## Parte I: Programación avanzada de UNIX con Linux



# Capítulo 1. Introducción

Este capítulo le muestra cómo realizar los pasos básicos necesarios para crear un programa Linux en C o C ++. En particular, este capítulo le muestra cómo crear y modificar código fuente C y C ++, compilar ese código y depurar el resultado. Si ya está acostumbrado a programar en Linux, puede pasar aCapítulo2, "Escritura de buen software GNU / Linux"; preste mucha atención a Sección 2.3, "Escritura y uso de bibliotecas", para obtener información sobre enlaces estáticos versus dinámicos que quizás no conozca.

A lo largo de este libro, asumiremos que está familiarizado con los lenguajes de programación C o C ++ y las funciones más comunes en la biblioteca C estándar. Los ejemplos de código fuente de este libro están en C, excepto cuando se demuestra una característica o complicación particular de la programación en C ++. También asumimos que sabe cómo realizar operaciones básicas en el shell de comandos de Linux, como crear directorios y copiar archivos. Debido a que muchos programadores de Linux comenzaron a programar en el entorno de Windows, ocasionalmente señalaremos similitudes y contrastes entre Windows y Linux.

## 1.1 Editando con Emacs

Un *editor* es el programa que utiliza para editar el código fuente. Hay muchos editores diferentes disponibles para Linux, pero el editor más popular y con todas las funciones probablemente sea GNU Emacs.

## Sobre Emacs

Emacs es mucho más que un editor. Es un programa increíblemente poderoso, tanto que en CodeSourcery, se lo conoce cariñosamente como One True Program, o simplemente OTP para abreviar. Puede leer y enviar correo electrónico desde Emacs, y puede personalizar y ampliar Emacs de formas demasiado numerosas para discutirlas aquí. ¡Incluso puede navegar por la Web desde dentro de Emacs!

Si está familiarizado con otro editor, ciertamente puede usarlo en su lugar. Nada en el resto de este libro depende del uso de Emacs. Si aún no tiene un editor de Linux favorito, debe seguir el mini-tutorial que se proporciona aquí.

Si le gusta Emacs y desea aprender sobre sus características avanzadas, podría considerar leer uno de los muchos libros de Emacs disponibles. Un excelente tutorial,*Aprendiendo GNU Emacs,* está escrito por Debra Cameron, Bill Rosenblatt y Eric S. Raymond (O'Reilly, 1996).

### 1.1.1 Apertura de un archivo fuente C o C ++

Puede iniciar Emacs escribiendo emacs en la ventana de su terminal y presionando la tecla Retorno. Cuando se ha iniciado Emacs, puede usar los menús en la parte superior para crear un nuevo archivo fuente. Haga clic en el menú Archivos, elija Abrir archivos y luego escriba el nombre del archivo que desea abrir en el "minibúfer" en la parte inferior de la pantalla.[1]Si desea crear un archivo fuente en C, use un nombre de archivo que termine en .C o .h. Si desea crear un archivo fuente C ++, use un nombre de archivo que termine en .cpp, .hpp, . cxx, .hxx, .C, o .H. Cuando el archivo está abierto, puede escribir como lo haría en cualquier programa de procesamiento de texto ordinario. Para guardar el archivo, elija la entrada Guardar búfer en el menú Archivos. Cuando termine de usar Emacs, puede elegir la opción Salir de Emacs en el menú Archivos.

[1] Si no está ejecutando en un sistema XWindow, tendrá que presionar F10 para acceder a los menús.

Si no le gusta apuntar y hacer clic, puede usar atajos de teclado para abrir archivos, guardar archivos y salir de Emacs automáticamente. Para abrir un archivo, escribaCx Cf. (los Cx significa mantener presionada la tecla Control y luego presionar la X .) Para guardar un archivo, escriba Cx Cs. Para salir de Emacs, simplemente escriba Cx Cc. Si desea familiarizarse un poco mejor con Emacs, elija la entrada Tutorial de Emacs en el menú Ayuda. El tutorial le brinda muchos consejos sobre cómo usar Emacs de manera efectiva.

## 1.1.2 Formateo automático

Si está acostumbrado a programar en un *Entorno de desarrollo integrado (IDE),* también estará acostumbrado a que el editor le ayude a formatear su código. Emacs puede proporcionar el mismo tipo de funcionalidad. Si abre un archivo fuente C o C ++, Emacs se da cuenta automáticamente de que el archivo contiene código fuente, no solo texto ordinario. Si presiona la tecla Tab en una línea en blanco, Emacs mueve el cursor a un punto apropiadamente sangrado. Si presiona la tecla Tab en una línea que ya contiene texto, Emacs sangra el texto. Entonces, por ejemplo, suponga que ha escrito lo siguiente:

```
int main ()
{
printf ("Hola, mundo \ n"); }
```

Si presiona la tecla Tab en la línea con la llamada a printf, Emacs reformateará su código para que se vea así:

```
int main ()
```

```
{
    printf ("Hola, mundo \ n");
}
```

Observe cómo la línea se ha sangrado adecuadamente.

A medida que utilice más Emacs, verá cómo puede ayudarle a realizar todo tipo de tareas de formato complicadas. Si es ambicioso, puede programar Emacs para que realice literalmente cualquier tipo de formateo automático que pueda imaginar. La gente ha utilizado esta función para implementar modos de Emacs para editar casi todo tipo de documento, para implementar juegos[2] e implementar interfaces de base de datos.

[2] Intenta ejecutar el comando Mx Dunnet si quieres jugar a un juego de aventuras de texto a la antigua.

### 1.1.3 Resaltado de sintaxis

Además de formatear su código, Emacs puede facilitar la lectura de código C y C ++ coloreando diferentes elementos de sintaxis. Por ejemplo, Emacs puede convertir las palabras clave en un color, tipos integrados como En t otro color, y comenta otro color. El uso del color hace que sea mucho más fácil detectar algunos errores de sintaxis comunes.

La forma más sencilla de activar la coloración es editar el archivo ~ /.emacs e inserte la siguiente cadena:

```
(modo de bloqueo de fuente global t)
```

Guarde el archivo, salga de Emacs y reinicie. ¡Ahora abra un archivo fuente C o C ++ y disfrute!

Es posible que haya notado que la cuerda que insertó en su .emacs parece un código del lenguaje de programación LISP. Eso es porque es*es* Código LISP! Gran parte de Emacs está escrito en LISP. Puede agregar funcionalidad a Emacs escribiendo más código LISP.

## 1.2 Compilación con GCC

A *compilador* convierte el código fuente legible por humanos en código objeto legible por máquina que realmente se puede ejecutar. Los compiladores de elección en los sistemas Linux son todos parte de la Colección de compiladores GNU, generalmente conocida como GCC.[3] GCC también incluye compiladores para C, C ++, Java, Objective-C, Fortran y Chill. Este libro se centra principalmente en la programación C y C ++.

[3] Para obtener más información sobre GCC, visite http://gcc.gnu.org.

Suponga que tiene un proyecto como el de Listado 1.2 con un archivo fuente C ++ (reciprocal.cpp) y un archivo fuente en C (C Principal) como en Listado 1.1. Se supone que estos dos archivos se compilan y luego se vinculan para producir un programa llamadorecíproco. [4] Este programa calculará el recíproco de un número entero.

[4] En Windows, los ejecutables suelen tener nombres que terminan en .exe. Los programas de Linux, por otro lado, generalmente no tienen extensión. Entonces, el equivalente de Windows de este programa probablemente se llamaríareciprocal.exe; la versión de Linux es sencilla recíproco.

***Listado 1.1 (main.c) Archivo fuente C — main.c***

```
# incluye <stdio.h>
# incluye "reciprocal.hpp"

int main (int argc, char ** argv) {

    int i;
```

```
    i = atoi (argv [1]);
    printf ("El recíproco de% d es% g \ n", i, recíproco (i)); return 0;

}
```

***Listado 1.2 (reciprocal.cpp) Archivo fuente de C ++ — reciprocal.cpp***

```
# include <cassert>
# incluye "reciprocal.hpp"

doble recíproco (int i) {
    // Debería ser distinto de cero.
    afirmar (i! = 0);
    return 1.0 / i;
}
```

También hay un archivo de encabezado llamado reciprocal.hpp (ver Listado 1.3).

***Listado 1.3 (reciprocal.hpp) Archivo de encabezado — reciprocal.hpp***

```
# ifdef __cplusplus
extern "C" {
# terminara si

externo doble recíproco (int i);

# ifdef __cplusplus
}
# terminara si
```

El primer paso es convertir el código fuente de C y C ++ en código objeto.

### 1.2.4 Compilación de un archivo de origen único

El nombre del compilador de C es gcc. Para compilar un archivo fuente en C, usa el -C opción. Entonces, por ejemplo, ingresar esto en el símbolo del sistema compila elC Principal archivo fuente:

```
% gcc -c main.c
```

El archivo de objeto resultante se llama main.o.

El compilador de C ++ se llama g ++. Su funcionamiento es muy similar al gcc; compilando reciprocal.cpp se logra ingresando lo siguiente:

```
% g ++ -c reciprocal.cpp
```

los -C la opción dice g ++ para compilar el programa en un archivo objeto solamente; sin ello,g ++ intentará vincular el programa para producir un ejecutable. Una vez que haya escrito este comando, tendrá un objeto archivo llamado recíproco. o.

Probablemente necesitará un par de opciones más para construir cualquier programa razonablemente grande. los-I La opción se usa para decirle a GCC dónde buscar archivos de encabezado. De forma predeterminada, GCC busca en el directorio actual y en los directorios donde están instaladas las cabeceras de las bibliotecas estándar. Si necesita incluir archivos de encabezado de otro lugar, necesitará el-I opción. Por ejemplo, suponga que su proyecto tiene un directorio llamadosrc, para archivos de origen, y otro llamado incluir. Tú compilaras reciprocal.cpp así para indicar que g ++ debería usar el ../incluir directorio además de encontrar reciprocal.hpp:

```
% g ++ -c -I ../include reciprocal.cpp
```

A veces querrá definir macros en la línea de comando. Por ejemplo, en el código de producción, no desea que la sobrecarga de la verificación de aserción esté presente enreciprocal.cpp; eso solo está ahí para ayudarte a depurar el programa. Desactivas el cheque definiendo la macro.NDEBUG. Podrías agregar un explícito #definir para reciprocal.cpp, pero eso requeriría cambiar la fuente en sí. Es más fácil de definir simplementeNDEBUG en la línea de comando, así:

```
% g ++ -c -D NDEBUG reciprocal.cpp
```

Si hubieras querido definir NDEBUG a algún valor particular, podría haber hecho algo como esto:

```
% g ++ -c -D NDEBUG = 3 reciprocal.cpp
```

Si realmente está creando código de producción, probablemente desee que GCC optimice el código para que se ejecute lo más rápido posible. Puede hacer esto usando el-O2 opción de línea de comandos. (GCC tiene varios niveles diferentes de optimización; el segundo nivel es apropiado para la mayoría de los programas). Por ejemplo, las siguientes compilacionesreciprocal.cpp con la optimización activada:

```
% g ++ -c -O2 reciprocal.cpp
```

Tenga en cuenta que compilar con optimización puede hacer que su programa sea más difícil de depurar con un depurador (consulte Sección 1.4, "Depuración con GDB"). Además, en ciertos casos, la compilación con optimización puede descubrir errores en su programa que no se manifestaron previamente.

Puede pasar muchas otras opciones a gcc y g ++. La mejor manera de obtener una lista completa es ver la documentación en línea. Puede hacer esto escribiendo lo siguiente en su símbolo del sistema:

```
% info gcc
```

### 1.2.5 Vincular archivos de objeto

Ahora que has compilado C Principal y utilities.cpp, querrás vincularlos. Siempre deberías usarg ++ para vincular un programa que contiene código C ++, incluso si también contiene código C. Si su programa contiene solo código C, debe usargcc en lugar de. Debido a que este programa contiene tanto C como C ++, debe usarg ++, como esto:

```
% g ++ -o recíproco principal.o recíproco.o
```

los -o La opción proporciona el nombre del archivo que se generará como resultado del paso de enlace. Ahora puedes correr recíproco como esto:

```
% ./ recíproco 7
El recíproco de 7 es 0.142857
```

Como se puede ver, g ++ se ha vinculado automáticamente en la biblioteca de tiempo de ejecución estándar de C que contiene la implementación de printf. Si hubiera necesitado vincular en otra biblioteca (como un juego de herramientas de interfaz gráfica de usuario), habría especificado la biblioteca con el -l opción. En Linux, los nombres de las bibliotecas casi siempre comienzan conlib. Por ejemplo, la biblioteca del Módulo de autenticación conectable (PAM) se llama libpam.a. Para vincular libpam.a, usas un comando como este:

```
% g ++ -o recíproco principal.o recíproco.o -lpam
```

El compilador agrega automáticamente el lib prefijo y el .a sufijo.

Al igual que con los archivos de encabezado, el vinculador busca bibliotecas en algunos lugares estándar, incluido el /lib y / usr / lib directorios que contienen las bibliotecas estándar del sistema. Si desea que el enlazador busque también en otros directorios, debe utilizar la-L opción, que es el paralelo de la -I opción discutida anteriormente. Puede utilizar esta línea para indicar al vinculador que busque bibliotecas en el / usr / local / lib / pam directorio antes de buscar en los lugares habituales:

% g ++ -o recíproco main.o recíproco.o -L / usr / local / lib / pam -lpam

Aunque no tiene que usar el -I para que el preprocesador busque en el directorio actual, debe usar la -L opción para que el enlazador busque en el directorio actual. En particular, puede utilizar lo siguiente para indicarle al vinculador que busque elprueba biblioteca en el directorio actual:

% gcc -o aplicación aplicación. o -L. -ltest

# 1.3 Automatizar el proceso con GNU Make

Si está acostumbrado a programar para el sistema operativo Windows, probablemente esté acostumbrado a trabajar con un entorno de desarrollo integrado (IDE). Agrega archivos de origen a su proyecto y luego el IDE construye su proyecto automáticamente. Aunque los IDE están disponibles para Linux, este libro no los analiza. En cambio, este libro le muestra cómo usar GNU Make para recompilar automáticamente su código, que es lo que la mayoría de los programadores de Linux hacen en realidad.

La idea básica detrás hacer es simple. Tu diceshacer qué *objetivos* quieres construir y luego dar *normas* explicando cómo construirlos. También especificas*dependencias* que indican cuándo se debe reconstruir un objetivo en particular.

En nuestra muestra recíproco proyecto, hay tres objetivos obvios: recíproco. o, main.o, y el recíproco sí mismo. Ya tiene reglas en mente para construir estos objetivos en la forma de las líneas de comando dadas anteriormente. Las dependencias requieren un poco de reflexión. Claramente, recíproco depende de recíproco. o y main.o porque no puede vincular el programa completo hasta que haya creado cada uno de los archivos objeto. Los archivos de objeto deben reconstruirse siempre que cambien los archivos de origen correspondientes. Hay un giro más en ese cambio areciprocal.hpp también debería hacer que ambos archivos de objeto se reconstruyan porque ambos archivos de origen incluyen ese archivo de encabezado.

Además de los objetivos obvios, siempre debe haber un limpio objetivo. Este objetivo elimina todos los programas y archivos de objetos generados para que pueda comenzar de nuevo. La regla para este objetivo usa larm comando para eliminar los archivos.

Puedes transmitir toda esa información a hacer poniendo la información en un archivo llamado Makefile. Esto es lo que Makefile contiene:

```
recíproco: main.o reciprocal.o
	g ++ $ (CFLAGS) -o recíproco principal.o recíproco.o

main.o: main.c reciprocal.hpp
	gcc $ (CFLAGS) -c main.c

reciprocal.o: reciprocal.cpp reciprocal.hpp
	g ++ $ (CFLAGS) -c reciprocal.cpp

limpio:
	rm -f * .o recíproco
```

Puede ver que los destinos se enumeran a la izquierda, seguidos de dos puntos y luego cualquier dependencia. La regla para construir ese objetivo está en la siguiente línea. (Ignora elPSCFLAGS) por el momento). La línea con la regla debe comenzar con un carácter de tabulación, o hacer se confundirá. Si edita su Makefile en Emacs, Emacs le ayudará con el formateo.

Si elimina los archivos de objeto que ya ha creado y simplemente escribe

```
% hacer
```

en la línea de comandos, verá lo siguiente:

```
% hacer
gcc -c main.c
g ++ -c reciprocal.cpp
g ++ -o recíproco principal.o recíproco.o
```

Puedes ver eso hacer ha creado automáticamente los archivos de objeto y luego los ha vinculado. Si ahora cambiasC Principal de alguna manera trivial y tipo hacer de nuevo, verá lo siguiente:

```
% hacer
gcc -c main.c
g ++ -o recíproco principal.o recíproco.o
```

Puedes ver eso hacer sabía reconstruir main.o y volver a vincular el programa, pero no se molestó en volver a compilar reciprocal.cpp porque ninguna de las dependencias para recíproco. o había cambiado.;

los PSCFLAGS) es un hacer variable. Puede definir esta variable en elMakefile en sí mismo o en la línea de comando. ÑUhacer sustituirá el valor de la variable cuando ejecute la regla. Entonces, por ejemplo, para volver a compilar con la optimización habilitada, haría esto:

```
% hacer limpia
rm -f * .o recíproco% hace
CFLAGS = -O2
gcc -O2 -c main.c
g ++ -O2 -c reciprocal.cpp
g ++ -O2 -o recíproco principal.o recíproco.o
```

Tenga en cuenta que el -O2 se insertó una bandera en lugar de PSCFLAGS) en las reglas.

En esta sección, ha visto solo las capacidades más básicas de hacer. Puede obtener más información escribiendo esto:

```
% info make
```

En ese manual, encontrará información sobre cómo hacer que el mantenimiento de un Makefile más fácil, cómo reducir la cantidad de reglas que necesita escribir y cómo calcular automáticamente las dependencias. También puede encontrar más información en*GNU, Autoconf, Automake y Libtool* por Gary V.Vaughan, Ben Elliston, Tom Tromey e Ian Lance Taylor (New Riders Publishing, 2000).

# 1.4 Depuración con GNU Debugger (GDB)

los *depurador* es el programa que utiliza para averiguar por qué su programa no se comporta de la manera que cree que debería. Harás mucho esto.El depurador GNU (GDB) es el depurador utilizado por la mayoría de los programadores de Linux. Puede usar GDB para recorrer su código, establecer puntos de interrupción y examinar el valor de las variables locales.

[5]. ... A menos que sus programas funcionen siempre la primera vez.

### 1.4.1 Compilación con información de depuración

Para usar GDB, deberá compilar con la información de depuración habilitada. Haga esto agregando el-gramo encienda la línea de comando de compilación. Si está usando unMakefile como se describió anteriormente, puede configurar CFLAGS igual a -gramo cuando corres hacer, como se muestra aquí:

```
% hace CFLAGS = -g
gcc -g -c main.c
g ++ -g -c reciprocal.cpp
g ++ -g -o recíproco principal.o recíproco.o
```

Cuando compila con -gramo, el compilador incluye información adicional en los archivos objeto y ejecutables. El depurador utiliza esta información para averiguar qué direcciones corresponden a qué líneas en qué archivos de origen, cómo imprimir las variables locales, etc.

### 1.4.2 Ejecución de GDB

Puedes empezar gdb escribiendo:

```
% gdb recíproco
```

Cuando gdb se inicia, debería ver el indicador de GDB:

```
(gdb)
```

El primer paso es ejecutar su programa dentro del depurador. Solo ingresa el comandocorrer y cualquier argumento del programa. Intente ejecutar el programa sin ningún argumento, como este:

```
(gdb) ejecutar
Programa de inicio: recíproco

Programa recibido señal SIGSEGV, Fallo de segmentación. __strtol_internal
(nptr = 0x0, endptr = 0x0, base = 10, group = 0) en strtol.c: 287

287 strtol.c: No existe tal archivo o directorio. (gdb)
```

El problema es que no hay un código de verificación de errores en principal. El programa espera un argumento, pero en este caso el programa se ejecutó sin argumentos. losSIGSEGV El mensaje indica un bloqueo del programa. GDB sabe que el bloqueo real ocurrió en una función llamada__strtol_internal. Esa función está en la biblioteca estándar y la fuente no está instalada, lo que explica el mensaje "No existe ese archivo o directorio". Puedes ver la pila usando eldónde mando:

```
(gdb) donde
# 0 __strtol_internal (nptr = 0x0, endptr = 0x0, base = 10, group = 0) en strtol.c: 287

# 1 0x40096fb6 en atoi (nptr = 0x0) en ../stdlib/stdlib.h:251
# 2 0x804863e en main (argc = 1, argv = 0xbffff5e4) en main.c: 8
```

Puede ver en esta pantalla que principal llamó al atoi funcionar con un NULO puntero, que es la fuente del problema.

Puedes subir dos niveles en la pila hasta llegar principal usando el hasta mando:

```
(gdb) hasta 2
# 2 0x804863e en main (argc = 1, argv = 0xbffff5e4) en main.c: 8 8 i = atoi (argv
[1]);
```

Tenga en cuenta que gdb es capaz de encontrar la fuente de C Principal, y muestra la línea donde ocurrió la llamada de función errónea. Puede ver el valor de las variables usando elimpresión mando:

```
(gdb) imprimir argv [1]
$ 2 = 0x0
```

Eso confirma que el problema es de hecho un NULO puntero pasado a atoi.

Puede establecer un punto de interrupción mediante el rotura mando:

```
(gdb) romper principal
Punto de interrupción 1 en 0x804862e: archivo main.c, línea 8.
```

Este comando establece un punto de interrupción en la primera línea de principal. Ahora intente volver a ejecutar el programa con un argumento, como este:

[6] Algunas personas han comentado que dicho romper principal es un poco divertido porque normalmente solo quieres hacer esto cuando principal ya está roto.

```
(gdb) ejecutar 7
Programa de inicio: recíproco 7

Punto de interrupción 1, principal (argc = 2, argv = 0xbffff5e4) en main.c: 8
8 i = atoi (argv [1]);
```

Puede ver que el depurador se ha detenido en el punto de interrupción.

Puede pasar por alto la llamada aatoi utilizando el Siguiente mando:

```
(gdb) siguiente
9               printf ("El recíproco de% d es% g \ n", i, recíproco (i));
```

Si quieres ver lo que pasa adentro recíproco, utilizar el paso comando como este:

```
(gdb) paso
recíproco (i = 7) en recíproco.cpp: 6 6 aseverar (i!
= 0);
```

Ahora estás en el cuerpo del recíproco función.

Puede que le resulte más conveniente ejecutar gdb desde dentro de Emacs en lugar de usar gdb directamente desde la línea de comando. Usa el comandoMx gdb comenzar gdb en una ventana de Emacs. Si se detiene en un punto de interrupción, Emacs automáticamente extrae el archivo fuente apropiado. Es más fácil averiguar qué está sucediendo cuando mira el archivo completo en lugar de solo una línea de texto.

## 1.5 Búsqueda de más información

Casi todas las distribuciones de Linux vienen con una gran cantidad de documentación útil. Puede aprender la mayor parte de lo que hablaremos en este libro leyendo la documentación de su distribución de Linux (aunque probablemente le llevará mucho más tiempo). Sin embargo, la documentación no siempre está bien organizada, por lo que la parte complicada es encontrar lo que necesita. La documentación a veces también está desactualizada, así que tome todo lo que lea con un grano de sal. Si el sistema no se comporta

camino a *página de manual*páginas de manual) dice que debería, por ejemplo, puede ser que la página de manual esté desactualizada.

Para ayudarlo a navegar, aquí están las fuentes de información más útiles sobre la programación avanzada de Linux.

#### 1.5.1 Páginas de manual

Las distribuciones de Linux incluyen páginas de manual para la mayoría de los comandos estándar, llamadas al sistema y funciones de biblioteca estándar. Las páginas de manual se dividen en secciones numeradas; para los programadores, los más importantes son estos:

(1) Comandos de usuario

(2) Llamadas al sistema

(3) Funciones de biblioteca estándar

(8) Comandos administrativos / del sistema

Los números denotan secciones de la página de manual. Las páginas de manual de Linux vienen instaladas en su sistema; utilizar el hombre comando para acceder a ellos. Para buscar una página de manual, simplemente invocahombre *nombre* , dónde *nombre* es un comando o nombre de función. En algunos casos, el mismo nombre aparece en más de una sección; puede especificar la sección explícitamente colocando el número de sección antes del nombre. Por ejemplo, si escribe lo siguiente, obtendrá la página de manual de ladormir comando (en la sección 1 de las páginas de manual de Linux):

```
% hombre duerme
```

Para ver la página de manual de dormir función de biblioteca, use este comando:

```
% hombre 3 duerme
```

Cada página de manual incluye un resumen de una línea del comando o función. losque es *nombre* comando muestra todas las páginas de manual (en todas las secciones) para un comando o función que coincida *nombre* . Si no está seguro de qué comando o función desea, puede realizar una búsqueda por palabra clave en las líneas de resumen, utilizando hombre -k *palabra clave* .

Las páginas de manual incluyen mucha información muy útil y deberían ser el primer lugar al que acudir para obtener ayuda. La página de manual de un comando describe las opciones y los argumentos de la línea de comandos, la entrada y la salida, los códigos de error, la configuración y similares. La página de manual de una función de biblioteca o llamada al sistema describe los parámetros y los valores de retorno, enumera los códigos de error y los efectos secundarios, y especifica qué archivo de inclusión se utilizará si llama a la función.

#### 1.5.2 Información

El sistema de documentación Info contiene documentación más detallada para muchos componentes centrales del sistema GNU / Linux, además de varios otros programas. Las páginas de información son documentos de hipertexto, similares a las páginas web. Para iniciar el navegador de información basado en texto, simplemente escribainfo en una ventana de concha. Se le presentará un menú de documentos de información instalados en su sistema. (Presione Control + H para mostrar las teclas para navegar por un documento de información).

Entre los documentos de información más útiles se encuentran estos:

- gcc— El compilador gcc
- libc— La biblioteca GNU C, incluidas muchas llamadas al sistema
- gdb— El depurador GNU
- emacs— El editor de texto de Emacs
- info— El propio sistema de información

Casi todas las herramientas de programación estándar de Linux (incluidas ld, el enlazador; como, el ensamblador; y gprof, el generador de perfiles) vienen con páginas de información útiles. Puede saltar directamente a un documento de información en particular especificando el nombre de la página en la línea de comando:

```
% info libc
```

Si realiza la mayor parte de su programación en Emacs, puede acceder al navegador de información integrado escribiendo Mx info o Ch i.

#### 1.5.3 Archivos de encabezado

Puede aprender mucho sobre las funciones del sistema que están disponibles y cómo usarlas mirando los archivos de encabezado del sistema. Estos residen en/usr / incluir y /usr / include / sys. Si está obteniendo errores de compilación al usar una llamada al sistema, por ejemplo, eche un vistazo en el archivo de encabezado correspondiente para verificar que la firma de la función sea la misma que se muestra en la página del manual.

En los sistemas Linux, muchos de los detalles esenciales de cómo funcionan las llamadas al sistema se reflejan en los archivos de encabezado de los directorios /usr / include / bits, /usr / include / asm, / usr / include / linux. Por ejemplo, los valores numéricos de las señales (descritos en Sección 3.3, "Señales", en Capítulo3, "Procesos") se definen en /usr / include / bits / signum.h. Estos archivos de encabezado son una buena lectura para las mentes inquisitivas. Sin embargo, no los incluya directamente en sus programas; siempre use los archivos de encabezado en/usr / incluir o como se menciona en la página de manual de la función que está utilizando.

### 1.5.4 Código fuente

Esto es de código abierto, ¿verdad? El árbitro final de cómo funciona el sistema es el código fuente del sistema en sí, y afortunadamente para los programadores de Linux, ese código fuente está disponible gratuitamente. Lo más probable es que su distribución de Linux incluya el código fuente completo para todo el sistema y todos los programas incluidos en él; si no, tiene derecho, según los términos de la Licencia Pública General GNU, a solicitarlo al distribuidor. (Sin embargo, es posible que el código fuente no esté instalado en su disco. Consulte la documentación de su distribución para obtener instrucciones sobre cómo instalarlo).

El código fuente del propio kernel de Linux generalmente se almacena en /usr / src / linux. Si este libro le deja sediento de detalles sobre cómo funcionan los procesos, la memoria compartida y los dispositivos del sistema, siempre puede aprender directamente del código fuente. La mayoría de las funciones del sistema descritas en este libro se implementan en la biblioteca GNU C; consulte la documentación de su distribución para conocer la ubicación del código fuente de la biblioteca C.