Manejo de terminales de texto

Programación para Ingeniería Eléctrica

March 12, 2014

El entorno de desarrollo a utilizar en el curso consiste en un sistema operativo de tipo GNU/Linux y un conjunto de herramientas para programar, generar programas ejecutables a partir del código (compilar) y ejecutar los programas generados dentro de ese sistema.

La filosofía de este curso es minimalista, buscando que el alumno se concentre en lo esencial. Debido a esto, las herramientas a utilizar, así como el entorno de ejecución, serán también las mínimas necesarias. Concretamente, se trabajará con editores de texto sencillos en lugar de entornos complejos de desarrollo; estos últimos suelen ahorrar tiempo a programadores experimentados, pero confunden a los principiantes e incluso evitan que éstos pasen por ciertas etapas fundamentales de lo que hace a la comprensión del proceso de generación de un programa.

Asimismo, en cuanto a la interfaz entre los programas generados y su entorno (incluyendo al usuario), se trabajará exclusivamente en *terminales de texto*, en donde la entrada de datos al programa se hace desde el teclado y la salida de datos se muestra como texto plano en pantalla.

El propósito de este pequeño tutorial es el de familiarizar al estudiante con los conceptos de UNIX, terminal de texto (terminal o consola), el intérprete de comandos (shell), y el sistema de archivos UNIX.

GNU/Linux

Al igual que Windows o Macintosh, Linux es un sistema operativo moderno con todas las facilidades de cualquiera de los otros en términos de comodidad, intuitividad y potencia. De manera similar a Macintosh, Linux deriva del sistema operativo UNIX desarrollado en la década de 1970. Todo lo que refiere al manejo de archivos, recursos, cuentas de usuario y dispositivos de estos sistemas está basado en el diseño original de UNIX.

La gran diferencia de Linux con Windows o Macintosh es que Linux es un sistema operativo *libre*, siendo que puede ser copiado, distribuído y modificado libremente por quien así lo desee. De igual manera, GNU/Linux dispone de un conjunto *enorme* de aplicaciones gratuitas que pueden ser instaladas mediante mecanismos propios del sistema operativo a pedido del usuario. Estas aplicaciones están almacenadas en *repositorios* de software (replicados en todas partes del mundo). Existen distintas variantes (llamadas *distribuciones*) de Linux según la combinación de aplicaciones que son incorporadas en cada distribución, y cómo éstas son configuradas. Un ejemplo muy popular es *Ubuntu*, que a su vez admite un número de variantes. La variante *Lubuntu* por ejemplo provee un Ubuntu minimalista, con foco en la agilidad y el bajo consumo de recursos.

Archivos de texto

Existen numerosas formas de describir documentos de texto de manera digital. Uno de ellos es el HTML de las páginas web, otro es la familia de formatos de texto utilizada por Microsoft Word.

Todos los lenguajes de programación utilizan un formato de documento de texto muy básico denominado *texto plano*. Este formato consiste en una secuencia de bytes (palabras binarias de 8 bits) en donde cada uno de los 256 posibles valores binarios de cada byte se traducen en una letra[1]. La correspondencia entre número binario y letra impresa se define mediante la llamada *codificación de caracteres*. La más antigua es la codificación ASCII, que sólo utiliza 7 bits y que sirve para codificar letras del alfabeto inglés solamente. Luego se definieron variantes para distintos lenguajes o regiones, como la ISO-8859-1 que se utiliza para lenguas latinas.

Es muy importante recalcar que ningún otro formato de documentos (por ejemplo Word) es compatible con las herramientas de desarrollo de ningún lenguaje medianamente conocido, en ninguna plataforma.

[1] Hoy en día los estándares modernos de texto plano, por ejemplo el Unicode, utilizan más de un byte por letra

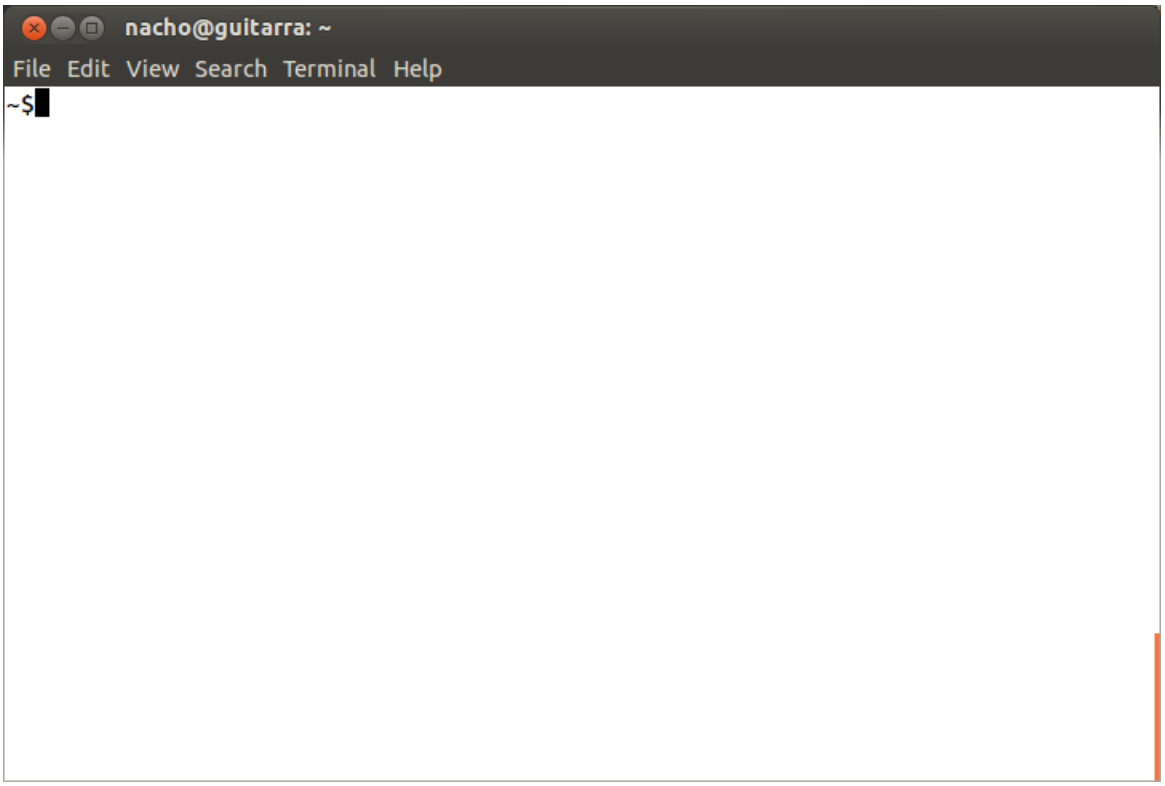

Figure 1: Símbolo del sistema

Terminal de comandos

La forma más popular de interactuar con el usuario hoy en día es a través de *interfaces gráficas* en donde el estado del sistema y los programas se presenta al usuario en forma de ventanas, botones, menúes, que además de ser accedidos por el teclado dependen de dispositivos apuntadores como un mouse o un touchpad. Este concepto fue desarrollado a principios de los años 1980 por Xerox, popularizado luego por Macintosh y finalmente adoptado masivamente con la llegada de Windows 3.1 a principios de los 1990.

Sin embargo, antes de la llegada de las interfaces gráficas, la forma prinicpal de acceso a computadoras era mediante *terminales de texto*. Más aún, este mecanismo está aún hoy disponible para todos los sistemas operativos modernos, y en buena parte muchas de las aplicaciones aún lo utilizan detrás de bambalinas para comunicarse con distintas partes del sistema (ni siquiera Android escapa a esto!).

Es por lo de arriba que trabajar con terminales de comandos es una habilidad requerida aún hoy cuando se trata de desarrollar programas. Además de eso, las terminales de comando proveen una forma muy sencilla e intuitiva de interactuar con el usuario, por lo que facilitan el desarrollo de programas.

Una forma útil de pensar en las terminales de comandos en términos modernos (de 2014) es la de *chatear con la computadora*. Uno le escribe una linea de texto diciendo en ella qué quiere que la computadora haga, la computadora ejecuta la orden, responde con una o más líneas de texto y, cuando termina, vuelve a quedar esperando por más comandos. En ambos casos, el flujo de texto en uno y otro sentido se hace mediante texto plano, tal como se mencionó en el apartado anterior.

Al abrir un nuevo terminal de comandos (en Ubuntu esto suele estar en el menú "Accesorios/Terminal"), se abre un recuadro con una línea que termina usualmente en "$". Este símbolo, llamado *prompt* o *símbolo del sistema*, indica al usuario que la computadora está esperando por un comando. Esto se ve en la Figura 1

El usuario responde entonces ingresando un *comando*. El comando es una secuencia de caracteres terminado en un `ENTER` en donde se especifica, de manera general:

1. La acción a realizar. Esto es en general el nombre de un programa o *comando del sistema*.
2. Opciones. Lista de parámetros opcionales que modifican el funcionamiento básico del programa a ejecutar
3. Argumentos. Datos o nombre de archivo de datos sobre los cuales el programa va a operar.

Las distintas partes de un comando se separan mediante espacios. A modo de ejemplo, el comando `cal`

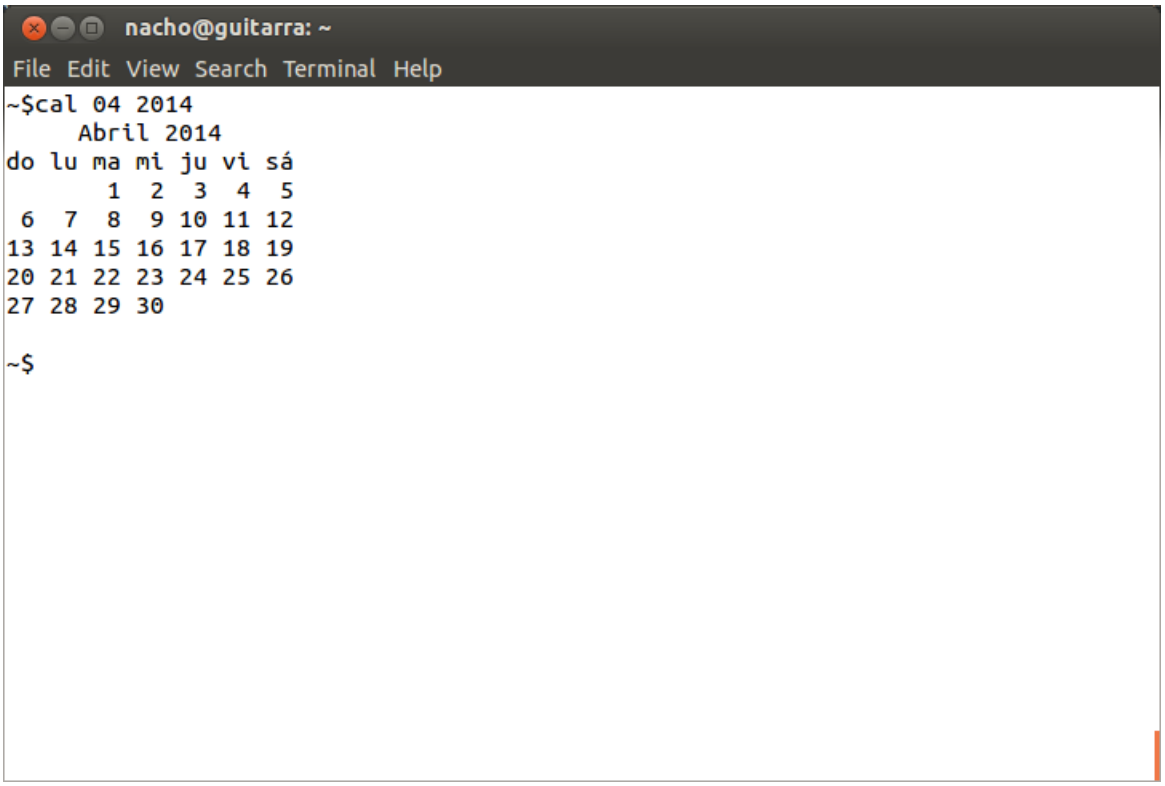

Figure 2: Primer ejemplo de comando. Observar cómo luego de generarse la salida del programa `cal`, el sistema vuelve a solicitar un comando por medio del símbolo del sistema "$".

`04 2014` tiene como resultado una versión en texto plano del calendario de abril de 2014 (Figura 2). Aquí, el comando es `cal`, y los argumentos son dos: mes (04) y año (2014).

En el caso anterior, el resultado de ejecutar el programa `cal` con tales argumentos es la salida por única vez de una representación textual del calendario de abril de 2014. Existen numerosos comandos de esta naturaleza, en donde el único rol del teclado es el de ingresar comandos en el prompt.

El ciclo anterior puede continuar indefinidamente hasta que el usuario desee terminar la sesión de texto cerrando la terminal.

En otros programas de caracter más interactivo, la ejecución del programa puede involucrar la lectura de caracteres desde el teclado. En este caso, el diálogo entre el programa y el usuario se dará mediante el teclado y la pantalla. Para que esto suceda, el sistema operativo cede temporalmente la entrada de la terminal (el teclado) y la salida (la pantalla) al programa hasta que éste termine.

Durante este tipo de interacciones, es necesario que el usuario pueda señalizar el fin del ingreso de datos mediante el teclado. Para ésto, el teclado es capaz de generar, además de letras y símbolos imprimibles, una serie de símbolos especiales que son interpretados de manera especial por la terminal. Entre estos se encuentran por ejemplo el caracter de "fin de archivo", generado por la combinación de teclas `CTRL-D`; tal caracter especial, así como los otros, son parte del estándar ASCII y todos los siguientes, de modo que existen valores binarios específicos asociados a estos caracteres especiales. En la Figura 3 vemos un ejemplo de programa interactivo.

La entrada y salida estándar

En principio, todo programa, cuando es ejecutado en una terminal, es *conectado* al teclado y la pantalla durante su ejecución, de manera que este pueda también interactuar con el usuario. El flujo de caracteres proveniente del teclado es denominado *entrada estándar*, y el flujo de caracteres a ser impresos en pantalla es denominado la *salida estándar* del programa.

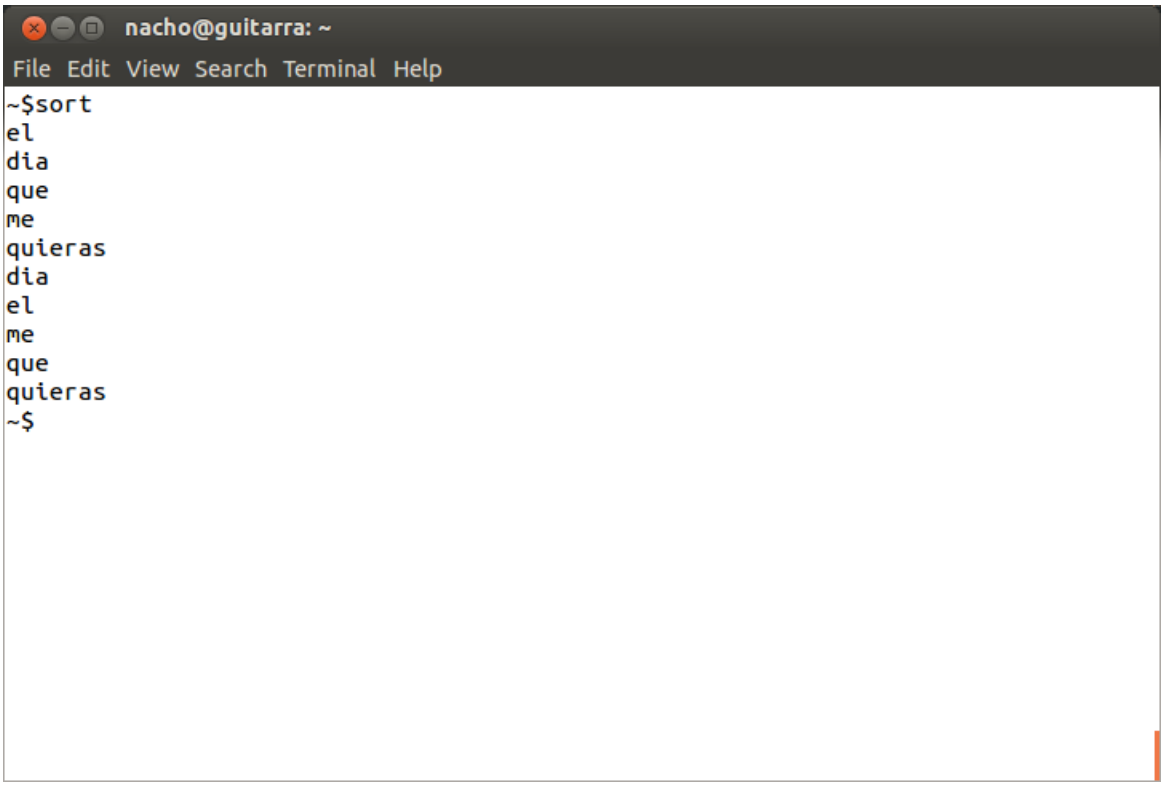

Figure 3: Ejemplo de comando interactivo `sort`. Luego de la ejecución del comando incial, el usuario ingresó un cierto número de líneas de texto (cada una terminadas con `ENTER`) y luego presionó `CTRL-D`. Al recibir éste caracter, el programa `sort` devolvió las mismas líneas ordenadas alfabéticamente.

El shell

El *terminal de texto* es en realidad sólamente responsable de leer el teclado letra por letra e imprimir texto (también letra por letra) en una ventana, de izquierda a derecha, de arriba a abajo.

Cuando uno abre un terminal, éste a su vez invoca a un programa llamado *shell*, que es quien se encarga de recibir comandos de texto del teclado (y mostrarlos en pantalla mientras se escriben, algo llamado *echo*), luego traducirlos en la ejecución de programas, redirigiendo durante ésta la entrada del teclado a la entrada estándar del programa siendo ejecutado y la salida estándar del programa hacia la terminal, para luego volver al prompt. Además de su función básica, el shell dispone de un número muy amplio de facilidades, llegando a ser un software bastante complejo. En Linux hoy en día se utiliza casi exclusivamente el shell *bash*, del cual hablaremos más adelante.

El sistema de archivos

El caso más común de los programas de computadora es que éstos operen sobre archivos de datos, eventualmente generando otros archivos como producto de su ejecución. Es por ello fundamental comprender cómo se identifica un archivo dentro de un medio de almacenamiento de modo que un programa pueda referirse de manera unívoca a un cierto archivo durante su ejecución.

La tarea de asociar un nombre de archivo a un archivo de datos en un medio de almacenamiento es una de las más importantes de todo sistema operativo. Tanto en Windows como en Linux, Mac, iOS, Android, etc., internamente los archivos presentes en medios de almacenamiento se organizan de manera jerárquica en un llamado *árbol de directorios*. Dicha jerarquía comienza en un directorio (o carpeta) *raíz* que en los sistemas como Linux o Mac es único y se identifica con el caracter "/". En Windows cada medio de almacenamiento tiene una raíz distinta identificada con una letra del alfabeto inglés, comenzando típicamente en "C:" por motivos históricos ("A:" y "B:" eran las dos primeras disketteras).

En un árbol de directorios, cada nodo del árbol se corresponde con un directorio, y su contenido son los archivos ubicados en ese directorio. Los nodos hijos de un nodo dado son sus subdirectorios. Salvo el nodo raíz "/", todos los nodos tienen un nodo padre al que se accede mediante el nombre especial "..".

Supongamos el siguiente ejemplo: **/** contiene un subdirectorio **src/** con dos archivos de nombre **pepe.c** y **pepe.h**, otro subdirectorio **bin/** con el archivo **pepe**. El directorio **src/** a su vez tiene una carpeta **contrib/** con el archivo **coco.a**, y otro subdirectorio **old/** con versiones antiguas de **pepe.c** y **pepe.h**. El árbol de directorios se vería de la siguiente manera:

```
/
+-src/
  +-pepe.c
  +-pepe.h
  +-contrib/
    +-coco.a
  +-old/
    +-pepe.h
    +-pepe.c
+-bin/
  +-pepe
```

Para identificar un archivo de manera unívoca en el sistema de archivos no basta con su nombre, por ejemplo **pepe.c**, sino que hay que especificar su ubicación en todo el árbol, o sea, su *nombre completo*. Para determinar el nombre completo de un archivo, se concatenan los nombres de los directorios desde la raíz hasta el nodo que lo contiene, mediante el caracter separador **/**. Para el caso del **pepe.h** nuevo, esto sería **/src/pepe.c**, y para el viejo, **/src/old/pepe.c**. A menudo se denomina a la concatenación de directorios (sin el nombre, aunque a veces también se lo incluye) como *ruta* del archivo (en inglés *path*).

El sistema de archivos y el shell

Un concepto importante al trabajar en un shell es el del "directorio de trabajo". En todo momento, el shell mantiene internamente una posición dentro del árbol de directorios desde la cual puede accederse a los archivos del sistema con una *ruta relativa* a esa posición en lugar de tener que especificar la ruta absoluta de cada archivo. El comando `pwd` permite ver cuál es el directorio de trabajo actual (pruébelo).

Cuando los comandos son ejecutados en un terminal, éstos heredan el directorio de trabajo actual.

Las rutas relativas se especifican tomando como base el directorio de trabajo en lugar de **/**. Por ejemplo, si la ruta de trabajo es **/src**, uno puede referirse al **pepe.c** viejo como **old/pepe.c**. El directorio de trabajo puede mostrarse explícitamente en la ruta relativa como el directorio **.**, de modo que **./old/pepe.c** es equivalente a **old/pepe.c**, en ambos casos refiriéndose a **/src/old/pepe.c**.

Si uno desea acceder a un directorio padre del directorio de trabajo actual, se utiliza el nombre de directorio **..** para referirse a éste. Por ejemplo, si quisiéramos acceder a **/bin/pepe** desde el dir. de trabajo **/src**, podríamos hacerlo a través de la ruta relativa **../bin/pepe**.

Comandos básicos del sistema

Hay numeros tutoriales de UNIX, en todos los idiomas, con descripciones de los comandos más comunes y útiles. Se recomienda hacer una búsqueda en la red al respecto y seguir alguno de ellos. Aquí mencionaremos sólamente un par de ellos (ya vimos `pwd`). El más importante es probablemente `ls`, que permite ver qué archivos hay en un directorio dado. Toma como argumento opcional el nombre de un directorio. En caso omiso utiliza el directorio de trabajo como argumento por defecto.

En el caso anterior, estando en el directorio de trabajo **src**, el ejecutar `ls` y luego `ENTER` en el prompt provocaría la salida

```
pepe.c pepe.h old/
```

Es posible incorporar parámetros opcionales para ver más datos de los archivos (o menos), ordenarlos por fechas, imprimirlos en colores, etc. Se sugiere mirar la *documentación en línea* del comando `ls` para tales opciones pero... de dóde se saca esa documentación?

La respuesta es el comando `man` (de `man`ual. El comando `man` toma como parámetro el nombre de otro comando sobre el cual se desea averiguar, y devuelve un documento interactivo describiendo cómo se utiliza el comando en cuestión. En este caso escribiríamos `man ls` (y `ENTER` obviamente). Ejercicio: pruébelo.

Por último, un comando muy útil es `apropos`. Este comando tiene como finalidad hacer búsquedas en la biblioteca de comandos disponibles en base a palabras clave, como quien busca en un buscador web. Por

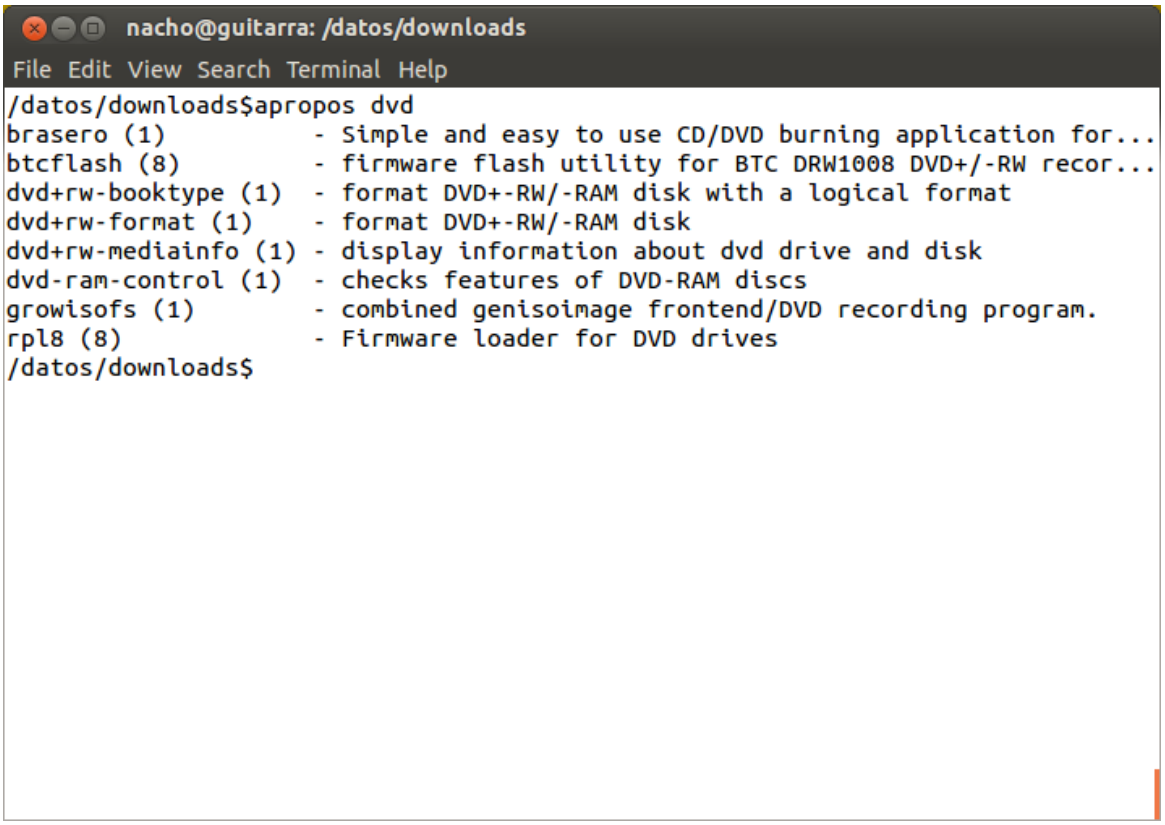

Figure 4: Salida del programa `apropos` con argumento `dvd`.

ejemplo, para saber qué comandos hay relacionados con el manejo de directorios, uno puede ejecutar la orden `apropos directory`, lo cual dará una lista (enorme) de programas que hacen "algo con directorios".

Las figuras 4 y 5 muestran ejemplos de `apropos` y `man`.

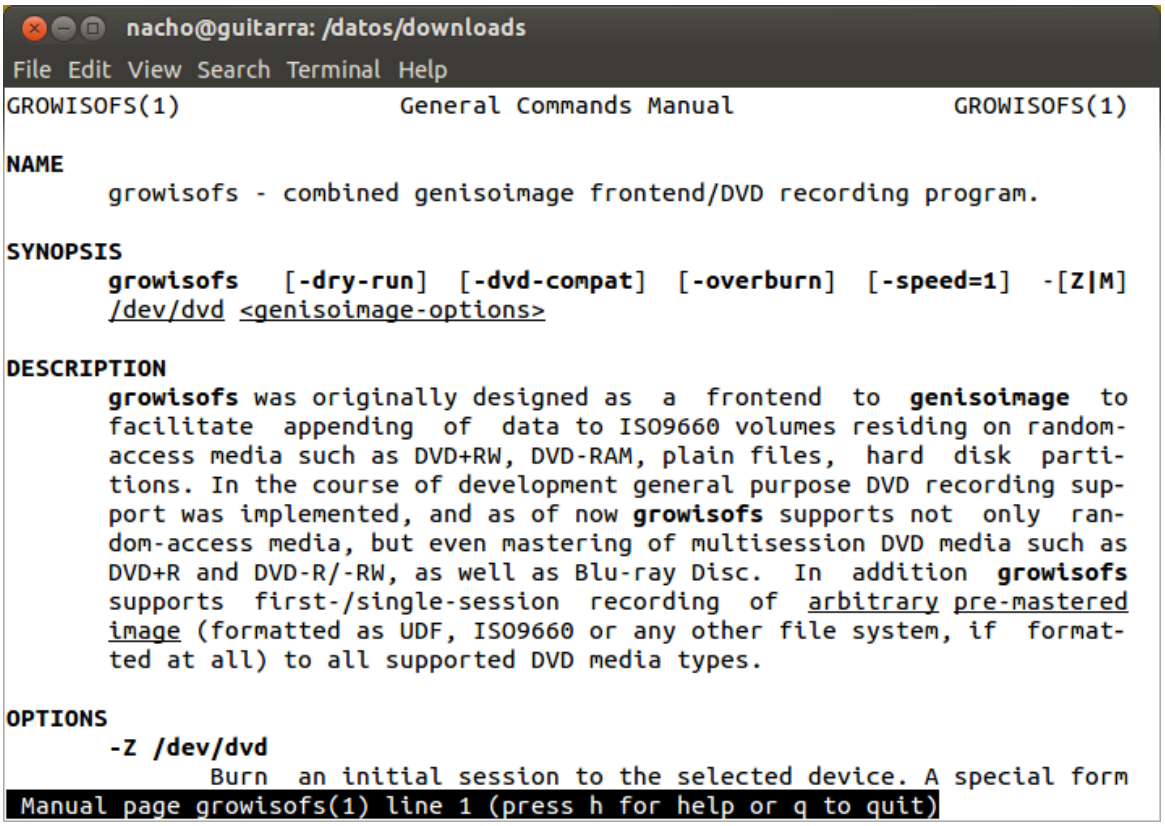

Figure 5: Salida (interactiva) del programa `man` con argumento `growisofs`. Notar que esta es una aplicación interactiva en la cual puede navegarse con el teclado para hacer scroll, ir (y volver) a y desde hipervínculos, hacer búsquedas de texto, y muchas otras cosas más. Para volver al prompt se sale con la tecla “q”.