido o está vacío, esta expansión da como resultado el valor de *word*. Además, el valor de *word* se asigna al *parámetro*. Si el *parámetro* no está vacío, la expansión da como resultado el valor del *parámetro*.

---

```
[me@linuxbox ~]$ foo=
[me@linuxbox ~]$ echo ${foo:="valor predeterminado si no se establece"}
Valor por defecto si no se
establece [me@linuxbox ~]$
echo $foo valor por defecto
si se desestablece
[me@linuxbox ~]$ foo=bar
[me@linuxbox ~]$ echo ${foo:="valor predeterminado si no se establece"}
barra
[me@linuxbox ~]$ echo $foo
barra
```

---

**Nota:** Los parámetros posicionales y otros parámetros especiales no se pueden asignar de esta manera.

Aquí usamos un signo de interrogación:

```
 ${parámetro:?palabra}
```

Si el *parámetro* no está establecido o está vacío, esta expansión hace que el script se cierre con un error y el contenido de *Word* se envíe al error estándar. Si el *parámetro* no está vacío, la expansión da como resultado el valor del *parámetro*.

---

```
[me@linuxbox ~]$ foo=
[me@linuxbox ~]$ echo ${foo:??" el parámetro está vacío"}
bash: foo: el parámetro está
vacío [me@linuxbox ~]$ echo
$?
1
[me@linuxbox ~]$ foo=bar
[me@linuxbox ~]$ echo ${foo:??" el parámetro está vacío"}
barra
[me@linuxbox ~]$ echo $?
0
```

---

Aquí usamos un signo más:

```
 ${parámetro:+palabra}
```

Si el *parámetro* no está establecido o está vacío, la expansión no da como resultado nada. Si *parámetro* no está vacío, el valor de *la palabra* se sustituye por el *parámetro*; sin embargo, el valor del *parámetro* no cambia.

---

```
[me@linuxbox ~]$ foo=
[me@linuxbox ~]$ echo ${foo:+"Sustituir valor si se establece"}

[me@linuxbox ~]$ foo=bar
[me@linuxbox ~]$ echo ${foo:+"Sustituir valor si se establece"}
Valor de sustitución si se establece
```

---

### *Expansiones que devuelven nombres de variables*

El shell tiene la capacidad de devolver los nombres de las variables. Esta característica se utiliza en algunas situaciones bastante exóticas.

```
 ${!prefijo*}  
 ${!prefijo@}
```

Esta expansión devuelve los nombres de las variables existentes con nombres que comienzan con *prefijo*. De acuerdo con la documentación de bash, ambas formas de expansión funcionan de manera idéntica. Aquí, enumeraremos todas las variables en el entorno con nombres que comienzan con BASH:

---

```
[me@linuxbox ~]$ echo ${! BASH*}  
BASH BASH_ARC BASH_ARGV BASH_COMMAND BASH_COMPLETION BASH_COMPLETION_DIR BASH_LINENO  
BASH_SOURCE BASH_SUBSHELL BASH_VERSINFO BASH_VERSION
```

---

### Operaciones de cadena

Hay un gran conjunto de expansiones que se pueden usar para operar en cadenas. Muchas de estas expansiones son especialmente adecuadas para operaciones en nombres de ruta. La expansión

```
 ${#parámetro}
```

se expande en la longitud de la cadena contenida por el *parámetro*. Normalmente, el *parámetro* es una cadena; sin embargo, si el *parámetro* es @ o \*, la expansión da como resultado el número de parámetros posicionales.

---

```
[me@linuxbox ~]$ foo="Esta cadena es larga."  
[me@linuxbox ~]$ echo "'$foo' tiene ${#foo} caracteres."  
'Esta cadena es larga.' tiene 20 caracteres.
```

---

```
 ${parámetro:desplazamiento}  
 ${parámetro:desplazamiento:longitud}
```

Esta expansión se utiliza para extraer una parte de la cadena contenida en el *parámetro*. La extracción comienza en *los caracteres de desplazamiento desde el principio de la cadena* y continúa hasta el final de la cadena, a menos que se especifique la *longitud*.

---

```
[me@linuxbox ~]$ foo="Esta cadena es larga."  
[me@linuxbox ~]$ echo ${foo:5}  
La cadena es larga.  
[me@linuxbox ~]$ echo ${foo:5:6}  
cuerda
```

---

Si el valor de *offset* es negativo, se considera que comienza desde el final de la cadena en lugar de desde el principio. Tenga en cuenta que los valores negativos deben ir precedidos de un espacio para evitar confusiones con la expansión \${parameter:-word}. *length*, si está presente, no debe ser inferior a 0.

Si el *parámetro* es @, el resultado de la expansión son los parámetros posicionales de longitud, comenzando en el *desplazamiento*.

---

```
[me@linuxbox ~]$ foo="Esta cadena es larga."  
[me@linuxbox ~]$ echo ${foo: -5}  
largo.  
[me@linuxbox ~]$ echo ${foo: -5:2}  
lo
```

---

```
 ${parámetro#patrón}
 ${parámetro##patrón}
```

Estas expansiones eliminan una parte inicial de la cadena contenida en el *parámetro* definido por *pattern*. *pattern* es un patrón de caracteres comodín como los que se usan en la expansión de nombres de ruta. La diferencia entre las dos formas es que la forma `#` elimina la coincidencia más corta, mientras que la forma `##` elimina la coincidencia más larga.

---

```
[me@linuxbox ~]$ foo=file.txt.zip
[me@linuxbox ~]$ echo ${foo#*.}
txt.zip
[me@linuxbox ~]$ echo ${foo##*.}
cremallera
```

---

```
 ${parámetro%patrón}
 ${parámetro%%patrón}
```

Estas expansiones son las mismas que las expansiones `#` y `##` anteriores, excepto que eliminan el texto del final de la cadena contenida en el *parámetro* en lugar del principio.

---

```
[me@linuxbox ~]$ foo=file.txt.zip
[me@linuxbox ~]$ echo ${foo%.}
file.txt
[me@linuxbox ~]$ echo ${foo%%.}
archivo
```

---

```
 ${parámetro/patrón/cadena}
 ${patrón de parámetro/cadena}
 ${parámetro/#patrón/cadena}
 ${parámetro/%patrón/cadena}
```

Esta expansión realiza una búsqueda y reemplazo en el contenido del *parámetro*. Si se encuentra texto que coincide con el patrón de comodín, se reemplaza por el contenido de la *cadena*. En la forma normal, solo se reemplaza la primera aparición del *patrón*. En el formulario, se reemplazan todas las ocurrencias. La forma `/#` requiere que la coincidencia se produzca al principio de la cadena, y la forma `%` requiere que la coincidencia se produzca al final de la cadena. `/string` se puede omitir, lo que hace que se elimine el texto que coincide con el *patrón*.

---

```
[me@linuxbox ~]$ foo=JPG.JPG
[me@linuxbox ~]$ echo ${foo/JPG/jpg}
jpg.JPG
[me@linuxbox ~]$ echo ${foo//JPG/jpg}
jpg.jpg
[me@linuxbox ~]$ echo ${foo/#JPG/jpg}
jpg.JPG
[me@linuxbox ~]$ echo ${foo/%JPG/jpg}
JPG.jpg
```

---

Es bueno conocer la expansión de parámetros. Las expansiones de manipulación de cadenas se pueden usar como sustitutos de otros comandos comunes, como `sed` y `cut`. Las expansiones mejoran la eficiencia de los scripts al eliminar el uso de programas externos. A modo de ejemplo, modificaremos el programa de palabras más largas discutido en el

capítulo anterior para usar la expansión de parámetros

`${#j}` en lugar de la sustitución de comandos `$(echo $j | wc -c)` y su subshell resultante, así:

---

```
#!/bin/bash

# longest-word3 : encuentra la cadena más

larga en un archivo para i; hacer
    si [[ -r $i ]]; entonces
        max_word=
        max_len=
        para j en $(cadenas $i); hacer
            len=${#j}
            si ( len > max_len )); entonces
                max_len=$len
                max_word=$j
            Fi
        hecho
        echo "$i: '$max_palabra' ($max_len caracteres)"
    Ca
    mbi
hecho o de
fi
```

---

A continuación, compararemos la eficiencia de las dos versiones utilizando el método

Comando de tiempo :

---

```
[me@linuxbox ~]$ la palabra más larga2 dirlist/usr-bin.txt
dirlist/usr-bin.txt: 'scrollkeeper-get-extended-content-list' (38 caracteres)

real      0m3.618s
usuario   0m1.544s
sys       0m1.768s
[me@linuxbox ~]$ tiempo-palabra-más larga3 dirlist/usr-bin.txt
dirlist/usr-bin.txt: 'scrollkeeper-get-extended-content-list' (38 caracteres)

real      0m0.060s
usuario   0m0.056s
sys       0m0.008s
```

---

La versión original del script tarda 3,618 segundos en escanear el archivo de texto, mientras que la nueva versión, que utiliza la expansión de parámetros, solo tarda 0,06 segundos, una mejora muy significativa.

## Evaluación y expansión aritmética

Vimos la expansión aritmética en el capítulo 7. Se utiliza para realizar diversas operaciones aritméticas con números enteros. Su forma básica es

`$((expresión))`

donde `expresión` es una expresión aritmética válida.

Esto está relacionado con el comando compuesto `(( ))` utilizado para la evaluación aritmética (pruebas de verdad) que encontramos en el Capítulo 27.

En capítulos anteriores, vimos algunos de los tipos comunes de

expresiones y operadores. Aquí, veremos una lista más completa.

## Bases numéricas

En el capítulo 9, echamos un vistazo a los números octales (base 8) y hexadecimales (base 16). En expresiones aritméticas, el shell admite constantes enteras en cualquier base. La Tabla 34-1 muestra las notaciones utilizadas para especificar las bases.

Tabla 34-1: Especificación de diferentes bases numéricas

Notación	Descripción
Número	De forma predeterminada, los números sin ninguna notación se tratan como enteros decimales (base 10).
0número	En las expresiones aritméticas, los números con un cero a la izquierda se consideran octales.
0xnumber	Notación hexadecimal
base#número	número está en base.

Algunos ejemplos:

```
[me@linuxbox ~]$ echo $((0xff))
255
[me@linuxbox ~]$ echo $((2#11111111))
255
```

En estos ejemplos, imprimimos el valor del número hexadecimal ff (el número más grande de dos dígitos) y el número binario más grande de ocho dígitos (base 2).

## Operadores unarios

Hay dos operadores unarios, el + y el -, que se utilizan para indicar si un número es positivo o negativo, respectivamente.

## Aritmética simple

Los operadores aritméticos ordinarios se enumeran en la Tabla 34-2.

Tabla 34-2: Operadores aritméticos

Operador	Descripción
+	Adición
-	Sustracción
*	Multiplicación
/	División de enteros
**	Exponenciación
%	Módulo (resto)

La mayoría de ellos se explican por sí mismos, pero la división de enteros y el módulo requieren una mayor discusión.

Dado que la aritmética de la cáscara opera sólo con números enteros, los resultados de la división son siempre números enteros:

---

```
[me@linuxbox ~]$ echo $(( 5 / 2 ))
2
```

---

Esto hace que la determinación de un residuo en una operación de división sea más importante:

---

```
[me@linuxbox ~]$ echo $(( 5 % 2 ))
1
```

---

Usando los operadores de división y módulo, podemos determinar que 5 dividido por 2 da como resultado 2, con un resto de 1.

Calcular el resto es útil en bucles. Permite realizar una operación a intervalos especificados durante la ejecución del bucle. En el siguiente ejemplo, mostramos una línea de números, resaltando cada múltiplo de 5:

---

```
#!/bin/bash

# módulo : demostrar el operador módulo para

((i = 0; i <= 20; i = i + 1)); hacer
    resto=$((i % 5))
    if (( resto == 0 )); A
        continuación, printf
            "<%d> " $i
    else
        printf "%d " $i
    fi
hecho
printf "\n"
```

---

Cuando se ejecuta, los resultados se ven así:

---

```
[me@linuxbox ~]$ módulo
<0> 1 2 3 4 <5> 6 7 8 9 <10> 11 12 13 14 <15> 16 17 18 19 <20>
```

---

## Asignación

Aunque sus usos pueden no ser evidentes de inmediato, las expresiones aritméticas pueden realizar la asignación. Hemos realizado tareas muchas veces, aunque en un contexto diferente. Cada vez que le damos un valor a una variable, estamos realizando una asignación. También podemos hacerlo dentro de expresiones aritméticas:

---

```
[me@linuxbox ~]$ foo=
[me@linuxbox ~]$ echo $foo
```

```
[me@linuxbox ~]$ si (( foo = 5 )); luego repite "Es verdad"; Fi
Es verdad.
[me@linuxbox ~]$ echo $foo
5
```

---

En el ejemplo anterior, primero asignamos un valor vacío a la variable `foo` y verificamos que realmente esté vacío. A continuación, realizamos un `if` con el comando com- compuesto (`(( foo = 5 ))`). Este proceso hace dos cosas interesantes:

- (1) asigna el valor de 5 a la variable `foo`, y (2) se evalúa como verdadero porque la asignación se realizó correctamente.

**Nota:** Es importante recordar el significado exacto de `=` en la expresión anterior. Un solo `=` realiza la tarea: `foo = 5` dice, "Haz que `foo` sea igual a 5." Un doble `==` evalúa la equivalencia: `foo == 5` dice: "¿`foo` es igual a 5?" Esto puede ser muy confuso porque el comando de prueba acepta un solo `=` para la equivalencia de cadenas. Esta es otra razón más para usar los comandos compuestos más modernos `[[ ]]` y `(( ))` en lugar de `test`.

Además de `=`, el shell proporciona notaciones que realizan algunas tareas muy útiles, como se muestra en la Tabla 34-3.

**Tabla 34-3: Operadores de asignación**

Notación	Descripción
<code>parámetro = valueSimple</code>	asignación. Asigna <code>valor</code> Para <code>parámetro</code> .
<code>parámetro += valor</code>	Adición. Equivalente a <code>parámetro = parámetro + valor</code> .
<code>parámetro -= valor</code>	Sustracción. Equivalente a <code>parámetro = parámetro - valor</code> .
<code>parámetro *= valor</code>	Multiplicación. Equivalente a <code>parámetro = parámetro × valor</code> .
<code>parámetro /= valor</code>	División de enteros. Equivalente a <code>parámetro = parámetro ÷ valor</code> .
<code>parámetro %= valueModule.</code>	Equivalente a <code>parámetro = parámetro % valor</code> .
<code>Parámetro++</code>	Post-incremento variable. Equivalente a <code>parámetro = parámetro + 1</code> . (Sin embargo, consulte la siguiente discusión).
<code>parámetro--</code>	Post-decremento variable. Equivalente a <code>parámetro = parámetro - 1</code> .
<code>++parámetro</code>	Pre-incremento variable. Equivalente a <code>parámetro = parámetro + 1</code> .
<code>--parámetro</code>	Pre-decremento variable. Equivalente a <code>parámetro = parámetro - 1</code> .

Estos operadores de asignación proporcionan una abreviatura conveniente para muchas tareas aritméticas comunes. De especial interés son los operadores de incremento `(++)` y decremento `(--)`, que aumentan o disminuyen el valor de sus parámetros en 1. Este estilo de notación se toma de la programación en C

y ha sido incorporado por varios otros lenguajes de programación, incluyendo bash.

Los operadores pueden aparecer al principio o al final de un parámetro. Si bien ambos incrementan o disminuyen el parámetro en 1, las dos ubicaciones tienen una diferencia sutil. Si se coloca al principio del parámetro, el parámetro se incrementa (o disminuye) antes de que se devuelva el parámetro. Si se coloca después, la operación se realiza *después* de que se devuelva el parámetro. Esto es bastante extraño, pero es el comportamiento previsto. Aquí hay una demostración:

---

```
[me@linuxbox ~]$ foo=1
[me@linuxbox ~]$ echo $((foo++))
1
[me@linuxbox ~]$ echo $foo
2
```

---

Si asignamos el valor de 1 a la variable `foo` y luego lo incrementamos con el operador `++` colocado después del nombre del parámetro, `foo` se devuelve con el valor de 1. Sin embargo, si observamos el valor de la variable por segunda vez, vemos el valor incrementado. Si colocamos el operador `++` delante del parámetro, obtenemos este comportamiento más esperado:

---

```
[me@linuxbox ~]$ foo=1
[me@linuxbox ~]$ echo $((++foo))
2
[me@linuxbox ~]$ echo $foo
2
```

---

Para la mayoría de las aplicaciones de shell, el prefijo del operador será lo más útil.

Los operadores `++` y `--` se utilizan a menudo junto con bucles. Haremos algunas mejoras a nuestro script de módulo para ajustarlo un poco:

---

```
#!/bin/bash

# modulo2 : demostrar el operador módulo

para ((i = 0; i <= 20; ++i )); hacer
    si (((i % 5) == 0 )); entonces
        printf "<%d> " $i
    else
        printf "%d " $i
    fi
hecho
printf "\n"
```

---

### **Operaciones de bits**

Una clase de operadores manipula los números de una manera inusual. Estos operadores funcionan a nivel de bits. Se utilizan para ciertos tipos de tareas de bajo nivel, que a menudo implican la configuración o lectura de indicadores de bits. En la tabla 34-4 se enumeran los operadores de bits.

**Tabla 34-4: Operadores de bits**

Operador	Descripción
<code>~</code>	Negación bit a bit. Niega todos los bits de un número.
<code>&lt;&lt;</code>	Desplazamiento bit a bit a la izquierda. Desplaza todos los bits de un número hacia la izquierda.
<code>&gt;&gt;</code>	Desplazamiento bit a bit a la derecha. Desplaza todos los bits de un número hacia la derecha.
<code>&amp;</code>	Realice una operación AND bit a bit en todos los bits de dos números.
<code> </code>	O bit a bit. Realice una operación OR en todos los bits de dos números.
<code>^</code>	XOR bit a bit. Realice una operación OR exclusiva en todos los bits de dos números.

Tenga en cuenta que también hay operadores de asignación correspondientes (por ejemplo, `<=`) para todas las negaciones excepto bits.

Aquí demostraremos la producción de una lista de potencias de 2, usando el operador de desplazamiento bit a bit a la izquierda:

```
[me@linuxbox ~]$ para ((i=0; i<8;++i)); hacer eco $((1<<i)); hecho
1
2
4
8
16
32
64
128
```

### Lógica

Como descubrimos en el capítulo 27, el comando compuesto (`(( ))`) admite una variedad de operadores de comparación. Hay algunos más que se pueden usar para evaluar la lógica. La tabla 34-5 muestra la lista completa.

**Tabla 34-5: Operadores de comparación**

Operador	Descripción
<code>&lt;=</code>	Menor o igual que
<code>&gt;=</code>	Mayor o igual que
<code>&lt;</code>	Menos que
<code>&gt;</code>	Mayor que
<code>==</code>	Igual a

*(continuación)*

Cuadro 34-5 (*continuación* )

Operador	Descripción
<code>!=</code>	No es igual a
<code>&amp;&amp;</code>	Lógico Y
<code>  </code>	Quirófano lógico
<code>expr1?expr2:expr3</code>	Operador de comparación (ternario). Expresión If <i>expr1</i> se evalúa como distinto de cero (aritmético verdadero) y luego <i>expr2</i> , o bien <i>expr3</i> .

Cuando se utilizan para operaciones lógicas, las expresiones siguen las reglas de la lógica aritmética; es decir, las expresiones que se evalúan como 0 se consideran falsas, mientras que las expresiones distintas de cero se consideran verdaderas. El comando compuesto (( )) mapea los resultados en los códigos de salida normales del shell:

```
[me@linuxbox ~]$ if ((1)); then echo "verdadero"; else echo "falso"; fi  
verdadero  
[me@linuxbox ~]$ if ((0)); then echo "true"; else echo "false"; fi  
falso
```

El más extraño de los operadores lógicos es el *operador ternario*. Este operador (que se basa en el del lenguaje de programación C) realiza una prueba lógica independiente. Se puede utilizar como una especie de declaración if/then/else. Actúa sobre tres expresiones aritméticas (las cadenas no funcionarán), y si la primera expresión es verdadera (o distinta de cero), se realiza la segunda expresión. De lo contrario, se realiza la tercera expresión. Podemos probar esto en la línea de comandos.

```
[me@linuxbox ~]$ a=0  
[me@linuxbox ~]$ ((a<1?++a:--a))  
[me@linuxbox ~]$ echo $a  
1  
[me@linuxbox ~]$ ((a<1?++a:--a))  
  
[me@linuxbox ~]$ echo $a  
0
```

Aquí vemos un operador ternario en acción. En este ejemplo se implementa un botón de alternancia. Cada vez que se realiza el operador, el valor de la variable a cambia de 0 a 1 o viceversa.

Tenga en cuenta que realizar la asignación dentro de las expresiones no es sencillo. Cuando se intenta esto, bash declarará un error:

```
[me@linuxbox ~]$ a=0  
[me@linuxbox ~]$ ((a<1?a+=1:a-=1))  
bash: ((: a<1?a+=1:a-=1: intento de asignación a no variable (el token de  
error es "-=1")
```

Este problema se puede mitigar rodeando la expresión de la asignación entre paréntesis:

```
[me@linuxbox ~]$ ((a<1?( a+=1):(a-=1)))
```

A continuación, vemos un ejemplo más completo del uso de operadores aritméticos en un script que produce una tabla simple de números:

```
#!/bin/bash

# arith-loop: script para demostrar operadores aritméticos

finished=0
a=0
printf "a\t${a**2}\t${a**3}\n"
printf "\t=\t==\t====\n"

hasta ((finalizado)); hacer
    b=$((a**2))
    c=$((a**3))
    printf "%d\t%d\t%d\n" $a $b $c
    ((a<10?++a:(terminado=1)))
hecho
```

En este script, implementamos un bucle until basado en el valor de la propiedad

Variable terminada. Inicialmente, la variable se establece en 0 (aritmética falsa) y continuamos el bucle hasta que se convierte en distinto de cero.

Dentro del bucle, calculamos el cuadrado y el cubo de la contravariante a. Al final del bucle, se evalúa el valor de la variable de contador. Si es menor que 10 (el número máximo de iteraciones), se incrementa en 1, de lo contrario, a la variable finished se le da el valor de 1, lo que hace que finished sea aritméticamente verdadero y, por lo tanto, finalice el bucle. Al ejecutar el script, se obtiene este resultado:

```
[me@linuxbox ~]$ bucle aritmético
un      A**2      A**3
=      ===      ====
0        0        0
1        1        1
2        4        8
3        9       27
4       16       64
5       25      125
6       36      216
7       49      343
8       64      512
9       81      729
10      100     1000
```

## bc: un lenguaje de calculadora de precisión arbitraria

Hemos visto que el shell puede manejar todo tipo de aritmética de enteros, pero ¿qué pasa si necesitamos realizar matemáticas superiores o incluso simplemente usar números de punto flotante? La respuesta es que no

podemos. Al menos no directamente con el caparazón. Para ello,

Necesitamos usar un programa externo. Hay varios enfoques que podemos adoptar. La incorporación de programas Perl o AWK es una solución posible, pero, desafortunadamente, está fuera del alcance de este libro.

Otro enfoque es utilizar un programa de calculadora especializado. Uno de estos programas que se encuentra en la mayoría de los sistemas Linux se llama bc.

El programa bc lee un fichero escrito en su propio lenguaje tipo C y lo ejecuta. Un script bc puede ser un archivo separado o puede leerse desde una entrada estándar. El lenguaje bc admite bastantes características, incluidas variables, bucles y funciones definidas por el programador. No cubriremos bc por completo aquí, solo lo suficiente para probarlo. BC está bien documentado por su página de manual.

Empecemos con un ejemplo sencillo. Escribiremos un script bc para sumar 2 más 2:

---

```
/* Un script bc muy simple */
```

---

```
2 + 2
```

---

La primera línea del script es un comentario. bc utiliza la misma sintaxis para los comentarios que el lenguaje de programación C. Los comentarios, que pueden abarcar varias líneas, comienzan con /\* y terminan con \*/.

### **Usando bc**

Si guardamos el script bc de arriba como *foo.bc*, podemos ejecutarlo de esta manera:

---

```
[me@linuxbox ~]$ bc foo.bc  
BC 1.06.94
```

Derechos de autor 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.

Este es software libre sin ABSOLUTAMENTE GARANTÍA.

Para obtener más información, escriba 'garantía'.

4

---

Si nos fijamos bien, podemos ver el resultado en la parte inferior, después del mensaje de derechos de autor. Este mensaje se puede suprimir con la opción -q (quiet).

BC también se puede utilizar de forma interactiva:

---

```
[me@linuxbox ~]$ bc -q  
2 + 2  
4  
renunciar
```

---

Al usar bc de forma interactiva, simplemente escribimos los cálculos que deseamos realizar y los resultados se muestran inmediatamente. El comando bc quit finaliza la sesión interactiva.

También es posible pasar un script a bc a través de una entrada estándar:

---

```
[me@linuxbox ~]$ bc < foo.bc  
4
```

La capacidad de tomar una entrada estándar significa que podemos usar documentos here, cadenas here y tuberías para pasar scripts. Este es un ejemplo de cadena aquí:

---

```
[me@linuxbox ~]$ bc <<< "2+2"
4
```

---

### *Un script de ejemplo*

Como ejemplo del mundo real, construiremos un script que realiza un cálculo común, los pagos mensuales del préstamo. En el siguiente script, usamos un documento aquí para pasar un script a bc:

---

```
#!/bin/bash

# loan-calc : script para calcular los pagos mensuales del préstamo

PROGNAME=$(basename $0)

uso () {
    gato <<- EOF
    Uso: $PROGNAME MESES DE INTERÉS PRINCIPAL

    Dónde:
        PRINCIPAL es el monto del préstamo. El INTERÉS
        es la TAE como un número (7% = 0,07). MESES
        es la duración del plazo del préstamo .

    EF
}

si (($# != 3)); A
    continuació
    n, el uso
    Salida 1
fi

Capital = $1
interés = $2 meses
= $3

bc <<- EOF
    Escala = 10
    i = $interest / 12
    p = $principal
    n = $months
    a = p * ((i * ((1 + i) ^ n)) / (((1 + i) ^ n) - 1))
    print a, "\n"
EF
```

---

Cuando se ejecuta, los resultados se ven así:

---

```
[me@linuxbox ~]$ loan-calc 135000 0.0775 180
1270.7222490000
```

---

Este ejemplo calcula el pago mensual de un préstamo de \$135,000 a 7.75% APR durante 180 meses (15 años). Fíjate en la precisión de la

respuesta. Esto está determinado por el valor dado a la variable de escala especial en el bc



Guión. Una descripción completa del lenguaje de scripting bc se proporciona en la página del comando man bc. Si bien su notación matemática es ligeramente diferente a la de la concha (bc se parece más a C), la mayor parte de ella será bastante familiar, según lo que hemos aprendido hasta ahora.

## Nota final

En este capítulo, hemos aprendido sobre muchas de las pequeñas cosas que se pueden usar para hacer el "trabajo real" en los guiones. A medida que crezca nuestra experiencia con las secuencias de comandos, la capacidad de manipular eficazmente cadenas y números resultará extremadamente valiosa. Nuestro script loan-calc demuestra que incluso los scripts simples pueden hacer cosas realmente útiles.

## Crédito Extra

Si bien la funcionalidad básica del script loan-calc está en su lugar, el script está lejos de estar completo. Para obtener crédito adicional, intente mejorar el script loan-calc con las siguientes características:

- Verificación completa de los argumentos de la línea de comandos
- Una opción de línea de comandos para implementar un modo "interactivo" que solicitará al usuario que ingrese el capital, la tasa de interés y el plazo del préstamo
- Un mejor formato para la salida

# 35

## MATRICES

En el último capítulo, vimos cómo el shell puede manipular cadenas y números. Los tipos de datos que hemos visto hasta ahora se conocen en los círculos de informática como *variables escalares*, es decir, variables que contienen un solo valor.

En este capítulo, veremos otro tipo de estructura de datos llamada *matriz*, que contiene múltiples valores. Las matrices son una característica de prácticamente todos los lenguajes de programación. El caparazón también las sostiene, aunque de una manera bastante limitada. Aun así, pueden ser muy útiles para resolver problemas de programación.

### ¿Qué son las matrices?

Las matrices son variables que contienen más de un valor a la vez. Las matrices se organizan como una tabla. Consideremos una hoja de cálculo como ejemplo. Una hoja de cálculo actúa como una *matriz bidimensional*. Tiene filas y columnas, y una celda individual en la hoja de cálculo se puede ubicar de acuerdo con su dirección de fila y columna. Una matriz se comporta de la misma manera. Una matriz tiene celdas, que

se denominan *elementos*, y cada elemento contiene datos. Se accede a un elemento de matriz individual utilizando una dirección llamada *índice* o *subíndice*.

La mayoría de los lenguajes de programación admiten *matrices multidimensionales*. Una hoja de cálculo es un ejemplo de una matriz multidimensional con dos dimensiones, ancho y alto. Muchos lenguajes admiten matrices con un número arbitrario de dimensiones, aunque las matrices bidimensionales y tridimensionales son probablemente las más utilizadas.

Las matrices en bash están limitadas a una sola dimensión. Podemos pensar en ellos como una hoja de cálculo con una sola columna. Incluso con esta limitación, hay muchas aplicaciones para ellos. El soporte de matrices apareció por primera vez en la versión 2 de bash. El programa de shell original de Unix, sh, no soportaba matrices en absoluto.

## Creación de una matriz

Las variables de matriz se nombran al igual que otras variables bash y se crean automáticamente cuando se accede a ellas. He aquí un ejemplo:

---

```
[me@linuxbox ~]$ a[1]=foo  
[me@linuxbox ~]$ echo ${a[1]}  
foo
```

---

Aquí vemos un ejemplo de la asignación y el acceso de un elemento de matriz. Con el primer comando, al elemento 1 de la matriz a se le asigna el valor foo. El segundo comando muestra el valor almacenado del elemento 1. El uso de llaves en el segundo comando es necesario para evitar que el shell intente expandir el nombre de ruta en el nombre del elemento de la matriz.

También se puede crear una matriz con el comando declare:

---

```
[me@linuxbox ~]$ declare -a a
```

---

Usando la opción -a, este ejemplo de declare crea la matriz a.

## Asignación de valores a una matriz

Los valores se pueden asignar de dos maneras. Los valores individuales se pueden asignar mediante la siguiente sintaxis:

*nombre[subíndice]*=valor

donde *name* es el nombre de la matriz y *subíndice* es un número entero (o expresión aritmética) mayor o igual que 0. Tenga en cuenta que el primer elemento de una matriz es el subíndice 0, no 1. *value* es una cadena o un número entero asignado al elemento de la matriz.

Se pueden asignar varios valores mediante la siguiente sintaxis:

*nombre=(valor1 valor2 ...)*

donde *name* es el nombre de la matriz y *value1 value2 ...* son valores asignados

secuencialmente a los elementos de la matriz, empezando por el elemento 0. Por ejemplo

Si quisieramos asignar días abreviados de la semana a la matriz Days, podríamos hacer lo siguiente:

---

```
[me@linuxbox ~]$ días=(Dom Lun Mar Mié Jue Vie Sáb)
```

---

También es posible asignar valores a un elemento específico especificando un subíndice para cada valor:

---

```
[me@linuxbox ~]$ días=([0]=Dom [1]=Lunes [2]=Martes [3]=Miércoles [4]=Jue [5]=Viernes  
[6]=Sábado)
```

---

## Acceso a los elementos de la matriz

Entonces, ¿para qué sirven las matrices? Al igual que muchas tareas de gestión de datos se pueden realizar con un programa de hoja de cálculo, muchas tareas de programación se pueden realizar con matrices.

Consideremos un ejemplo sencillo de recopilación y presentación de datos. Construiremos un script que examine los tiempos de modificación de los archivos en un directorio específico. A partir de estos datos, nuestro script generará una tabla que muestra a qué hora del día se modificaron por última vez los archivos. Un script de este tipo podría utilizarse para determinar cuándo un sistema está más activo. Este script, llamado horas, produce este resultado:

---

```
[me@linuxbox ~]$ horas .  
Hora Archi Hora Arch  
---- vos ---- ivos  
-----  
00 0 12 11  
01 1 13 7  
02 0 14 1  
03 0 15 7  
04 1 16 6  
05 1 17 5  
06 6 18 4  
07 3 19 4  
08 1 20 1  
09 14 21 0  
10 2 22 0  
11 5 23 0
```

Total de archivos = 80

---

Ejecutamos el programa de horas , especificando el directorio actual como destino. Genera una tabla que muestra, para cada hora del día (0-23), cuántos archivos se modificaron por última vez. El código para producir esto es el siguiente:

---

```
#!/bin/bash  
  
# hours : script para contar archivos por  
# modificación tiempo de uso () {  
#     echo "Uso: $(nombrebbase $0) directorio" >&2  
# }
```

```

# Compruebe que el argumento es
un directorio si [[ ! -d $1 ]];
entonces
    Salida
    de uso
    1
fi

# Inicializar matriz
for i in {0..23}; do hours[i]=0; done

# Recopilar datos
for i in $(stat -c %"$1"/* | cut -c 12-13); do
    j=${i/#0}
    ((++horas[j]))
    ((++conteo))
hecho

# Mostrar datos
echo -e
"Horas\tArchivos\tHora\tArchivos"
echo -e "-----\t-----\t-----\t-----"
para i en {0..11}; do
    j=$((i + 12))
    printf "%02d\t%02d\t%02d\t%02d\n" $i ${hours[i]} $j ${hours[j]}
hecho
printf "\nTotal de archivos = %d\n" $count

```

---

El script consta de una función (uso) y un cuerpo principal con cuatro secciones. En la primera sección, comprobamos que hay un argumento de línea de comandos y que es un directorio. Si no es así, mostramos el mensaje de uso y salimos.

La segunda sección inicializa las horas de la matriz. Para ello, asigna a cada elemento un valor de 0. No hay ningún requisito especial para preparar matrices antes de su uso, pero nuestro script debe asegurarse de que ningún elemento esté vacío. Nótese la interesante forma en que se construye el bucle. Al emplear la expansión de llaves ({0..23}), podemos generar fácilmente una secuencia de palabras para el comando for.

La siguiente sección recopila los datos ejecutando el programa stat en cada archivo del directorio. Usamos el corte para extraer la hora de dos dígitos del resultado. Dentro del bucle, necesitamos eliminar los ceros a la izquierda del campo de la hora, ya que el shell intentará (y finalmente fallará) interpretar los valores del 00 al 09 como números octales (ver Tabla 34-1). A continuación, incrementamos el valor del elemento de la matriz correspondiente a la hora del día. Finalmente, incrementamos un contador (count) para rastrear el número total de archivos en el directorio.

La última sección del script muestra el contenido de la matriz. Primero generamos un par de líneas de encabezado y luego entramos en un bucle que produce dos columnas de salida. Por último, generamos el recuento final de archivos.

## Operaciones de matriz

Hay muchas operaciones de matriz comunes. Cosas como la eliminación de

matrices, la determinación de su tamaño, la clasificación, etc., tienen muchas aplicaciones en las secuencias de comandos.

## **Salida de todo el contenido de una matriz**

Los subíndices \* y @ se pueden utilizar para acceder a todos los elementos de una matriz. Al igual que con los parámetros posicionales, la notación @ es la más útil de las dos. Aquí hay una demostración:

---

```
[me@linuxbox ~]$ animals=("un perro" "un gato" "un pez")
[me@linuxbox ~]$ for i in ${animals[*]}; do echo $i; done
a
perr
o,
un
gat
o,
un
pez
[me@linuxbox ~]$ for i in ${animals[@]}; do echo $i; done
un
perr
o,
un
gato
, un
pez
[me@linuxbox ~]$ for i in "${animals[*]}"; do echo $i; done
un perro, un gato, un pez
[me@linuxbox ~]$ for i in "${animals[@]}"; do echo $i; done
un
perro,
un
gato,
un
pez
```

---

Creamos los animales de la matriz y le asignamos tres cadenas de dos palabras. A continuación, ejecutamos cuatro bucles para ver el efecto de la división de palabras en el contenido de la matriz. El comportamiento de las notaciones \${animals[\*]} y \${animals[@]} es idéntico hasta que se entrecorren. La notación \* da como resultado una sola palabra que contiene el contenido de la matriz, mientras que la notación @ da como resultado tres palabras, lo que coincide con el contenido "real" de la matriz.

## **Determinación del número de elementos de la matriz**

Usando la expansión de parámetros, podemos determinar el número de elementos en una matriz de la misma manera que encontrar la longitud de una cadena. He aquí un ejemplo:

---

```
[me@linuxbox ~]$ a[100]=foo
[me@linuxbox ~]$ echo ${#a[@]} # número de elementos de la matriz
1
[me@linuxbox ~]$ echo ${#a[100]} # longitud del elemento 100
3
```

---

Creamos la matriz a y asignamos la cadena foo al elemento 100. A continuación, usamos la expansión de parámetros para examinar la longitud de la matriz, usando la notación @. Por último, nos fijamos en la longitud

del elemento 100, que contiene la cadena `foo`. Es interesante notar que mientras asignamos nuestra cadena al elemento 100, bash reporta solo un elemento en la matriz. Esto difiere del comportamiento de algunos otros lenguajes, en los que los elementos no utilizados de la matriz (elementos 0-99) se inicializarían con valores vacíos y se contaría.

## *Encontrar los subíndices utilizados por una matriz*

Como bash permite que las matrices contengan "brechas" en la asignación de subíndices, a veces es útil determinar qué elementos existen realmente. Esto se puede hacer con una expansión de parámetros utilizando las siguientes formas:

```
${{!matriz[*]}}  
${{!matriz[@]}}
```

donde *array* es el nombre de una variable de matriz. Al igual que las otras expansiones que usan \* y @, la forma @ entre comillas es la más útil, ya que se expande en palabras separadas:

---

```
[me@linuxbox ~]$ foo=(2=a [4]=b [6]=c)  
[me@linuxbox ~]$ for i in "${foo[@]}"; do echo $i; done  
a  
b  
c  
[me@linuxbox ~]$ for i in "${!foo[@]}"; do echo $i; done  
2  
4  
6
```

---

## *Adición de elementos al final de una matriz*

Conocer el número de elementos en una matriz no es útil si necesitamos agregar valores al final de una matriz, ya que los valores devueltos por las notaciones \* y @ no nos indican el índice máximo de la matriz en uso. Afortunadamente, el caparazón nos proporciona una solución. Al usar el operador de asignación +=, podemos agregar automáticamente valores al final de una matriz. Aquí, asignamos tres valores a la matriz *foo* y luego agregamos tres más.

---

```
[me@linuxbox ~]$ foo=(a b c)  
[me@linuxbox ~]$ echo ${foo[@]}  
a b c  
[me@linuxbox ~]$ foo+=(d e f)  
[me@linuxbox ~]$ echo ${foo[@]}  
a b c d e f
```

---

## *Ordenar una matriz*

Al igual que con las hojas de cálculo, a menudo es necesario ordenar los valores en una columna de datos. El shell no tiene una forma directa de hacer esto, pero no es difícil de hacer con un poco de codificación:

---

```
#!/bin/bash  
  
# array-sort : Ordenar un  
  
array a=(f e d c b a)  
echo "Matriz original: ${a[@]}"  
a_sorted=($(para Yo en "${a[@]}"; hacer eco $i; Hecho  
| sort)) echo "Matriz ordenada: ${a_sorted[@]}"
```

---

Cuando se ejecuta, el script produce lo siguiente:

---

```
[me@linuxbox ~]$ array-
sort Matriz original: f e d c
b a Matriz ordenada: a b c d
e f
```

---

El script funciona copiando el contenido de la matriz original (`a`) en una segunda matriz (`a_sorted`) con una pieza complicada de sustitución de comandos. Esta técnica básica se puede usar para realizar muchos tipos de operaciones en la matriz cambiando el diseño de la canalización.

### ***Eliminación de una matriz***

Para eliminar una matriz, utilice el comando `unset`:

---

```
[me@linuxbox ~]$ foo=(a b c d e f)
[me@linuxbox ~]$ echo ${foo[@]}
a b c d e f
[me@linuxbox ~]$ unset foo
[me@linuxbox ~]$ echo ${foo[@]}

[me@linuxbox ~]$
```

---

`unset` también se puede usar para eliminar elementos de una sola matriz:

---

```
[me@linuxbox ~]$ foo=(a b c d e f)
[me@linuxbox ~]$ echo ${foo[@]}
a b c d e f
[me@linuxbox ~]$ unset 'foo[2]'
[me@linuxbox ~]$ echo ${foo[@]}
a b d e f
```

---

En este ejemplo, eliminamos el tercer elemento de la matriz, el subíndice 2. Recuerde, las matrices comienzan con el subíndice 0, ¡no con 1! Tenga en cuenta también que el elemento de la matriz debe estar entre comillas para evitar que el shell realice la expansión del nombre de ruta.

Curiosamente, la asignación de un valor vacío a una matriz no vacía su contenido:

---

```
[me@linuxbox ~]$ foo=(a b c d e f)
[me@linuxbox ~]$ foo=
[me@linuxbox ~]$ echo ${foo[@]}
b c d e f
```

---

Cualquier referencia a una variable de matriz sin un subíndice se refiere al elemento 0 de la matriz:

---

```
[me@linuxbox ~]$ foo=(a b c d e f)
[me@linuxbox ~]$ echo ${foo[@]}
a b c d e f [me@linuxbox
~]$ foo=A
[me@linuxbox ~]$ echo ${foo[@]}
A b c d e f
```

---

## Nota final

Si buscamos la palabra array en la página de manual de *bash*, encontramos muchos casos en los que bash hace uso de variables de array. La mayoría de estos son bastante oscuros, pero pueden proporcionar una utilidad ocasional en algunas circunstancias especiales. De hecho, todo el tema de las matrices está bastante infrautilizado en la programación de shell, en gran parte porque los programas de shell tradicionales de Unix (como sh) carecían de soporte para arrays. Esta falta de popularidad es desafortunada, porque las matrices se usan ampliamente en otros lenguajes de programación y proporcionan una herramienta poderosa para resolver muchos tipos de problemas de programación.

Las matrices y los bucles tienen una afinidad natural y, a menudo, se usan juntos. La siguiente forma de bucle es particularmente adecuada para calcular subíndices de matriz:

```
para ((expr1; expr2; expr3))
```

# 36

## EXÓTICA

En este, el último capítulo de nuestro viaje, veremos algunos detalles y extremos. Si bien es cierto que hemos cubierto mucho terreno en los capítulos anteriores, hay muchas características de bash que no hemos cubierto. La mayoría son bastante oscuros y útiles principalmente para aquellos que integran bash en una distribución de Linux. Sin embargo, hay algunos que, aunque no son de uso común, son útiles para ciertos problemas de programación. Los cubriremos aquí.

### Comandos de grupo y subshells

bash permite agrupar comandos. Esto se puede hacer de dos maneras: con un *comando de grupo* o con un *subshell*. A continuación, se muestran ejemplos de la sintaxis de cada uno.

Comando de grupo:

```
{ comando1; comando2; [comando3; ...] }
```

Subcapa:

```
(comando1; comando2; [comando3;...])
```



Las dos formas difieren en que un comando de grupo rodea sus comandos con llaves y un subshell usa paréntesis. Es importante tener en cuenta que, debido a la forma en que bash implementa los comandos de grupo, las llaves deben estar separadas de los comandos por un espacio y el último comando debe terminar con un punto y coma o una nueva línea antes de la llave de cierre.

## Realización de redireccionamientos

Entonces, ¿para qué sirven los comandos de grupo y los subshells? Si bien tienen una diferencia importante (a la que llegaremos en un momento), ambos se utilizan para administrar la redirección. Consideremos un segmento de script que realiza redirecciones en varios comandos:

---

```
ls -l > output.txt
echo "Listado de foo.txt" >>
output.txt gato foo.txt >> output.txt
```

---

Esto es bastante sencillo: tres comandos con su salida redirigida a un archivo llamado *output.txt*. Usando un comando de grupo, podríamos codificar esto de la siguiente manera:

---

```
{ ls -l; echo "Listado de foo.txt"; gato foo.txt; } > output.txt
```

---

El uso de un subshell es similar:

---

```
(ls -l; echo "Listado de foo.txt"; cat foo.txt) > output.txt
```

---

Con esta técnica, nos hemos ahorrado algo de escritura, pero donde realmente brilla un comando de grupo o subshell es con las tuberías. Al construir una canalización de comandos, a menudo es útil combinar los resultados de varios comandos en un solo flujo. Los comandos de grupo y los subshells lo hacen fácil:

---

```
{ ls -l; echo "Listado de foo.txt"; gato foo.txt; } | LPR
```

---

Aquí hemos combinado la salida de nuestros tres comandos y los hemos canalizado a la entrada de lpr para producir un informe impreso.

## Sustitución de procesos

Si bien tienen un aspecto similar y se pueden usar para combinar flujos para la redirección, existe una diferencia importante entre los comandos de grupo y los subshells. Mientras que un comando de grupo ejecuta todos sus comandos en el shell actual, un subshell (como su nombre indica) ejecuta sus comandos en una copia secundaria del shell actual. Esto significa que el medio ambiente

Se copia y se da a una nueva instancia del shell. Cuando se cierra el subshell, se pierde la copia del entorno, por lo que los cambios realizados en el entorno del subshell (incluida la asignación de variables) también se pierden.

Por lo tanto, en la mayoría de los casos, a menos que un script requiera un subshell, los comandos de grupo son preferibles a los subshells. Los comandos de grupo son más rápidos y requieren menos memoria.

Vimos un ejemplo del problema del entorno de subshell en el Capítulo 28, cuando descubrimos que un comando de lectura en una tubería no funciona como podríamos esperar intuitivamente. En resumen, cuando construimos una tubería como esta:

---

```
echo "foo" | Leer  
Echo $REPLY
```

---

el contenido de la variable REPLY siempre está vacío, porque el comando read se ejecuta en un subshell y su copia de REPLY se destruye cuando termina el subshell.

Dado que los comandos de las canalizaciones siempre se ejecutan en subshells, cualquier comando que asigne variables se encontrará con este problema. Afortunadamente, la cáscara proporciona una forma exótica de expansión llamada sustitución de *procesos* que se puede utilizar para solucionar este problema.

La sustitución de procesos se expresa de dos maneras: para los procesos que producen una salida estándar:

<(list)

o para procesos que toman entrada estándar:

>(list)

donde *list* es una lista de comandos.

Para resolver nuestro problema con la lectura, podemos emplear la sustitución de procesos de la siguiente manera:

---

```
leer < <(echo "foo")  
eco $REPLY
```

---

La sustitución de procesos nos permite tratar la salida de un subshell como un archivo ordinario con fines de redirección. De hecho, dado que es una forma de expansión, podemos examinar su valor real:

---

```
[me@linuxbox ~]$ echo <(echo "foo")  
/dev/fd/63
```

---

Al usar echo para ver el resultado de la expansión, vemos que la salida de la subshell es proporcionada por un archivo llamado /dev/fd/63.

La sustitución de procesos se utiliza a menudo con bucles que contienen lectura. A continuación se muestra un ejemplo de un bucle de lectura que procesa el contenido de una lista de directorios creada por un subshell:

---

```
#!/bin/bash
```

```
# pro-sub : demostración de sustitución de procesos
```

mientras que los enlaces read attr propietario tamaño del grupo fecha hora nombre de archivo; hacer

```
gato <- EOF
Nombre: $filename
Tamaño: $size
Dueño: $owner
Grupo: $group
Modificado: $date $time
Enlaces: $links
Atributos: $attr

EF
Hecho < <(ls -l | cola -n +2)
```

---

El bucle ejecuta la lectura para cada línea de una lista de directorios. El listado en sí se produce en la última línea del script. Esta línea redirige la salida de la sustitución del proceso a la entrada estándar del lazo. El comando tail se incluye en la canalización de sustitución de procesos para eliminar la primera línea de la lista, que no es necesaria.

Cuando se ejecuta, el script produce una salida como esta:

```
[me@linuxbox ~]$ pro_sub | cabeza -n 20
Nombre: Tamaño de
addresses.ldif:
14540
Dueño: me
Grupo: me
Modificado: 2012-04-02 11:12
Enlaces: 1
Atributos: -rw-r--r--

Nombre: bote
Tamaño: 4096
Dueño: me
Grupo: me
Modificado: 2012-07-10 07:31
Enlaces: 2
Atributos: drwxr-xr-x

Nombre: bookmarks.html
Tamaño: 394213
Dueño: me
Grupo: me
```

---

## Tram

### pas

En el capítulo 10, vimos cómo los programas pueden responder a las señales. También podemos agregar esta capacidad a nuestros scripts. Si bien los scripts que hemos escrito hasta ahora no han necesitado esta capacidad (porque tienen tiempos de ejecución muy cortos y no crean archivos temporales), los scripts más grandes y complicados pueden beneficiarse de tener una rutina de manejo de señales.

Cuando diseñamos un script grande y complicado, es importante tener en cuenta lo que sucede si el usuario cierra la sesión o apaga el equipo mientras se ejecuta el script. Cuando se produce un evento de este tipo, se enviará una señal a todos los procesos afectados. A su vez, los programas que representan esos procesos pueden realizar acciones para garantizar una terminación adecuada y ordenada del programa. Supongamos, por ejemplo,

que  
escribim  
os un  
script  
que creó  
un  
archivo  
temporal



durante su ejecución. En el curso de un buen diseño, haríamos que el script eliminara el archivo cuando el script termine su trabajo. También sería inteligente hacer que el script borre el archivo si se recibe una señal que indica que el programa va a terminar prematuramente.

bash proporciona un mecanismo para este propósito conocido como trampa. Las trampas se implementan con el comando integrado con el nombre adecuado. trap utiliza la siguiente sintaxis:

```
argumento trampa señal [señal...]
```

donde *argument* es una cadena que se leerá y tratará como un comando, y *signal* es la especificación de una señal que desencadenará la ejecución del comando interpretado.

He aquí un ejemplo sencillo:

---

```
#!/bin/bash

# trap-demo : demostración sencilla de manejo de señales

trampa "echo 'Te estoy ignorando'". SIGINT

SIGTERM for i in {1..5}; hacer
    echo "Iteración $i de 5"
    dormir 5
hecho
```

---

Este script define una trampa que ejecutará un comando echo cada vez se recibe la señal SIGINT o SIGTERM mientras se ejecuta el script. La ejecución del programa se ve así cuando el usuario intenta detener el script presionando CTRL-C:

---

```
[me@linuxbox ~]$ trampa-demo
Iteración 1 de 5
Iteración 2 de 5
Te estoy
ignorando.
Iteración 3 de 5
Te estoy
ignorando.
Iteración 4 de 5
Iteración 5 de 5
```

---

Como podemos ver, cada vez que el usuario intenta interrumpir el programa, se imprime el mensaje en su lugar.

Construir una cadena para formar una secuencia útil de comandos puede ser incómodo, por lo que es una práctica común especificar una función de shell como comando. En este ejemplo, se especifica una función de shell independiente para cada señal que se va a manejar:

---

```
#!/bin/bash

# trap-demo2 : demostración de manejo de señal simple

exit_on_signal_SIGINT () {
    echo "Guion interrumpido". 2>&1
    salida 0
}
```

---

```

exit_on_signal_SIGTERM () {
    echo "Guion terminado. 2>&1
    salida 0
}

trampa exit_on_signal_SIGNALT
trampa SIGNALT
exit_on_signal_SIGNALTERM SIGNALTERM

porque i en {1..5}; do
    echo "Iteración $i de 5"
    dormir 5
hecho

```

---

Este script cuenta con dos comandos de trampa, uno para cada señal. Cada trampa,

A su vez, especifica una función de shell que se ejecutará cuando se reciba la señal en particular. Obsérvese la inclusión de un comando de salida en cada una de las funciones de manejo de señales. Sin una salida, el script continuaría después de completar la función.

Cuando el usuario presiona CTRL-C durante la ejecución de este script, los resultados son los siguientes:

---

```
[me@linuxbox ~]$ trampa-demo2
Iteración 1 de 5
Iteración 2 de 5
Script
```

---

interrumpido.

## EXPEDIENTES TEMPORARIOS

Una de las razones por las que se incluyen controladores de señales en los scripts es para eliminar los archivos temporales que el script puede crear para contener resultados intermedios durante la ejecución. Hay algo de arte en nombrar archivos temporales. Tradicionalmente, los programas en sistemas tipo Unix crean sus archivos temporales en el directorio */tmp*, un directorio compartido destinado a dichos archivos. Sin embargo, dado que el directorio es compartido, esto plantea ciertos problemas de seguridad, especialmente para los programas que se ejecutan con privilegios de superusuario. Aparte del paso obvio de establecer los permisos adecuados para los archivos expuestos a todos los usuarios del sistema, es importante dar a los archivos temporales nombres de archivo no predecibles. Esto evita un exploit conocido como ataque de *carrera temporal*.

Una forma de crear un nombre no predecible (pero descriptivo) es hacer algo como esto:

```
 tempfile=/tmp/${nombrebase $0}.$$.RANDOM
```

Esto creará un nombre de archivo que consta del nombre del programa, seguido de su ID de proceso (PID), seguido de un número entero aleatorio. Nótese, sin embargo, que el

`$RANDOM` variable de shell devuelve un valor solo en el rango de 1 a 32767, que no es un rango muy grande en términos informáticos, por lo que una sola instancia de la variable no es suficiente para superar a un atacante.



El programa mktemp acepta una plantilla como argumento que se utiliza para construir el nombre del archivo. La plantilla debe incluir una serie de caracteres X, que se sustituyen por un número correspondiente de letras y números aleatorios. Cuanto más larga sea la serie de X caracteres, más larga será la serie de caracteres aleatorios. He aquí un ejemplo:

```
 tempfile=$(mktemp /tmp/foobar.$$.XXXXXXXX)
```

Esto crea un archivo temporal y asigna su nombre a la variable tempfile. Los caracteres X de la plantilla se sustituyen por letras y números aleatorios para que el nombre de archivo final (que, en este ejemplo, también incluya el valor expandido del parámetro especial \$\$ para obtener el PID) sea algo así como

```
/tmp/foobar.6593.UOZuvM6654
```

Mientras que la página de manual mktemp indica que mktemp crea un nombre de archivo temporal,  
mktemp también crea el archivo.

En el caso de los scripts ejecutados por usuarios normales, puede ser conveniente evitar el uso del directorio /tmp y crear un directorio para archivos temporales dentro del directorio principal del usuario, con una línea de código como esta:

## Ejecución asincrónica

A veces es deseable realizar más de una tarea al mismo tiempo. Hemos visto que todos los sistemas operativos modernos son al menos multitarea, si no multiusuario también. Los scripts se pueden construir para que se comporten de forma multitarea.

Por lo general, esto implica iniciar un script que, a su vez, inicia uno o más scripts secundarios que realizan una tarea adicional mientras el script principal continúa ejecutándose. Sin embargo, cuando una serie de scripts se ejecuta de esta manera, puede haber problemas para mantener coordinados al elemento primario y al secundario. Es decir, ¿qué pasa si el padre o el hijo dependen el uno del otro, y un script debe esperar a que el otro termine su tarea antes de terminar la suya?

bash tiene un comando incorporado para ayudar a administrar *la ejecución asíncrona* como esta. El comando wait hace que un script principal se detenga hasta que finalice un proceso especificado (es decir, el script secundario).

### esperar

Primero demostraremos el comando de espera. Para ello, necesitaremos dos scripts. Este es el script principal:

```
#!/bin/bash  
  
# async-parent : Demostración de ejecución asíncrona  
  
(padre) echo "Padre: iniciando..."
```

```
echo "Padre: iniciando el script
secundario..." async-child &
pid=$!
echo "Padre: hijo (PID= $pid) lanzado.

echo "Padre: continuando...
Dormir 2

echo "Padre: haciendo una pausa para esperar a que
el niño termine..." Espera $pid

echo "Padre: el niño ha terminado.
Continuando..." echo "Padre: el padre está
hecho. Saliendo".
```

---

Y aquí está el script secundario:

```
#!/bin/bash

# async-child : Demostración de ejecución asíncrona

(child) echo "Child: child is running..."
Duerme 5
echo "Niño: el niño está hecho. Saliendo".
```

---

En este ejemplo, vemos que el script hijo es muy simple. La verdadera acción la realiza el padre. En el script principal, el script secundario se inicia y se coloca en segundo plano. El ID de proceso del script secundario se registra asignando la variable pid con el valor de \$! shell, que siempre contendrá el ID de proceso del último trabajo puesto en segundo plano.

El script principal continúa y luego ejecuta un comando de espera con el PID del proceso secundario. Esto hace que el script principal se detenga hasta que se cierre el script secundario, momento en el que concluye el script principal.

Cuando se ejecutan, los scripts padre y secundario producen el siguiente resultado:

```
[me@linuxbox ~]$ async-parent
Padre: a partir de...
Padre: iniciando el script
secundario... Padre: hijo (PID= 6741)
iniciado. Padre: continuando...
Niño: el niño está corriendo...
Padre: haciendo una pausa para esperar a que
el niño termine... Niño: el niño está hecho.
Salir.
Padre: el hijo ha terminado. Continuar...
Padre: el padre está hecho. Salir.
```

---

## Canalizaciones

En la mayoría de los sistemas tipo Unix, es posible crear un tipo especial de archivo llamado tubería con nombre. *Las tuberías con nombre* se utilizan para crear una conexión entre dos procesos y se pueden utilizar como otros tipos de archivos. No son tan populares, pero es bueno conocerlos.

Existe una arquitectura de programación común llamada *cliente/servidor*, que puede hacer uso de un método de comunicación, como tuberías con nombre, así como otros tipos de *comunicación entre procesos*, como conexiones de red.

El tipo de sistema cliente/servidor más utilizado es, por supuesto, un navegador web que se comunica con un servidor web. El navegador web actúa como cliente, realizando solicitudes al servidor, y el servidor responde al navegador con páginas web.

Las canalizaciones con nombre se comportan como archivos, pero en realidad forman búferes FIFO (primero en entrar, primero en salir). Al igual que con las tuberías ordinarias (sin nombre), los datos entran por un extremo y salen por el otro. Con tuberías con nombre, es posible configurar algo como esto:

```
proceso1 > named_pipe
```

y

```
proceso2 < named_pipe
```

y se comportará como si

```
proceso1 | Proceso2
```

### **Configuración de una canalización con nombre**

Primero, debemos crear una tubería con nombre. Esto se hace usando el comando mkfifo:

---

```
[me@linuxbox ~]$ mkfifo pipe1  
[me@linuxbox ~]$ ls -l pipe1  
prw-r--r-- 1 me me 0 2012-07-17 06:41 pipa1
```

---

Aquí usamos mkfifo para crear una tubería con nombre llamada pipe1. Usando ls, examinamos el archivo y vemos que la primera letra en el campo de atributos es p, lo que indica que es una tubería con nombre.

### **Uso de canalizaciones con nombre**

Para demostrar cómo funciona la tubería nombrada, necesitaremos dos ventanas de terminal (o, alternativamente, dos consolas virtuales). En el primer terminal, introducimos un comando simple y redirigimos su salida a la tubería nombrada:

---

```
[me@linuxbox ~]$ ls -l > pipe1
```

---

Después de presionar ENTER, el comando aparecerá para colgarse. Esto se debe a que todavía no hay nada que reciba datos del otro extremo de la tubería. Cuando esto ocurre, se dice que la tubería está *bloqueada*. Esta condición se borrará una vez que conectemos un proceso al otro extremo y comience a leer la entrada de la tubería. Usando la segunda ventana del terminal, ingresamos este comando:

---

```
[me@linuxbox ~]$ gato < pipa1
```

---

La lista de directorios producida desde la primera ventana del terminal aparece en el segundo terminal como la salida del comando cat. El comando ls en el primer terminal se completa con éxito una vez que ya no está bloqueado.

## **Nota final**

Bueno, hemos completado nuestro viaje. Lo único que queda por hacer ahora es la práctica, la práctica, la práctica. A pesar de que cubrimos mucho terreno en nuestra caminata, apenas arañamos la superficie en lo que respecta a la línea de comando. Todavía quedan miles de programas de línea de comandos por descubrir y disfrutar. ¡Empieza a buscar en `/usr/bin` y verás!

# INDEX

## Símbolos

- opción de ayuda , 42
- \$\*, 386
- \$@, 386
- \${!matriz[\*]}, 420
- \${!matriz[@]}, 420
- \${!prefijo\*}, 402
- \${!prefijo@}, 402
- \$#{parámetro}, 402
- \${parámetro:=palabra}, 400
- \${parámetro:-palabra}, 400
- \${parámetro:+palabra}, 401
- \${parámetro:?palabra}, 401
- \${patrón de parámetro/cadena}, 403
- \${parámetro/#patrón/cadena}, 403
- \${parámetro/%patrón/cadena}, 403
- \${parámetro/patrón/cadena}, 403
- \${parámetro##patrón}, 403
- \${parámetro#patrón}, 403
- \${parámetro%%patrón}, 403
- \${parámetro%patrón}, 403
- \$!, 430
- \$#, 382
- \$((expresión)), 404
- 0 \$ 385
- ./configurar, 302
- .bash\_history, 73 años
- .bash\_login, 113
- .bash\_profile, 112
- .bashrc, 113, 115, 312, 332, 385
- .perfil, 113
- .ssh/known\_hosts, 184
- /, 19
- /papelera, 19
- /bota, 19
- /boot/grub/grub.conf, 19
- /boot/vmlinuz, 19
- /dev, 20
- /dev/cdrom, 165
- /dev/dvd, 165
- /dev/disquete, 165
- /dev/null, 52
- /etc, 20
- /etc/bash.bashrc, 113
- /etc/crontab, 20
- /etc/fstab, 20, 160, 170
- /etc/grupo, 79
- /etc/passwd, 20, 79, 241, 245, 352
- /etc/perfil, 112, 114
- /etc/sombra, 79
- /etc/sudoers, 87
- /lib, 20
- /perdidos+encontrados, 20
- /medios de comunicación, 20
- /mnt, 20
- /opt, 20
- /proc, 21
- /raíz, 21, 88
- /sbin, 21
- /tmp, 21, 429
- /usr, 21
- /usr/bin, 21
- /usr/lib, 21
- /usr/local, 21
- /usr/local/bin, 21, 307, 312
- /usr/local/sbin, 312
- /usr/sbin, 21

*/usr/share*, 21  
*/usr/share/dict*, 219  
*/usr/share/doc*, 21, 45  
*/var*, 22  
*/var/registro*, 22  
*/var/registro/mensajes*, 22, 57, 166  
(( )) comando compuesto, 404, 409  
[ comando, 365

## Un

Comando A2PS , 292  
Nombres de ruta absolutos, 9  
Comando de alias , 46, 111  
alias, 40, 46, 110  
Normas Nacionales Americanas  
(ANSI), 142 Código  
Estándar Americano para  
Intercambio de  
información.  
Ver ASCII  
anclas, 219  
servidores FTP anónimos, 179  
ANSI (Instituto Nacional Americano de  
Normalización ), 142  
Códigos de escape ANSI, 143  
ANSI.SYS, 142  
Servidor web Apache, 104  
A propósito del comando, 43  
apt-cache , 152  
Comando apt-get , 152  
Comando de aptitud, 152  
archivo, 205  
expansión aritmética, 62, 65–66, 321,  
399, 404  
expresiones aritméticas, 62, 396, 404,  
406, 416  
operadores aritméticos, 62, 405  
Pruebas de verdad aritmética,  
342, 404 matrices  
anexando valores al final, 420  
asignando valores, 416  
creando, 416  
suprimiendo, 421  
determinando el número  
de  
elementos, 419  
hallazgo de subíndices  
utilizados, 420 índice, 416

multidimensional, 416  
lectura de variables, 348  
clasificación, 420  
Subíndice, 416  
bidimensional, 415  
ASCII (Código Estándar Americano  
para el Intercambio de  
Información), 17, 68, 71,  
198, 222, 292

Carácter de campana, 140  
retorno de carro, 236  
Orden de cotejo, 222, 224, 339  
Códigos de control, 68, 222,  
286 Controlador de salida  
Groff, 280 Carácter de  
avance de línea, 236  
carácter nulo, 198  
caracteres imprimibles, 222  
texto, 17

Comando aspell, 263  
ensamblar, 298  
Lenguaje ensamblador, 298  
operadores de asignación, 407  
ejecución asincrónica, 429  
CD de audio, 163, 172  
Lenguaje de programación AWK,  
263, 412

## B

Referencias inversas, 232, 260  
secuencias de escape de barra  
invertida, 68 especiales de  
escape de barra invertida  
caracteres, 140  
Copias de seguridad,  
incrementales, 208  
comando basename, 385  
bash (shell) 3, 110  
Página de manual,  
44  
expresiones regulares básicas, 224, 231,  
257, 260, 269  
Comando BC , 412  
Distribución de software de  
Berkeley (BSD), 290  
Comando BG , 102  
binario, 81–82, 85, 298, 405  
Máscara de bits, 84  
operadores de bits, 409

Bourne, Steve, 3 años  
expansión de la abrazadera, 63, 65, 394

- ramificación, 333  
Comando de ruptura , 360, 389  
Enlaces rotos, 37  
BSD (Berkeley Software Distribución), 290  
Comportamiento al estilo BSD, 98  
almacenamiento en búfer, 164  
Insectos, 369–373  
Entorno de compilación, 302  
Comando bzip2 , 204
- C**
- lenguaje de programación C, 298, 396, 407, 410  
Lenguaje de programación C++, 298  
comando cal , 5  
Comando de cancelación, 296  
retorno de carro, 17, 68, 140, 222–223, 235, 262, 289  
Comando compuesto de caso , 376  
Comando del gato , 53, 235  
Comando CD , 9–10  
comando cdrecord , 172  
CD-ROM, 162–163, 172  
Paquete CDRtools , 172  
clases de caracteres, 26–27, 220–224, 227, 255, 262  
Rangos de caracteres, 27, 220–221, 262  
    Comando chgrp , 91  
    proceso infantil, 96  
    Comando chmod , 81, 92, 311  
    Comando Chown , 90, 92  
    Ordenación cronológica, 241  
    Texto claro, 179, 182  
    arquitectura cliente-servidor,  
    lenguaje de programación COBOL  
    431, orden de intercalación 298,  
    111, 222, 224,  
        254, 339  
        ASCII, 224, 339  
    Diccionario, 222  
        tradicional, 224  
comando de comunicación, 249  
    Historial de comandos, 4, 73  
    línea de comandos  
        Argumentos, 382  
        Edición, 4, 70  
expansión, 59  
Historia, 4, 74  
interfaces, 26, 28  
opciones de comando, 14  
Sustitución de comandos, 64–65, 394  
comandos  
    Argumentos, 14, 382  
    tipo determinante, 40  
    documentación, 41  
    archivos de programa ejecutables, 40, 299 ejecutándose como otro usuario, 87 opciones largas, 14  
    opciones, 14  
Comentarios, 114, 118, 262, 310, 371  
Sistema de impresión Unix común (CUPS), 288  
operadores de comparación, 409  
Compilación, 298  
Finalizaciones, 72  
comandos compuestos  
    ( ), 342, 354, 404  
    [[ ]], 341, 354  
    Caja, 376  
    A favor, 393  
    Si, 334  
    Hasta, 361  
    mientras, 358  
algoritmos de compresión, 202  
Expresiones condicionales, 366  
Archivos de configuración, 17, 20, 109  
Comando de configuración , 302  
constantes, 319  
Comando Continuar, 360  
Caracteres de control, 141, 235  
Códigos de control, 68,  
222 operadores de control  
    &&, 345, 354  
    ||, 345  
terminal de control, 96  
*COPIAR* (archivo de documentación), copiar y pegar 301  
    En la línea de comandos,  
    70 en Vim, 129  
    con X Window System, paquete de 5 coreutils, 42, 44–45, 246  
contando palabras en un archivo, 55

Comando CP , 28, 33, 116, 185  
CPU, 95, 298  
Cron Job, 189  
Crucigramas, 219  
Comando csplit , 266  
CUPS (Sistema de impresión común de Unix), 288  
directorio de trabajo actual, 8 movimiento del cursor, 70 comando de corte, 243, 403

## D

Programas demoníacos, 96, 104  
compresión de datos, 202  
redundancia de datos, 202  
validación de datos, 341  
comando de fecha , 5  
formatos de fecha, 241  
Comando DD , 171  
Debian, 150  
depuración, 330, 370  
Programación defensiva, 367, 370  
Delimitadores, 66, 239, 241  
Dependencias, 151, 305  
diseño, 368, 370  
controladores de dispositivos, 156, 298  
nombres de dispositivos, 164  
Nodos de dispositivo, 20  
Comando DF , 6, 331  
DHCP (Protocolo de configuración dinámica de host ), 178  
programa de dicción, 300  
orden de intercalación del diccionario, 222 comando diff, 250  
Gestión de derechos digitales (DRM), 151  
Directarios  
    archivo, 205  
    cambiante, 9  
    copiando, 28  
    Creación, 28, 33  
    corriente de trabajo, 8  
    Suprimir, 31, 37  
    jerárquico, 7  
    Inicio, 20, 79, 332  
    listado, 13

en movimiento, 30, 35  
navegando, 7  
OLD\_PWD variable, 111  
padre, 8  
Variable PATH , 111  
Variable PWD , 112  
remover, 31, 37  
cambio de nombre, 30, 35  
raíz, 7  
compartidos, 91  
broca pegajosa, 86  
sincronización, 211  
transferencia a través de una red, 211  
visualización de contenidos, 8  
particiones de disco, 161  
Variable DISPLAY , 111  
Delfín, 28 años  
DOS2UNIX Comando, 236  
comillas dobles, 65  
Comando dpkg , 152  
DRM (Derechos Digitales  
    Gestión), 151  
du command, 238, 332  
Dynamic Host Configuration  
    Protocolo (DHCP), 178

## E

Comando Eco , 60, 111, 316  
    opción -e , 68  
    opción -n , 349  
casos de borde y esquina, 370 variable EDITOR , 111 ID de grupo efectivo, 86 ID de usuario efectivo, 86, 96  
Declaración del ELIF, 339  
correo electrónico, 234  
Sistemas embebidos, 298  
variables vacías, 400  
túneles encriptados, 185  
Cifrado, 255  
Bucle sin fin, 361  
Fin de archivo, 54, 322  
comando de suscripción , 294  
Medio Ambiente, 88, 109, 353  
    alias, 110  
    estableciendo, 112

- examinando, 110  
Shell de inicio de sesión, 112  
Funciones de shell, 110  
variables de shell, 110  
archivos de inicio, 112  
subcapas, 424  
variables, 110  
**Comando EQN**, 279  
Programas ejecutables, 40, 299, 303  
determinación de la ubicación, 41  
**Variable PATH**, 111  
**Comando de salida**, 6, 338, 356  
Estado de salida, 334, 338  
Expandir comando, 246  
expansiones, 59  
    aritmética, 62, 65–66, 321, 399, 404  
    brace, 63, 65, 394  
    sustitución de comando, 64–65, 394  
    delimitadores, 66  
    Errores resultantes de, 365  
    Historia, 74–76  
    parámetro, 64, 65–66, 319, 323, 399  
    Nombre de la ruta, 60, 65, 394  
    tilde, 61, 65  
    División de palabras, 65  
expresiones  
    aritmética, 62, 396, 404, 406, 416  
    condicional, 366  
Sistema de archivos ext3, 169  
expresiones regulares extendidas, 224  
Lenguaje de marcado extensible  
    (XML), 234
- F**
- Falsos comandos, 335  
**Comando fdformat**, 171  
**Comando fdisk**, 167  
**Comando FG**, 102  
FIFO (primero en entrar, primero en salir), 431  
**comando de archivo**, 16  
Descriptor de archivo, 51  
Protocolo de transferencia de archivos (FTP), 179
- Nombres de archivo, 198  
    Distingue entre mayúsculas y minúsculas, 11  
Espacios empotrados en, 11, 232  
extensiones, 11  
oculto, 11  
archivos  
    acceso, 78  
    Archivo, 205, 209  
    atributos, 79  
    Bloque especial, 80  
    Bloquear dispositivo especial, 190  
    cambiar el modo de archivo, 81  
    cambiar el propietario y el grupo propietario, 90  
    Carácter especial, 80  
    Dispositivo especial de 80 caracteres, 190  
    Compresión, 202  
    Configuración, 17, 109, 234  
    copia, 28, 33  
    copiando a través de una red, 179  
    creando vacío, 51  
    .deb, 150  
    suprimiendo, 31, 37, 195  
    Determinación de contenidos, 16  
    Nodos de dispositivo, 20  
    Acceso a la ejecución, 79  
    expresiones, 336, 338, 340  
    hallazgo, 187  
    Oculto, 11  
    Imagen ISO, 172–173  
    Listado, 8, 13  
    modo, 79  
    en movimiento, 30, 34  
    propietario, 81  
    permisos, 78  
    Acceso a la lectura, 79  
    regular, 190  
    remover, 31, 37  
    cambio de nombre, 30, 34–35  
    .rpm, 150  
    Biblioteca compartida, 20  
    Puesta en marcha, 112  
    broca pegajosa, 86  
    Vínculos simbólicos, 190  
    sincronización, 211  
    temporales, 428

- Archivos  
(*continuació*  
*n*) texto, 17  
transferencia a través de una red,  
179,  
209, 211  
truncando, 51  
tipo, 79  
Visualización de contenidos, 17  
Acceso de escritura, 79  
corrupción del sistema de archivos,  
164  
filtros, 55  
Encontrar comando, 189, 208  
cortafuegos, 176  
primero en entrar, primero en  
salir (FIFO), 431 disquetes,  
159, 165, 171 control de flujo  
ramificación, 333  
Comando compuesto de caso ,  
376  
Declaración del ELIF, 339  
Bucle sin fin, 361  
para el comando compuesto,  
393  
bucle for , 393  
enunciado de función, 327  
si comando compuesto, 334  
bucle, 357  
Basado en menús, 355  
Decisiones de opción múltiple,  
375 archivos de lectura con while  
y until  
    bucles, 362  
terminando un bucle, 360  
    trampas, 427  
Hasta el bucle, 361  
    bucle while, 359  
Comando FMT , 271  
Política de enfoque, 5  
Comando de plegado , 271  
para comando compuesto,  
393 para bucle, 393  
Prospectiva, 150  
Lenguaje de programación Fortran,  
298, 395  
Comando Libre , 6, 164  
Fundación para el Software Libre,  
xxix  
Comando FSCK , 170
- FTP (Protocolo de Transferencia de Archivos),  
179  
Comando FTP , 179, 186,  
300, 323  
Servidores FTP, 179, 323

Variable FUNCNAME , 385  
enunciado de función, 327

## G

GCC (compilador), 299  
Comando gedit , 101, 115  
Comando Genisoimage, 172  
Gentoo, 150  
Guión fantasma, 288  
gid (ID de grupo primario), 78  
variables globales, 328  
Globbing, 26  
GNOMO, 3, 28, 38, 84, 115, 186  
terminal gnome, 3  
Paquete GNU binutils, 395  
Compilador C de GNU, 299  
Paquete GNU coreutils, 42,  
    44–45, 246  
    Proyecto GNU, 14, 29, 300–301  
        comando info , 44–45  
GNU/Linux, 29  
interfaz gráfica de usuario (GUI), xxvi, 5, 28,  
    38, 70, 84, 112  
Comando Grep , 56, 216, 352  
Comando Groff , 279  
comandos de grupo, 423  
grupos, 78  
    ID de grupo efectivo, 86 ID de  
        grupo primario, 78  
GUI (interfaz gráfica de usuario), xxvi, 5, 28,  
    38, 70, 84, 112  
Comando Gunzip , 202  
comando gzip , 45, 202

## H

discos duros, 159  
Enlaces duros, 23, 32, 35  
    creando, 35  
    listado, 36  
Comando de la cabeza, 56  
Archivos de cabecera, 302  
Programa "Hola Mundo", 310  
    Comando de ayuda, 41  
aquí documentos, 321

aquí cuerdas, 353  
hexadecimal, 82, 405  
Archivos ocultos, 11, 61  
Estructura jerárquica de directorios, 7 programas de alto nivel  
idiomas, 298  
historia  
expansión, 74–76  
Buscando, 74  
**Comando de Historia**, 74  
directorios de inicio, 8, 10, 20, 61, 88, 111  
*/etc/passwd*, 79  
Cuenta raíz, 21  
**Variable HOME**, 111  
nombre de host, 140  
**HTML** (Marcado de hipertexto Idioma), 234, 263, 279, 315, 326

**Y**

Comando id, 78  
IDE, 165  
si comando compuesto, 114, 365, 375  
**IFS** (Separador de Campo Interno) variable, 351  
ECHO\_REQUEST de la CPIM, 176 copias de seguridad incrementales, 208  
Archivos de información, 45  
**Programa de inicio**, 96  
guiones de inicio, 96  
inodos, 36  
*INSTALL* (archivo de documentación), asistente de instalación 301, 150  
Enteros  
Aritmética, 62, 411  
división, 62, 405  
interactividad, 347  
Separador de campo interno (**IFS**) variable, 351  
lenguas interpretadas, 299  
programas interpretados, 299  
intérprete, 299

Redirección de E/S, 49. Véase también imágenes ISO  
de redirección, 172–173  
ISO9660 (Tipo de dispositivo), 162, 173

## J

control de trabajos, 101  
Número de empleos, 101  
Jobspec, 102  
comando de unión, 247  
Extensiones de Joliet, 173  
Alegría, Bill, 122

## K

**Comando Kate**, 115  
KDE, 3, 28, 38, 84, 115, 186  
**Comando kedit**, 115  
kernel, xxv, xxix, 19, 43, 95, 104, 157, 165, 253, 305  
controladores de dispositivos, 156  
Campos clave, 239  
**Comando de matar**, 103  
**Comando Killall**, 106  
Texto asesino, 70  
Knuth, Donald, 279  
Konqueror, 28, 84, 186  
konsole (emulador de terminal), 3 kwrite command, 101, 115

## L

LANG variable, 111, 222, 224  
menos mando, 17, 55, 211, 231  
**Comando LFTP**, 181  
bibliotecas, 299  
editores de línea, 122  
carácter de continuación de línea, 262, 313  
Enlazador (programa), 299  
Enlace (proceso), 298  
enlaces  
roto, 37  
creando, 32  
Duro, 23, 32  
simbólico, 22, 33

- Comunidad Linux, 149  
Distribuciones de Linux, 149  
    CentOS, 150, 294  
    Debian, 150, 297  
    Fedora, xxviii, 79, 150, 294  
    Prospectiva, 150  
    Gentoo, 150  
    Linspire, 150  
    Arruinado, 150  
    OpenSUSE, xxviii, 150  
    sistemas de embalaje, 149  
    PCLinuxOS, 150  
    Red Hat Enterprise Linux, 150  
    Slackware, 150  
    Ubuntu, xxviii, 149–150, 294  
    Xandros, 150  
Estándar de jerarquía del sistema de archivos de Linux, 19, 312  
Linux kernel, xxv, xxix, 19, 43, 95, 104, 157, 165, 253, 305  
controladores de dispositivos, 156  
    caracteres literales, 218  
    Comando LN , 32, 35  
    variables locales, 329  
Localización, 222, 224, 254, 339  
Comando local , 224  
localhost, 182  
Comando de localización, 188, 230  
Errores lógicos, 366  
Operadores lógicos, 192–193, 343  
relaciones lógicas, 192, 195  
administrador de volúmenes lógicos (LVM),  
    159, 162  
Mensaje de inicio de sesión, 6, 180  
Shell de inicio de sesión, 79, 88, 112  
opciones largas, 14  
Interfaz de bucle invertido, 178  
bucle, 357  
bucles, 367, 406, 408, 422, 425  
compresión sin pérdidas, 202  
Compresión con pérdidas, 202  
Comando LP , 291  
Comando lpq , 295  
Comando LPR , 290  
Comando LPRM , 296  
Comando lpstat , 294
- ls comando, 8, 13  
Formato largo, 15  
    visualización de atributos de archivo, 79  
Lukyanov, Alexander, 181  
LVM (gestor de volúmenes lógicos),  
    159, 162
- ## M
- lenguaje de máquina, 298  
mantenimiento, 312, 316, 318, 325  
Hacer comando, 303  
*Makefile*, 303  
Comando del hombre , 42  
Páginas de manual, 42, 280  
Lenguajes de marcado, 234, 279  
memoria  
    asignados a cada proceso, 96  
    que muestran libres, 6  
    RSS (Tamaño del conjunto residente), 98  
    violación de segmentación, 105  
    Uso, 98, 106  
    virtual, 98  
programas basados en menús, 355  
Clave meta, 72  
Metacaracteres, 218  
metadatos, 150, 152  
Metasecuencias, 218  
Comando mkdir , 28, 33  
Comando mkfifo , 431  
Comando MKFS , 169, 171  
Comando MKISOFS , 172  
Comando mktemp , 428  
mnemotecnia, 298  
Editor modal, 124  
Fuentes monoespaciadas, 288  
Moolenaar, Bram, 122  
Comando de montaje , 161, 173  
Puntos de montaje, 20, 161, 163  
montaje, 160  
Archivos MP3, 91  
decisiones de opción múltiple, 375  
multitarea, 77, 95, 429  
sistemas multiusuario, 77  
Comando MV , 30, 34

**N**

- Pipas con nombre, 430
- Comando nano , 122
- Nautilus, 28, 84, 186
- comando netstat , 178
- redes, 175
  - Servidores FTP anónimos, 179
  - ruta predeterminada, 179
  - Configuración dinámica de host
    - Protocolo (DHCP), 178
  - túneles encriptados, 185
  - examinando la configuración de la red y
    - Estadísticas, 178
  - Protocolo de transferencia de archivos (FTP), 179
  - firewalls, 176
  - red de área local (LAN), 179
  - interfaz de circuito cerrado, 178
  - Ataques de intermediarios, 182
  - enrutadores, 178
  - Comunicación segura con
    - hosts remotos, 182
  - probando si un host está activo, 176
  - rastreando la ruta a un host, 177
  - transfiriendo archivos, 211
  - transporte de archivos, 179
  - red privada virtual, 185
  - caracteres newline, 66, 140
  - NEWS* (archivo de documentación), 301 comando nl , 268
  - Artículo 279
  - carácter nulo, 198
  - bases numéricas, 405

**O**

- octal, 82, 405, 418
- Archivos Ogg Vorbis,
- 91 OLD\_PWD variable, 111
- OpenOffice.org escritor, 17 años
- OpenSSH, 182
- Operadores
  - Aritmética, 62, 405
  - asignación, 407
  - binario, 366
  - comparación, 409
  - ternario, 410

## P

- archivos de paquetes, 150
- Mantenedores de paquetes, 151
- gestión de paquetes, 149 estilo Debian (*.deb*), 150 búsqueda de paquetes, 152 herramientas de alto nivel, 152 instalación de paquetes, 153 herramientas de bajo nivel, 152 repositorios de paquetes, 151 estilo Red Hat (*.rpm*), 150 eliminación de paquetes, 154 Actualización de paquetes, 154 sistemas de embalaje, 149 lenguaje de descripción de página, 234, 281, 287      Variable de PAGER , 111
- buscapersonas, 18
- expansión de parámetros, 64, 65–66, 319, 323, 399
- proceso de los padres, 96
- Comando passwd , 93
- contraseñas, 93
- comando de pegado, 246 discos duros PATA, 165 comando de parche, 253
- parches, 250
- Variable PATH , 111, 114, 311, 327
- Expansión de nombre de ruta, 60, 65, 394
- Nombres de rutas, 230
  - Absoluto, 9
  - finalización, 72
  - pariente, 9
- PDF (Formato de documento portátil), 281, 290
- Lenguaje de programación Perl, 40, 216, 263, 299, 412
- permisos, 310
- Lenguaje de programación PHP, 299
- comando ping , 176
- Oleoductos, 54, 353, 425
  - en sustitución de mando, 64
  - portabilidad, 304, 332, 345
- Formato de documento portátil (PDF), 281, 292

Interfaz de sistema operativo portátil cara (POSIX). *Ver* POSIX  
(Interfaz de Sistema Operativo Portátil)  
parámetros posicionales, 381, 400–402  
POSIX (Sistema de Operación Portátil)  
    tiene interfaz), 222,  
    224–225, 345  
    clases de caracteres, 26, 221,  
        223–224, 227, 255, 262  
Posdata, 234, 280, 287, 292  
comando `pr`, 274, 288 ID de grupo primario (gid), 78  
caracteres imprimibles, 222  
Comando `printenv`, 64, 110  
impresoras, 164  
    salida de almacenamiento en búfer, 164  
    códigos de control, 286  
    rueda de margarita, 286  
    Nombres de dispositivos, 165  
    conductores, 288  
    gráfico, 287  
    impacto, 286  
    láser, 287  
Comando `printf`, 275, 398  
 impresión  
    Determinación del estado del sistema, 294 Historia de, 286  
    Protocolo de impresión de Internet, 295 fuentes monoespaciadas, 286  
    Preparación del texto, 288  
    Bonito, 292  
    fuentes proporcionales, 287  
    colas, 294, 295–296  
    bobina, 294  
    Finalización de trabajos de impresión, 296 trabajos de visualización, 295  
ID de proceso, 96  
sustitución de procesos, 425  
procesos, 95  
    Antecedentes, 101  
    Niño, 96 años  
    controlando, 100  
    primer plano, 101  
    interrumpiendo, 101  
    control de trabajos, 101  
asesinatos, 103

Niza, 97  
padre, 96 años  
ID de proceso, 96  
**SIGINTA**, 427  
señales, 103  
**SIGTERM**, 427  
durmiendo, 97  
Estado, 97  
parada, 102  
Visionado, 96, 98  
zombi, 97  
uso de producción, 368  
finalización programable, 73  
Comando **PS** , 96  
**PS1** variable, 112, 140  
**PS2** variable, 317  
Comando **PS2pdf** , 281  
**PS4** variable, 372  
Pseudocódigo, 333, 358  
Comando **PSTREE** , 106  
**PuTTY**, 186  
Comando **PcD** , 8  
Variable **PWD** , 112  
Lenguaje de programación Python, 299

## **Q**

citando, 65  
comillas dobles, 65  
Carácter de escape, 67  
Cita faltante, 364  
comillas simples, 67

## **R**

RAID (matriz redundante de discos independientes), 159  
procesador de imágenes rasterizadas (RIP), 288  
comando de lectura, 348–351, 362,  
368, 425  
Readline, 70  
*README* (archivo de documentación),  
45, 301  
redirecciónamiento  
tubería bloqueada, comandos de  
grupo 431 y  
subcapas, 424

- aquí documentos, 321
- aquí cuerdas, 353
- Error estándar, 51
- Entrada estándar, 53, 323
- Salida estándar, 50
- operadores de redirecciónamiento
  - >, 52
  - >, 50
  - >>, 51 años
  - >(*lista*), 425
  - <, 54
  - <<, 322–323
  - <<-, 323
  - <<<, 353
  - <(*lista*), 425
  - |, 54
- matriz redundante de discos
  - independientes (RAID), 159
- expresiones regulares, 56, 215, 259, 341, 352
  - anclas, 219
  - Referencias traseras, 232, 259–260
  - Básico, 224, 231–232, 257, 260, 269
  - ampliado, 224
- Bases de datos relacionales, 247
- Nombres de ruta relativos, 9
- "release early, release often", 369
- eliminando líneas duplicadas en un archivo, 55 REPLY variable, 348, 425
- Generador de informes, 315
- repositorios, 151
- comando de retorno , 328, 338
- RIP (procesador de imágenes rasterizadas), 288
- comando rlogin , 182
- Comando RM , 31
- Extensiones de Rock Ridge, 173
- Comando Roff , 279
- Codificación ROT13, 255
- Comando rpm , 152
- Comando rsync , 212
- Protocolo de actualización remota rsync, 212 Lenguaje de programación Ruby, 299

## S

variables escalares, 415

Comando SCP , 185  
comando de script , 76  
Lenguajes de scripting, 40, 299  
Comando sdiff , 266  
búsqueda de patrones en un archivo, 56  
historial de búsqueda, 74  
Shell seguro (SSH), 182  
Comando sed , 256, 282, 403  
Comando de conjunto , 110, 371  
Setuid, 86, 337  
Seward, Julián, 204  
Comando SFTP , 186  
Bibliotecas compartidas, 20, 151  
shebang, 311  
Construcciones de capas, 40  
Funciones de shell, 40, 110, 327, 385  
Indicaciones de shell, 4, 9, 75, 88, 101, 112,  
    139, 183, 317  
Scripts de shell, 309  
Variable SHELL , 111  
variables de shell, 110  
Comando de turno , 383, 388  
Señal SIGINT , 427  
señales, 426  
comillas simples, 67  
Slackware, 150  
Comando de sueño , 360  
Eslabón blando, 22  
Comando de ordenación , 55, 236  
Claves de ordenación, 239  
Código fuente, 150, 156, 235, 297  
Comando de la fuente, 118, 312  
Árbol de origen, 300  
Parámetros especiales, 385, 401  
comando dividido , 266 SSH  
(Secure Shell), 182  
programa ssh , 77, 183, 209 Stallman, Richard,  
xxv, xxix, 116,  
    225, 299  
        Error est\'andar, 50  
    desechando, 52 redirigiendo a un  
        archivo, 51  
Entrada est\'andar, 50, 323, 348  
    redireccionamiento, 53  
Salida est\'andar, 50 Anexando a un  
    fichero, 51 Desechando, 52

Salida estándar ( <i>continuación</i> )	Resaltado de sintaxis, 310, 314
Redirecciónamiento Error estándar a, 52	
Redirecciónamiento a un archivo, 50	
archivos de inicio, 112	
Comando de estadísticas , 199	
broca pegajosa, 86	
dispositivos de almacenamiento, 159	
CD de audio, 163, 172	
CD-ROM, 162–163, 172	
Creación de sistemas de archivos, 167	
nombres de dispositivos, 164	
particiones de disco, 161	
FAT32, 167	
disquetes, 165, 171	
formato, 167	
LVM, 162	
puntos de montaje, 161, 163	
particiones, 167	
leer y escribir directamente, 171	
reparación de sistemas de archivos, 170	
desmontaje, 163	
memorias USB, 171	
Editor de transmisión, 256, 282, 403 cadenas	
\${ <i>parámetro</i> : <i>desplazamiento</i> }, 402	
\${ <i>parameter</i> : <i>offset</i> : <i>length</i> }, 402	
extraer una porción de, 402	
longitud de, 402	
Realizar búsqueda y reemplazo sobre, 403	
Eliminar la parte delantera de, 403 Eliminar la parte final de, 403	
comando strings , 395	
Talones, 330, 369	
<i>estilo</i> (archivo de programa), 302	
Su Comando, 87	
subcapas, 353, 423	
Comando Sudo, 87–89	
Sun Microsystems, 122	
superusuario, 4, 79, 88, 106	
Enlaces simbólicos, 22, 33, 36	
creando, 36, 38	
listado, 36	
Errores de sintaxis, 363	

# T

mesas, 247  
Datos tabulares, 239, 278  
**Comando de cola**, 56  
Archivo de cintas, 205  
**Comando tar**, 205  
bolas de alquitrán, 300  
objetivos, 303  
Administrador de tareas, 100  
Tatham, Simón, 186  
**Comando TBL**, 279, 282  
**Comando de tee**, 57  
teletipo, 96  
comando telnet, 182  
**PLAZO** variable, 112  
emuladores de terminal, 3  
sesiones de terminal  
    controlando el terminal, 96 efecto de  
    **.bashrc**, 312 entorno, 88  
    Salida, 6  
    login shell, 88, 112 con  
    sistemas remotos, 77 **TERM**  
    variable, 112  
    Uso de canalizaciones con  
        nombre, 431 virtual, 6  
Terminales, 71, 77, 142, 279  
operador ternario, 410  
Casos de prueba, 369  
comando de prueba, 336, 341, 359, 366  
Cobertura de pruebas, 370  
Pruebas, 369–370  
TEX, 279  
texto, 17  
    ajuste de la longitud de la línea, 271  
    ASCII, 17  
    retorno de carro, 236  
    comparando, 249  
    conversión de MS-DOS a Unix, 254 palabras, 55  
    corte, 243  
    Eliminación de líneas duplicadas, 242  
    Eliminación de varias líneas en blanco, 236  
    Detección de diferencias, 250 Visualización  
        de líneas comunes, 249

- visualización de caracteres de control, formato 235 DOS, 236
- Variable EDITOR , 111
- Lengüetas expansivas, 246
- Archivos, 17
- filtrado, 55
- plegable, 271
- formato, 268
- Formato para tipógrafos, 279
- tablas de formato, 282
- uniéndose, 247
- Carácter de salto de línea, de 236 minúsculas a mayúsculas conversión, 254
- Líneas de numeración, 236, 268
- Paginación, 274
- pegar, 246
- preparación para la impresión, 288 eliminación de líneas duplicadas, 55 renderizado en PostScript, 280 codificado ROT13, 255 búsqueda de patrones, 56 clasificación, 55, 236 revisión ortográfica, 263 sustitución, 259
- Sustitución de tabulaciones por espacios, 246 Tabulaciones delimitadas, 245 transliteración de caracteres, 254
- Formato Unix, 236
- ver con menos, 17, 55
- Editores de texto, 115, 234, 254
- emacs, 116
- GEDIT, 115, 310
- interactivo, 254
- Kate, 115, 310
- kedit, 115
- kwrite, 115
- línea, 122
- Nano, 115, 122
- Pico, 115
- Arroyo, 256
- Resaltado de sintaxis, 310, 314
- VI, 115
- Vim, 115, 310, 314
- Visual, 122
- para escribir scripts de shell, 310
- Expansión de tilde, 61, 65
- Comando tload , 106
- Mando superior , 98
- diseño de arriba hacia abajo, 326 Torvalds, Linus, xxv
- Comando táctil , 198–199, 213, 305, 389
- Comando TR , 254
- Comando traceroute , 177
- rastreo, 371
- transliteración de caracteres, 254
- trampas, 427
- Comando Troff, 279
- Comando verdadero , 335
- TTY (campo), 96
- comando de tipo , 40
- tipógrafos, 279, 287
- TZ variable, 112

## U

- Ubuntu, 79, 89, 149, 222, 312
- Comando UMASK , 84, 92
- Comando de montaje , 163
- Comando Unalias , 47
- operador unario esperado (error mensaje), 366
- operadores unarios, 405
- comando de desexpandir , 246
- Tokens inesperados, 365
- Comando Uniq , 55, 242
- Unix, xxvi
- Sistema Unix V, 290
- Comando Unix2DOS, 236
- Comando unset , 421
- hasta el comando compuesto, 361
- hasta el bucle, 361
- Comando de descompresión , 210
- Comando actualizadoB , 189
- proveedores ascendentes, 151
- Tiempo de actividad, 326
- Comando de tiempo de actividad, 331
- Memorias USB, 159, 171
- Usenet, 255
- Variable USER , 110, 112

## Usuarios

/etc/passwd, 79  
/etc/sombra, 79  
cuentas, 78  
cambio de identidad, 87  
cambio de contraseñas, 93  
ID de usuario efectivo, 86, 96  
Directorio de inicio, 79  
identidad, 78  
contraseña, 79  
Setuid, 86  
superusuario, 79, 81, 86–87, 93

## V

Validación de entradas, 353  
variables, 64, 318, 400  
Asignación de valores, 320, 406  
constantes, 319  
declarando, 318, 320  
medio ambiente, 110  
Global, 328  
Local, 329  
nombres, 319, 401  
escalar, 415  
caparazón, 110  
Sistema de archivos VFAT, 170  
VI Comando, 121  
Comando Vim , 232, 314  
virtual consoles, 6  
red privada virtual (VPN), 185  
terminales virtuales, 6  
Editores visuales, 122  
Comando vmstat , 106  
VPN (red privada virtual), 185

## W

Comando de espera, 429  
Comando WC , 55  
Páginas Web, 234  
Comando wget , 181  
Lo que ves es lo que obtienes  
(WYSIWYG), 286  
¿Qué es el comando?, 44  
que mando, 41  
mientras que el comando compuesto,  
358 comodines, 26, 53, 60, 216, 221  
Comando Wodim , 173  
División de palabras, 65–67  
mundo, 78  
WYSIWYG (Lo que ves es lo que  
obtienes), 286

## X

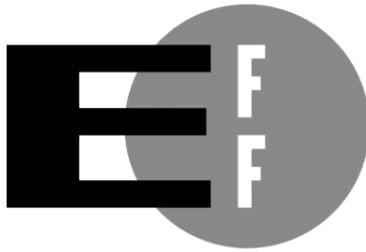
Sistema de ventanas X, 5, 77, 185  
Comando Xargs , 197  
Comando xload , 106  
comando xlogo , 100 XML  
(marcado extensible  
Idioma), 234

## Y

Tirando de texto, 70  
Comando Yum , 152

## Z

comando zgrep , 232  
comando zip , 209  
Comando Zless , 45



The **Electronic Frontier Foundation** (EFF) is the leading organization defending civil liberties in the digital world. We defend free speech on the Internet, fight illegal surveillance, promote the rights of innovators to develop new digital technologies, and work to ensure that the rights and freedoms we enjoy are enhanced – rather than eroded – as our use of technology grows.



**EFF.ORG**

**ELECTRONIC FRONTIER FOUNDATION**

Protecting Rights and Promoting Freedom on the Electronic Frontier

Protecting Rights and Promoting Freedom on the Electronic Frontier

La EFF es una organización apoyada por sus miembros. ¡Únete ahora!

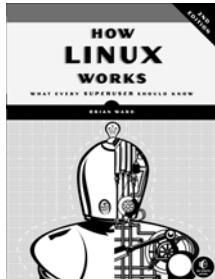
[www.eff.org/support](http://www.eff.org/support)

*La línea de comandos de Linux* se escribió utilizando OpenOffice.org Writer en un Dell Inspiron 530N, configurado de fábrica con Ubuntu 8.04. Las fuentes utilizadas en este libro son New Baskerville, Futura, TheSansMono Condensed y Dogma. El libro fue tipografiado en LibreOffice Writer.

# ACTUALIZACIONES

Visite <http://nostarch.com/tlcl.htm> para obtener actualizaciones, erratas y otra información.

Más libros sensatos de

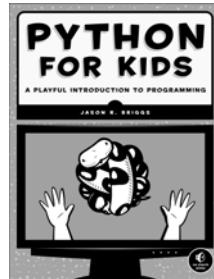


## CÓMO FUNCIONA LINUX, 2<sup>a</sup> EDICIÓN

Lo que todo superusuario debe saber  
por BRIAN WARD

NOVIEMBRE 2014, 392 PP., \$39.95

ISBN 978-1-59327-567-9



## PYTHON PARA NIÑOS

Una introducción lúdica a la programación  
por JASON R. BRIGGS

DICIEMBRE 2012, 344 PP., \$34.95

ISBN 978-1-59327-407-8

A todo color

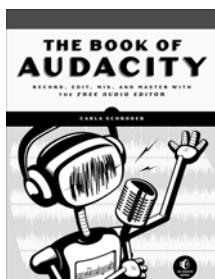


## JAVASCRIPT ELOCUENTE, 2<sup>a</sup> EDICIÓN

Una introducción moderna a la programación  
por MARIJN HAVERBEKE

NOVIEMBRE 2014, 400 PP., \$39.95

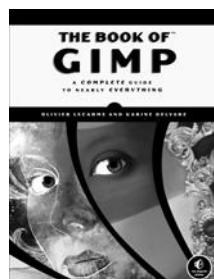
ISBN 978-1-59327-584-6



## EL LIBRO DE LA AUDACIA

Graba, edita, mezcla y masteriza con el editor de audio gratuito  
por CARLA SCHRODER

MARZO 2011, 384 PP., \$34.95  
ISBN 978-1-59327-270-8



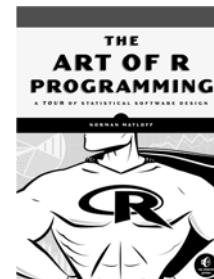
## EL LIBRO DE GIMP

Una guía completa de casi todo  
por OLIVIER LECARME Y KARINE DELVARE

ENERO DE 2013, 676 PP., \$49.95

ISBN 978-1-59327-383-5

A todo color



## EL ARTE DE LA PROGRAMACIÓN EN R

Un recorrido por el diseño de software estadístico  
por NORMAN MATLOFF

OCTUBRE 2011, 400 PP., \$39.95

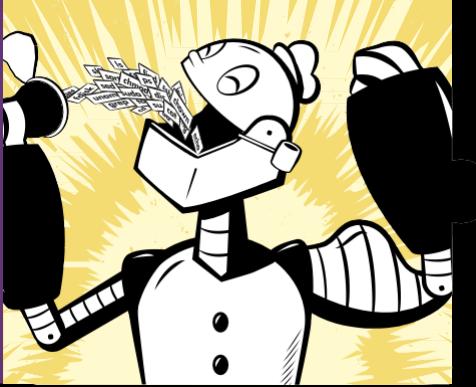
978-1-59327-384-2

TELÉFONO:  
800.420.7240 o  
415.863.9900

CORREO ELECTRÓNICO:  
[VENTAS@NOSTARCH.COM](mailto:VENTAS@NOSTARCH.COM)  
M

PÁGINA WEB:  
[WWW.NOSTAROUÍA.COM](http://WWW.NOSTAROUÍA.COM)

# BANISH YOUR MOUSE



Ha experimentado la superficie brillante de apuntar y hacer clic de su computadora Linux, ahora sumérjase y explore sus profundidades con el poder de la línea de comandos.

*La línea de comandos de Linux* te lleva desde las primeras pulsaciones de teclas del terminal hasta la escritura de programas completos en Bash, el shell de Linux más popular. A lo largo del camino, aprenderá las habilidades atemporales transmitidas por generaciones de

Gurús de barba gris que evitan el ratón: navegación de archivos, configuración del entorno, encadenamiento de comandos, coincidencia de patrones con expresiones regulares y mucho más.

Además de ese conocimiento práctico, el autor William Shotts revela la filosofía detrás de estas herramientas y la rica herencia que su máquina Linux de escritorio ha heredado de las supercomputadoras Unix de antaño.

A medida que avanza a través de los capítulos cortos y fáciles de digerir del libro, aprenderá a:

- Crear y eliminar archivos, directorios y enlaces simbólicos
- Administre su sistema, incluidas las redes, la instalación de paquetes y la gestión de

- Uso de entrada y salida estándar, redirección y canalizaciones
- Edita archivos con Vi, el editor de texto más popular del mundo
- Escriba scripts de shell para automatizar tareas comunes o aburridas
- Segmenta y corte archivos de texto con cortar, pegar, grep, parchear y sed

Una vez que supere su "conmoción inicial", descubrirá que la línea de comandos es una forma natural y expresiva de comunicarse con su computadora. Pero no te sorprendas si tu ratón empieza a acumular polvo.

## ABOUT THE AUTHOR

William E. Shotts, Jr., ha sido un profesional del software y un ávido usuario de Linux durante más de 15 años. Tiene una amplia experiencia en desarrollo de software, incluido el soporte técnico, el control de calidad y la documentación. También es el creador de LinuxCommand.org, un sitio de educación y defensa de Linux que ofrece noticias, reseñas y un amplio soporte para el uso de la línea de comandos de Linux.



REVIEWERS

EL FÍNEST IN GEEK ENTERTAINMENT™

[www.nostarch.com](http://www.nostarch.com)

"*IT IS FLAT.*"

ISBN: 978-1-59327-389-7

9 781593 273897

5 4 9 9 5

6 89145 73894 0

**49,95 \$ (52,95 dólares canadienses)**

