Uso y Comprensión Avanzados del GNU/Linux

Lucas Di Pentima `lucas@lunix.com.ar`
Nicolás César `nico@lunix.com.ar`

12 de mayo de 2001

Índice General

1 Introducción **1**
- 1.1 Historia de UNIX 1
- 1.2 Versiones de UNIX 1
- 1.3 El Proyecto GNU 1
- 1.4 Linus Torvalds completa el rompecabezas 1
- 1.5 Filosofía *Open Source* y *Free Software* 1
  - 1.5.1 Diferencias 1
  - 1.5.2 Ventajas y desventajas 1
  - 1.5.3 Como colaborar 1

2 Trabajando en modo texto **3**
- 2.1 Entrar en el sistema 3
  - 2.1.1 Introducción 3
  - 2.1.2 Iniciando una sesión 3
  - 2.1.3 La base de datos de los usuarios 4
- 2.2 Comandos básicos 5
  - 2.2.1 Introducción 5
  - 2.2.2 Listado de comandos 6
- 2.3 Permisos de usuario 20
  - 2.3.1 Introducción 20
  - 2.3.2 Conceptos básicos 21
  - 2.3.3 Un caso especial, los directorios 22
  - 2.3.4 Cambio de permisos: el comando `chmod` 22
  - 2.3.5 Representación octal 23
- 2.4 Variables de entorno 23
- 2.5 Redirección de entradas y salidas 24
  - 2.5.1 Entrada, salida y error estándar 24
  - 2.5.2 Tuberías (pipes) 26
  - 2.5.3 Redirección 28
- 2.6 Archivos de configuración del intérprete de comandos `bash` 30
  - 2.6.1 Introduccion 30

2.6.2 .bashrc 30
2.6.3 .profile 30
2.6.4 .bash_profile 30
2.6.5 .bashrc 31
2.6.6 .bash_profile 31
2.7 Programación en shell 31
2.7.1 Introducción 31
2.7.2 Códigos de salida 32
2.7.3 El comando if 33
2.7.4 El comando while 35
2.7.5 El comando for 35

3 Correo Electrónico 39
3.1 Introducción 39
3.2 Uso de clientes de correo 40
3.2.1 Pine 41
3.2.2 Mutt 49
3.3 Protegiendo nuestra privacidad: uso del GnuPG 49
3.3.1 La seguridad del correo electrónico 49
3.3.2 ¿Qué es el GnuPG? 50
3.3.3 Configurando el GnuPG 51
3.3.4 Uso diario del GnuPG 53
3.4 Integración de GPG con pine 57
3.5 Envío de correo automatizado con el comando mail 58
3.6 Herramientas de tratamiento de codificación MIME, UUE, ROT13, etc. 58
3.7 Configuración y uso de fetchmail 58
3.8 Configuración de filtros con procmail 58

4 Edición de archivos 59
4.1 Diferencias entre vi y emacs 59
4.2 Configuración y uso básico del editor vi 59
4.3 Configuración y uso básico del editor emacs 59
4.4 Configuración y uso del editor pico 59
4.5 Sistemas de documentación 59
4.5.1 TeX 59
4.5.2 LaTeX 59
4.5.3 groff 59
4.5.4 Texinfo 59
4.5.5 SGML 59
4.5.6 DocBook 59

5 Redes **61**
5.1 Uso básico de un cliente de FTP 61
5.1.1 ftp 61
5.1.2 ncftp 61
5.2 Uso básico de un cliente de IRC 61
5.2.1 ircii 61
5.2.2 BitchX 61
5.3 Uso básico de un cliente de web 61
5.3.1 lynx 61
5.3.2 links 61

6 Programación **63**
6.1 Conceptos sobre bibliotecas compartidas 64
6.2 Uso de compiladores 64
6.2.1 C 64
6.2.2 C++ 64
6.2.3 Fortran 64
6.2.4 Java 64
6.2.5 Perl 64
6.2.6 Python 64
6.2.7 Configuración del Apache para uso de PHP 64
6.3 Automatización de compilaciones con Makefile, autoconf, etc. . . . 64
6.4 Uso de depuradores 64
6.4.1 gdb 64
6.4.2 xgdb 64
6.4.3 ddd 64

7 Inicio del sistema GNU/Linux **65**
7.1 LILO 65
7.2 Carga del kernel 65
7.3 Scripts de inicio 65

8 Particiones **67**
8.1 ¿Por qué conviene tener más de una sola partición? 67
8.2 Algunas propuestas de esquema de particiones 67

9 Distribuciones **69**
9.1 Distribuciones mejor diseñadas para servidores 69
9.2 Distribuciones mejor diseñadas para estaciones de trabajo 69
9.3 Diferencias básicas 69

10 El comando `su` y el uso de la *fuerza* **71**

11 Tareas administrativas básicas **73**
11.1 Administración de usuarios y grupos 73
11.2 Personalización y compilación del núcleo 73
11.2.1 Introducción 73
11.2.2 Soporte de hardware 74
11.2.3 Actualización del núcleo 76
11.2.4 Compilación del núcleo 77
11.2.5 LILO 86
11.2.6 Módulos 90
11.2.7 Automatizando un poco mas los módulos 93
11.3 Instalación y actualización de paquetes utilizando RPM/DEB . . . 95
11.3.1 ¿Qué paquetes instalar para un servidor promedio? 95
11.3.2 ¿Por qué no debo instalar todo? 95
11.3.3 ¿Qué me conviene para una estación de trabajo promedio? 95
11.4 Programación de tareas con `cron` y `at` 95
11.5 Creación y chequeo de sistemas de archivos 95
11.6 Copias de respaldo, utilidades de archivado y compresión 95

12 Conocimientos básicos de redes **97**
12.1 Nociones de TCP/IP. 97
12.2 Conceptos básicos de interfaz de red y ruteo 97
12.3 Conexiones punto a punto 97
12.3.1 PPP 97
12.3.2 SLIP 97
12.3.3 PLIP 97
12.4 Interfaces ethernet 97
12.5 Conceptos básicos de filtrado de paquetes (ipfwadm, etc...ver kernel 2.4) 97
12.6 Conceptos básicos de enmascaramiento de IPs 97

13 Conceptos y configuración básica del servidor DNS **99**

14 Configuración básica de Samba **101**

15 Consiguración básica de Apache **103**

16 Configuración básica de Squid **105**

17 Configuración de Leafnode/INN. 107
17.1 Leafnode. 107
17.1.1 Introducción. 107
17.1.2 Obtención e Instalación. 107
17.1.3 Configuración. 109
17.1.4 Los filtros. 111
17.2 INN 112
17.2.1 Obtención e instalación 112

18 Configuración de servidor FTP 113

19 Agentes de Transporte de Correos 115
19.1 Introducción 115
19.2 Configuración básica de MTAs 115
19.3 Configuración de herramientas anti-spam 115

Índice de Figuras

2.1 Esquema de entrada y salida estándar del ejemplo 27
2.2 Esquema de entrada y salida estándar del ejemplo 2 27

3.1 Pantalla inicial del `pine` 41
3.2 Pantalla principal del `pine` 43
3.3 Lista de carpetas del `pine` 45
3.4 Índice de mensajes del `pine` 46
3.5 Leyendo mensajes en `pine` 47
3.6 Pantalla de composición de mensajes del `pine` 47
3.7 Seleccionando un archivo a incluir en el mensaje 48
3.8 Agregando un contacto en la libreta de direcciones del `pine` . . . 49
3.9 Configurando `pine` para uso de GPG 58

11.1 Configurando el núcleo con la interfaz de línea de comandos . . . 78
11.2 Interfaz de texto a pantalla completa 79
11.3 Interfaz gráfica para configurar el núcleo 79
11.4 Seleccionando opciones como módulos 91

17.1 Página de Leafnode 108
17.2 Viendo las noticias en knode 112

Índice de Tablas

2.1 Traducción de binario a octal 23

Capítulo 1

Introducción

1.1 Historia de UNIX

1.2 Versiones de UNIX

1.3 El Proyecto GNU

1.4 Linus Torvalds completa el rompecabezas

1.5 Filosofía *Open Source* y *Free Software*

1.5.1 Diferencias

1.5.2 Ventajas y desventajas

1.5.3 Como colaborar

Capítulo 2

Trabajando en modo texto

2.1 Entrar en el sistema

2.1.1 Introducción

Los tipos de trabajos que se pueden realizar sobre un Unix cualquiera, pero particularmente sobre GNU/Linux, difieren en la forma de interactuar con el usuario y el formato de la interfaz de usuario. A simple vista, se puede decir que hay dos tipos de acceso interactivo en lo que respecta al formato de la interfaz: usando interfaz gráfica o usando interfaz de texto. En este curso se le dará especial importancia a la interfaz de textos, ya que es lo mas normal que se encuentra en los equipos que funcionan como servidores, y las herramientas basadas en interfaz de texto tienen mucho mas tiempo de desarrollo, que las otras, lo que las hace mas convenientes para la tarea de administrar un sistema GNU/Linux.

2.1.2 Iniciando una sesión

Existen diferentes métodos para poder conectar los terminales al sistema:

- En primer lugar podemos conectarnos a un GNU/Linux a través de el *puerto serie (RS232)*, con una terminal no inteligente o bien con otro equipo y un emulador de terminales. En ambos casos existe un programa que atiende las solicitudes de conexión a través del puerto serie. Cuando hay una solicitud de conexión, este programa la atiende solicitando al usuario que se identifique ante el sistema. Cuando termina la conexión, este programa se reactiva para seguir atendiendo nuevas solicitudes.

- Mediante *tarjeta de red*. En este caso, tenemos un programa que escucha las solicitudes de conexión a través de la tarjeta de red. Cuando llega una solicitud este programa se desdobla de forma que una parte atiende la conexión y otra continúa atendiendo nuevas conexiones. Así podemos tener más de una conexión a través de la tarjeta de red. Algunos servicios que proveen esta funcionalidad son el `telnet` (sin encriptación de datos) y el `ssh` (Secure Shell, con encriptación de datos). Esto se verá mas adelante.

- La *consola*. Evidentemente, en un sistema GNU/Linux también podemos trabajar desde el teclado y monitor que están conectados directamente al sistema. Normalmente en la mayoría de las distribuciones, en la consola hay hasta 6 terminales virtuales, accediendo a cada una de ellas con `Alt-F1` a `Alt-F6`.

Una vez que se ha conseguido conectar a un sistema GNU/Linux tenemos que iniciar una sesión de trabajo. GNU/Linux es un sistema multiusuario, y esto exige que el usuario se presente al sistema y que este lo acepte como usuario reconocido. Así cada vez que iniciamos una sesión GNU/Linux nos responde con

```
Login:
```

a lo que se debe responder con el nombre de usuario. Acto seguido, GNU/Linux solicita una clave para poder comprobar que el usuario es quien dice ser:

```
Password:
```

En este caso se teclea la clave de acceso. Por motivos de seguridad esta clave no aparecerá en la pantalla. Si la pareja nombre de usuario/clave es correcta el sistema inicia un intérprete de comandos con el que se puede trabajar.

Habitualmente será el símbolo `$`, aunque puede ser también el símbolo `%` (si usamos un shell C). Cuando es el administrador (root) quien está trabajando en el sistema, el indicador que aparece es `#`.

2.1.3 La base de datos de los usuarios

Se ha visto que para iniciar una sesión de trabajo en un sistema GNU/Linux se debe suministrar al sistema una pareja de nombre de usuario/clave. Estos datos se almacenan en un archivo llamado `/etc/passwd`. Este archivo contiene una línea por cada usuario del sistema. Cada línea consta de una serie de campos separados por dos puntos (:). Estos campos son, en el orden que aparecen:

1. **Nombre de usuario**. Es es nombre con el que no presentamos al sistema, con el que tenemos que responder a `Login:` y por el que nos identifica el sistema.

2. **Clave cifrada**. El siguiente campo es la clave de acceso al sistema. Esta clave no se guarda como se introduce, sino que se almacena transformada mediante el algoritmo **DES** para que nadie pueda averiguarla.

3. **UID**. Identificador de usuario. Es el número de usuario que tiene cada cuenta abierta en el sistema. El sistema trabaja de forma interna con el UID, mientras que nosotros trabajamos con el nombre de usuario. Ambos son equivalentes.

4. **GID**. Identificador de grupo. Es el número de grupo principal al que pertenece el usuario.

5. **Nombre completo de usuario**. Este es un campo meramente informativo, en el que se suele poner el nombre completo del usuario.

6. **Directorio personal**. Este campo indica el directorio personal de un usuario, en el cual el usuario puede guardar su información.

7. **Intérprete de comandos**. El último campo indica un programa que se ejecutará cuando el usuario inicie una sesión de trabajo. Normalmente este campo es un intérprete de comandos («shell» en inglés) que proporciona una línea de órdenes para que el usuario trabaje. Ejemplo:

```
usuario:x%6YkH$Ss:505:705:Usuario:/home/usuario:/bin/bash
   ^          ^    ^   ^     ^           ^          ^
   |          |    |   |     |           |          |
   |          |    |   |     |           |  I. de comandos
   |          |    |   |     |      directorio personal
   |          |    |   |  Nombre completo del usuario
   |          |    |  Número de grupo (GID)
   |          |   Número de usuario (UID)
   |       Clave cifrada
 Nombre de usuario
```

2.2 Comandos básicos

2.2.1 Introducción

Existe un conjunto de comandos que todo usuario debe conocer para poder manejarse en un sistema GNU/Linux. La mayoría de estos comandos están relacionados con el manejo de archivos y directorios[1].

[1] Que en realidad son un tipo especial de archivos

Como se ha dicho anteriormente, éstas herramientas tienen un tiempo de desarrollo y prueba mucho mayores que sus símiles en interfaz gráfica, y aunque no lo parezca, una vez familiarizado con el entorno de texto, resulta mas ágil y cómodo para las tareas diarias.

Este grupo de comandos, llamados muchas veces «Caja de herramientas UNIX», posee detrás de cada «herramienta», una filosofía de desarrollo. Cada uno de los comandos, fue creado con dos ideas en mente:

- Debe realizar una sola función.
- Dicha función debe realizarse correctamente.

Con esto en mente, la simpleza de las herramientas permite que puedan combinarse para solucionar diferentes problemas que individualmente no podrían. La forma de combinar estas herramientas se verá mas adelante en la sección 2.5.

Es importante hacer notar, que muchas de las tareas que se pueden hacer con los comandos que se explican en esta sección se pueden realizar con el administrador de archivos `mc`, un clon del *Norton Commander* muy bueno por cierto, pero igualmente no se puede obtener toda la flexibilidad que se tiene con los comandos, como ya se verá mas adelante.

2.2.2 Listado de comandos

A continuación se describirán los comandos mas esenciales necesarios para el resto del curso. Aquellos argumentos que se muestren entre corchetes, se deberán tomar como opcionales.

El comando `cp`

Se utiliza para copiar archivos, su sintaxis es la siguiente:

```
cp [opciones] archivo-origen camino-destino
cp [opciones] archivos-origen... directorio-destino
```

Entre las opciones mas relevantes, se tiene:

-f Borrar los archivos de destino ya existentes.

-p Preservar los permisos, el usuario y el grupo del archivo a copiar.

-R Copia directorios recursivamente.

-a Equivalente a utilizar las opciones `-dpR`

-u No copia un archivo (no directorio) si en el destino ya existe tal archivo, el cual tiene igual tiempo de modificación o mas reciente.

-v Da información en pantalla sobre los archivos que se van copiando.

El comando `mv`

Este comando se usa tanto para mover archivos, como para renombrarlos (que, al fin de cuentas, es una manera de mover archivos), su sintaxis es la siguiente:

```
mv [opción...] origen destino
mv [opción...] origen... destino
```

Si el último argumento, `destino` es un directorio existente, `mv` mueve cada uno de los otros archivos a `destino`. Algunos opciones de este comando son:

-f Borrar los archivos de destino existentes sin preguntar al usuario.

-i Lo contrario de `-f`, pregunta por cada archivo a sobreescribirse antes de hacerlo.

-v Muestra el nombre de cada archivo a ser movido.

El comando `ls`

Quizás uno de los comandos mas utilizados, sirve para listar archivos. Su sintaxis es:

```
ls [opciones] [archivo...]
```

Si se ejecuta `ls` sin argumentos, dará como resultado un listado de todos los archivos (incluyendo directorios) del directorio donde el usuario está posicionado. Sus opciones son:

-a Lista todos los archivos, incluyendo aquellos que comienzan con un «.».

-d Lista el nombre del directorio en vez de los archivos contenidos en él.

-l Lista los archivos con mucho mas detalle, especificando para cada archivo sus permisos, el número de enlaces rígidos, el nombre del propietario, el grupo al que pertenece, el tamaño en bytes, y la fecha de modificación.

-r Invierte el orden de listado de los archivos.

-s Muestra el tamaño de cada archivo en bloques de 1024 bytes a la izquierda del nombre.

-t Lista los archivos ordenados por el tiempo de modificación en vez de ordenarlos alfabéticamente.

-A Lista todos los archivos excepto el «.» y el «..».

-R Lista los contenidos de todos los directorios recursivamente.

-S Ordena el listado por el tamaño de los archivos.

–color=[cuándo] Especifica si emplear color para distinguir los diferentes tipos de archivos. El argumento `cuándo` puede tener varios valores:

- **none** No usar colores. Esta opción es la predeterminada.
- **auto** Usar colores solamente cuando la salida estándar es una terminal.
- **always** Usar siempre colores. Si `ls` se usa con la opción `–color` sin especificar la opción de color, el resultado es el mismo que cuando se usa `–color=always`.

El comando `cd`

Este comando se usa para cambiar de directorio. Generalmente cuando el usuario inicia una sesión en GNU/Linux, el directorio donde comienza es su directorio personal. De ahí uno puede moverse a los diferentes directorios donde se tenga acceso usando este comando. Su sintaxis es la siguiente:

```
cd directorio
```

Éste es un comando interno del intérprete (por ejemplo, `bash`), y no lleva opciones que sean de relevancia como para nombrarlas.

El comando `touch`

Este comando se utiliza para cambiar la fecha de acceso y/o modificación a un archivo. Su sintaxis es la que sigue:

```
touch [opción...] archivo...
```

Si el argumento `archivo` corresponde al nombre de un archivo que no existe, a menos que se le diga, `touch` creará el archivo con dicho nombre y sin ningún contenido. Sus opciones mas importantes son:

-a Cambia solamente el tiempo de acceso.

-c No crear archivos que no existían antes.

-d fecha Usar `fecha` en lugar de la fecha actual. El formato de `fecha` es el siguiente: `MMDDHHMMAAAA`, por ejemplo para representar el 7 de abril de 2001 a la 1:00 a.m., se escribirá: 040701002001. Si el año a usar es el año actual, se puede obviar, entonces el ejemplo anterior quedaría así: 04070100.

Este comando es muy útil cuando se necesita recompilar cierta parte de un programa evitando compilar todo el programa completo, sólo aquellos sectores modificados[2].

El comando `sort`

Este comando se utiliza para ordenar líneas de texto a partir de varios criterios, su sintaxis es similar a la de todos los comandos:

```
sort [opción...] [archivo...]
```

Si no se le provee al menos un argumento `archivo`, este comando tomará su entrada de la entrada estándar, ya veremos esto en la sección 2.5.

El criterio de orden que utiliza `sort` por defecto es alfabético, esto se debe tener en cuenta siempre que se necesite ordenar listas de números, si no se le especifica a `sort` que debe ordenar numéricamente, tomará a los números como una lista de palabras y el resultado no será el deseado. Por ejemplo, alfabéticamente el número 10 está antes que el número 2.

La lista de opciones de `sort` es la siguiente:

-c Chequear si el/los archivos están ordenados, pero no ordenar.

-d Considerar únicamente los caracteres alfanuméricos.

-n Utilizar criterio numérico de ordenamiento.

-o ARCHIVO Escribir el resultado en `ARCHIVO` en lugar de enviarlo a la salida estándar.

-r Devolver el resultado inverso del ordenamiento.

-t SEP Utilizar `SEP` como separador en lugar de un espacio en blanco.

-T DIR Usar `DIR` como directorio temporal en lugar de `/tmp`.

[2] De hecho, en la compilación del núcleo se utiliza

El comando `less`

Este comando es de mucha utilidad, su función es paginar texto en pantalla. Muchas veces ocurre que cuando se ejecuta algún comando, la salida del mismo es demasiada información como para que se pueda leer en la pantalla del monitor, entonces se puede redireccionar esta salida al `less` para que permita al usuario leer sin mayores problemas, pudiendo avanzar o retroceder en el texto con las flechas de cursor del teclado. También se utiliza para visualizar archivos de texto almacenados en disco.

La idea de `less` proviene de un paginador llamado `more`, un clásico en los UNIX. El `more` no era lo suficientemente amigable, es por eso que hicieron `less`. Su sintaxis es la siguiente:

```
less [archivo...]
```

Este comando es un programa interactivo, es por eso que no se hablará de argumentos sino de comandos:

`ESPACIO` Si se oprime la barra espaciadora, el less avanzará un número de líneas igual al número de líneas por pantalla que posea la terminal que se esté usando.

`ENTER` Pulsando la tecla `ENTER` se va avanzando de a una línea.

`G` Ir al final del texto.

`g` Ir al inicio del texto.

`/` Ingresar una palabra a ser buscada avanzando dentro del texto.

`?` Ingresar una palabra a ser buscada retrocediendo dentro del texto.

`n` Buscar la siguiente ocurrencia de la búsqueda.

`AvPág` Avanzar una pantalla de texto.

`RePág` Retroceder una pantalla de texto.

`v` Cargar el editor de texto en el lugar donde se encuentre el usuario dentro del archivo. El editor que normalmente se utiliza es el `vi`, el cual se dará en la sección 4.2.

`q` Salir del programa.

`R` Repintar la pantalla. Útil cuando se está visualizando un archivo que ha sido modificado por otro programa.

El comando `head`

Escribe por salida estándar la primer parte de un archivo. Su sintaxis es como sigue:

```
head [opción...] [archivo...]
```

Si no se especifica el argumento `archivo`, este comando tomará su entrada de la entrada estándar. La lista de opciones mas importantes sigue a continuación:

-c N Escribe los primeros `N` bytes.

-n N Escribe las primeras `N` líneas en vez de las primeras 10 (que es el valor predeterminado).

El comando `tail`

Este comando es al `head` como el `less` es al `more`[3]. El comando `tail` escribe a la salida estándar la última parte de un archivo. Su sintaxis es:

```
tail [opción...] [archivo...]
```

Al igual que `head`, si no se le proporciona un argumento `archivo`, este comando tomará su entrada desde la entrada estándar. Alguna de sus opciones son las siguientes:

-c N Escribe los últimos `N` bytes.

-n N Escribe las últimas `N` líneas.

-f Escribir la última parte del archivo a medida que va creciendo. Esta opción es muy útil para monitorear archivos de registro que van creciendo con el tiempo.

El comando `grep`

Escribir en salida estándar aquellas líneas que concuerden con un patrón. Su sintaxis es como sigue:

```
grep [opciones] PATRÓN [ARCHIVO...]
grep [opciones] [-e PATRÓN | -f ARCHIVO] [ARCHIVO...]
```

[3] Perdonen al autor, aunque a veces es interesante mezclar conceptos matemáticos con informáticos, este no es el caso.

Este comando realiza una búsqueda en los `ARCHIVOs` (o en la entrada estándar, si no se especifica ninguno) para encontrar líneas que concuerden con `PATRÓN`. Por defecto `grep` imprime en pantalla dichas líneas. Sus opciones mas interesantes son:

-c Modifica la salida normal del programa, en lugar de imprimir por salida estándar las líneas coincidentes, imprime la cantidad de líneas que coincidieron en cada archivo.

-e PATRÓN Usar `PATRÓN` como el patrón de búsqueda, muy útil para proteger aquellos patrones de búsqueda que comienzan con el signo «-».

-f ARCHIVO Obtiene los patrones del archivo `ARCHIVO`.

-H Imprimir el nombre del archivo con cada coincidencia.

-r Buscar recursivamente dentro de todos los subdirectorios del directorio actual.

El patrón de búsqueda normalmente es una palabra o una parte de una palabra. También se pueden utilizar *expresiones regulares*, para realizar búsquedas mas flexibles, por ejemplo, si se quisiera buscar la ocurrencia de todas las palabras que comiencen con «a» minúscula, la ejecución del comando sería algo así:

```
usuario@maquina:~/$ grep a* archivo
```

El tema de manejo de expresiones regulares es bastante largo y complejo, mas adelante se dará con mas detalle.

El comando `find`

Se utiliza este comando para buscar archivos dentro de una jerarquía de directorios. La búsqueda, como veremos mas adelante, se puede realizar mediante varios criterios. La sintaxis de este comando es:

```
find [camino...] [expresión]
```

La `expresión` se conforma de opciones, pruebas y acciones. En este manual no enumeraremos todas las opciones, pruebas y acciones de este comando, sino las expresiones que son mas cotidianas, dejamos al alumno para que investigue todo el potencial de este comando mediante la lectura de la página de manual por medio de la ejecución del siguiente comando:

```
man find
```

Algunos de los criterios de búsqueda que se pueden utilizar son:

```
find CAMINO -name ARCHIVO
find CAMINO -name ARCHIVO -perm MODO
```

`ARCHIVO` corresponde al nombre entero o en parte del archivo que se está buscando, `MODO` son los permisos del archivo a buscar representados en octal, como manejarse con permisos de usuario se verá en la sección 2.3.

El comando `rm`

He aquí un comando peligroso, `rm` se utiliza para borrar archivos o directorios, su sintaxis es:

```
rm [opciones] archivo...
```

Se debe *siempre* pensar dos veces lo que se está haciendo antes de ejecutar este comando. Quizás esto parezca una advertencia para tontos, pero mas aún cuando se está administrando un equipo que da servicios a varios usuarios, un «teclazo» en falso, y fácilmente se pierden datos importantes. Sus opciones mas utilizadas son:

-f No imprimir mensajes de error, ni preguntar al usuario confirmación de cada archivo borrado.

-r Borrar los contenidos de directorios recursivamente.

-v Muestra el nombre de cada archivo eliminado.

el argumento `archivo` puede ser tanto un nombre de archivo, como una expresión regular.

El comando `mkdir`

Este comando es bastante simple, su finalidad es la creación de directorios, y su sintaxis es así:

```
mkdir [opciones] directorio...
```

Sus opciones son las que siguen:

-m modo Establece los permisos de los directorios creados.

-p Crea los directorios padre que falten para cada argumento `directorio`.

El comando `ln`

Este comando sirve para establecer enlaces entre archivos. Un enlace puede ser rígido o simbólico, el primer tipo es simplemente una forma de dar otro nombre a un archivo, por ejemplo teniendo el archivo `/etc/passwd`, se puede hacer un enlace y tener el nuevo nombre en `/home/usuario/claves`, y ambos nombres de archivos refiriéndose al mismo archivo. El segundo tipo es parecido al primero, pero se pueden enlazar directorios, y además de diferentes sistemas de archivos, este tipo de enlace es el que mas se utiliza. La sintaxis del comando `ln` es:

```
ln [opciones] origen [destino]
ln [opciones] origen... directorio
```

Sus opciones mas importantes son las siguientes:

-d Permite al *super-usuario* hacer enlaces rígidos a directorios.

-s Crear enlace simbólico.

-f Borrar los archivos de destino que ya existen.

Para el caso del ejemplo anterior, se debería ejecutar:

```
ln -s /etc/passwd /home/usuario/claves
```

Cuando se ejecuta `ls -l` en un directorio donde hay un enlace simbólico, éste se nota de la siguiente manera:

```
usuario@maquina:~/$ ls -l claves
lrwxrwxrwx    1 usuario usuario 11 Apr  8 13:33 claves -
> /etc/passwd
```

La «l» al comienzo de la línea especifica el tipo de archivo listado, en este caso, un *link*.

El comando `pwd`

Este es un comando muy simple y a la vez útil. Su función es la de imprimir en pantalla el directorio donde el usuario está trabajando.

El comando `df`

Provee información sobre la utilización del espacio en disco en los diferentes sistemas de archivos montados en el sistema. Para un sistema GNU/Linux, quedarse sin espacio libre es algo bastante grave, ya que muchos *demonios* y programas en general utilizan el directorio `/tmp` para guardar información mientras se ejecutan. La sintaxis de `df` es la siguiente:

```
df [opciones] [sistema-de-archivo...]
```

Si no se provee del argumento `sistema-de-archivo`, `df` informará acerca de todos los sistemas de archivos montados y en funcionamiento. Las opciones de `df` mas relevantes son:

-h Imprimir los tamaños de forma mas legible para humanos.

-i Informar sobre la utilización de los nodos-í[4]. Los nodos-í son estructuras internas del sistema de archivos, cuando éste se queda sin nodos-í libres, por mas que haya espacio libre en disco, no se podrán crear nuevos archivos hasta que se liberen nodos-í, generalmente esto no pasa a menos que se generen una enorme cantidad de archivos muy pequeños.

-k Mostrar los tamaños en bloques de 1024 bytes.

-m Mostrar los tamaños en bloques de mega-bytes.

Un ejemplo de ejecución del `df` es:

```
usuario@maquina:~/$ df
Filesystem           1k-blocks      Used Available Use% Moun-
ted on
/dev/hda2              2949060   2102856    696400  75% /
/dev/hda1                23302      2593     19506  12% /boot
/dev/hda4             10144728   5506796   4637932  54% /home
/dev/hdb2              3678764   3175268    503496  86% /u
```

El comando `man`

Quizás uno de los comandos mas importantes para cualquier aprendiz (y a veces no tan aprendiz), el comando `man` sirve para desplegar en pantalla las *páginas de manual*, que proporcionan ayuda en línea acerca de cualquier comando, función de programación, archivo de configuración, etc.

Hay diferentes tipos de páginas de manual, cada tipo se diferencia por un número, que en la siguiente se detallan:

[4]Abreviación de *nodos índice*, en inglés i-nodes

1. Programas ejecutables y guiones[5] del intérprete de comandos.
2. Llamadas del sistema (funciones servidas por el núcleo).
3. Llamadas de la biblioteca (funciones contenidas en las bibliotecas del sistema).
4. Archivos especiales (se encuentran generalmente en `/dev`).
5. Formato de archivos y convenios, por ejemplo `/etc/passwd`.
6. Juegos.
7. Paquetes de macros y convenios, por ejemplo `man(7)`, `groff(7)`
8. Comandos de administración del sistema (generalmente solo son para root).
9. Rutinas del núcleo.

El comando `passwd`

`passwd` se utiliza para cambiar la contraseña de usuario, su sintaxis es:

```
passwd [nombre-usuario]
```

Si se especifica `nombre-usuario`, se cambiará la contraseña de dicho usuario, si no, la del usuario que ejecuta el comando. La mecánica de cambio de contraseña tiene 3 pasos:

1. Ingresar la contraseña antigua.
2. Ingresar la contraseña nueva.
3. Repetir la contraseña nueva para confirmar.

El comando `whoami`

Este es otro comando muy simple como `pwd`. Su función consiste en presentar en pantalla el nombre de usuario del usuario que lo ejecuta. Ejemplo:

```
usuario@maquina:~/$ whoami
usuario
```

[5] En inglés, scripts, son programas creados en el lenguaje del intérprete de comandos

El comando `whereis`

Este comando se utiliza para localizar el archivo binario, el código fuente y la página de manual de un determinado comando. Su sintaxis es como sigue:

```
whereis [opciones] archivo...
```

La lista de opciones mas utilizadas es:

-b Buscar solamente el archivo binario.

-m Buscar solamente la página manual.

-s Buscar solamente el código fuente.

Como ejemplos, se ve lo siguiente:

```
usuario@maquina:~/$ whereis -m whereis
whereis: /usr/share/man/man1/whereis.1.gz

usuario@maquina:~/$ whereis man
passwd: /usr/bin/passwd /etc/passwd /usr/share/man/man5/passwd.5.gz
```

En el primer ejemplo se ha pedido la página de manual del mismo comando `whereis`[6], mientras que en el segundo se han pedido todos los archivos que tengan que ver con el comando `passwd`.

El comando `locate`

`locate` es un comando de búsqueda de archivos, bastante parecido al comando anteriormente visto, el `find`. La diferencia de `locate` es que la búsqueda la hace en una base de datos indexada para aumentar significativamente la velocidad de respuesta. Esto quiere decir, que `locate` realmente no busca en el disco del sistema, sino que en un archivo con la lista de todos los archivos que existen en el GNU/Linux. Generalmente todas las distribuciones de GNU/Linux ejecutan a una hora determinada (generalmente cerca de las 4:00am, ya que tarda algún tiempo realizar esta tarea) un comando para actualizar la base de datos que utiliza `locate`, dicho comando se llama `updatedb`. Su sintaxis es:

```
locate PATRÓN
```

Donde `PATRÓN` corresponde al mismo tipo de patrón que en el comando `find`. Ejemplo de ejecución:

[6] El lector ya está advertido que al autor le gustan las «recursividades», por favor perdonenlo.

```
usuario@maquina:~/$ locate locate
/usr/bin/locate
/usr/lib/locate
/usr/lib/locate/bigram
/usr/lib/locate/code
/usr/lib/locate/frcode
/usr/share/doc/kde/HTML/en/kcontrol/kcmlocate.docbook.gz
/usr/share/doc/xlibs-dev/XdbeAllocateBackBufferName.3.html
/usr/share/doc/xlibs-dev/XdbeDeallocateBackBufferName.3.html
/usr/share/doc/xlibs-dev/XtAllocateGC.3.html
/usr/share/emacs/20.7/lisp/locate.elc
/usr/share/gnome/help/gsearchtool/C/locate.png
/usr/share/man/man1/locate.1.gz
/usr/share/man/man5/locatedb.5.gz
/usr/X11R6/man/man3/XdbeAllocateBackBufferName.3x.gz
/usr/X11R6/man/man3/XdbeDeallocateBackBufferName.3x.gz
/usr/X11R6/man/man3/XtAllocateGC.3x.gz
/var/lib/locate
/var/lib/locate/locatedb
/var/lib/locate/locatedb.n
```

Como se puede observar en el ejemplo, `locate` ha listado todos aquellos archivos que posean la palabra «locate» en su nombre (los directorios están incluidos).

El comando `cal`

Este es un comando bastante útil, que aunque no tenga mucha relación con los anteriormente dados, sirve para demostrar que las herramientas basadas en texto no son inútiles para tareas domésticas. `cal` es una herramienta que sirve para mostrar en pantalla el calendario. Su sintaxis es la siguiente:

```
cal [-jy] [[mes] año]
```

Si `cal` se ejecuta sin argumentos, mostrará en pantalla el calendario del mes y año actuales, por ejemplo:

```
usuario@maquina:~/$ cal
     April 2001
 S  M Tu  W Th  F  S
 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
```

Las opciones de este comando son:

-j Muestra la fecha en formato Juliano.

-y Muestra el calendario completo del año actual.

El comando `wc`

El nombre del comando `wc` proviene de *word count*, y como es de suponer, sirve para contar palabras. Pero no sólo palabras como veremos a continuación. Su sintaxis es como sigue:

```
wc [opción...] [archivo...]
```

Si se omite el argumento `archivo`, `wc` tomará los datos (naturalmente) de la entrada estándar.

La lista de opciones mas importantes es la siguiente:

-c Contar bytes.

-l Contar líneas.

-w Contar palabras.

Como ejemplo, se pueden contar las líneas del archivo `/etc/passwd` y de esta manera se sabrá rápidamente cuantos usuarios tiene definidos el sistema:

```
usuario@maquina:~/$ wc -l /etc/passwd
     32 /etc/passwd
```

Se pueden combinar varios argumentos a la vez.

El comando `date`

Este comando tiene dos funciones: una es la de mostrar en pantalla la fecha del sistema (en varios formatos, como veremos a continuación), la otra es la función de configurar la hora del sistema, pero para que esta funcionalidad se cumpla, se debe ejecutar el comando desde una sesión de *root*. La sintaxis de este comando es:

```
date [opción...] [+FORMAT]
date [opción] [MMDDhhmm[[CC]AA][.ss]]
```

`FORMAT` controla el formato con que se mostrará la fecha, alguna de las opciones de este argumento son:

%a Día de la semana abreviado.

%A Día de la semana completo.

%b Nombre del mes abreviado.

%B Nombre del mes completo.

%d Día del mes.

%m Número de mes.

%H Hora, en formato 24h.

%M Minuto.

%S Segundos.

Existen muchísimas mas opciones de formato que alentamos al lector a verlas en la página de manual de este comando `date`.

Ejemplo de ejecución:

```
usuario@maquina:~/$ date
Sun Apr  8 15:09:32 ART 2001
usuario@maquina:~/$ date +"%A %d %B"
Sunday 08 April
```

2.3 Permisos de usuario

2.3.1 Introducción

¿Para qué sirven los permisos de usuario? Bueno, es una pregunta bastante obvia teniendo en cuenta que GNU/Linux es un sistema operativo multiusuario. Cuando muchas personas utilizan un mismo equipo, debe haber un mecanismo que sirva para diferenciar los archivos de un usuario de los demás archivos.

Un concepto no del todo correcto es pensar que los usuarios se utilizan exclusivamente por personas. Los procesos[7] que se ejecutan en un sistema GNU/Linux

[7]Recordemos que un proceso es un programa en ejecución

tienen también un usuario «dueño», que coincide generalmente con el usuario que ejecutó dicho programa. Además, los *demonios*[8] tienen su propio usuario por cuestiones de seguridad.

En esta sección se verá el tema de los permisos de usuario desde el punto de vista de un usuario común, y no de un administrador. Mas adelante se verán los detalles mas complejos que generalmente incumben al administrador.

2.3.2 Conceptos básicos

Los permisos de un archivo cualquiera (inclusive los directorios) se agrupan en 3 grupos de 3 bits cada uno, como se muestra mas abajo:

```
rwx   rwx   rwx
 |     |     |
 |     |    otros
 |    grupo
usuario
```

Como se ha dicho, cada grupo posee 3 bits:

Bit r: Lectura

Bit w: Escritura

Bit x: Ejecución

Con las diferentes combinaciones, se puede configurar un archivo para que pueda ser leído y modificado por su dueño, y sólo leído por el grupo y los demás, por ejemplo el archivo `/etc/passwd`:

```
-rw-r--r--    1 root    root    1509 Apr  4 12:44 /etc/passwd
```

Este archivo es del usuario **root**, y del grupo del mismo nombre, solamente se puede modificar (bit «w» de escritura) por su usuario dueño, y leer por el grupo y los demás.

Los grupos son un tema mas que nada administrativo, no lo tocaremos en esta sección, sólo hay que tener en cuenta que generalmente en un sistema GNU/Linux, un usuario cualquiera pertenece a su grupo (grupo del mismo nombre que su nombre de usuario) y al grupo *users*.

A diferencia de sistemas operativos como *DOS* y *Windows*, el hecho de que un archivo tenga una extensión `.com` o `.exe` no significa que será un programa

[8]Procesos que dan servicios, como por ejemplo el servidor de páginas web

ejecutable. Al necesitar restringir los derechos de ejecución de cualquier archivo[9], la acción de ejecutar cualquier programa estará supeditada al permiso correspondiente (bit «x» de ejecución). Esto es importante de tener en cuenta a la hora de escribir programas que serán interpretados, ya que al final de cuentas los archivos serán de texto, y para que se ejecuten se le deberá activar el permiso de ejecución.

2.3.3 Un caso especial, los directorios

Quizás a mas de un lector le ha asaltado la siguiente duda: ¿para qué servirá el bit de ejecución en los directorios?. Obviamente, los directorios no se ejecutan, y evidentemente, el bit «x» en los directorios existe, como se ha aclarado anteriormente, en estos casos, dicho bit tiene un significado especial.

El bit de ejecución en los directorios permite el poder ver la información acerca de los archivos que contienen.

El bit de lectura permite listar los contenidos de un directorio.

El bit de escritura permite crear y borrar archivos dentro de un directorio.

Generalmente es conveniente manejar los permisos de lectura y ejecución de los directorios en forma conjunta, para evitar confusiones.

2.3.4 Cambio de permisos: el comando `chmod`

Para cambiar los permisos de los archivos se usa el comando `chmod`. Su sintaxis es la siguiente:

```
chmod [-R] modo archivo...
```

La opción `-R` permite cambiar recursivamente los permisos de todos los archivos dentro de un directorio.

El argumento `modo` está compuesto por alguna combinación de las letras *u* (usuario dueño), *g* (grupo dueño), y *o* (otros), seguido de un símbolo + o - dependiendo si se quiere activar o desactivar un permiso, siguiendo por último una combinación de las letras correspondientes a los distintos permisos: *r*, *w* y *x*. Así, si se necesita dar permisos de ejecución al usuario y al grupo de un archivo, el comando deberá ejecutarse de la siguiente manera:

```
chmod ug+x nombre-de-archivo
```

O si se necesita sacar el permiso de lectura y ejecución de todos los archivos y subdirectorios del directorio `/home/usuario/prueba` para el *grupo* y los *otros*, se debe ejecutar:

```
chmod -R go-rx /home/usuario/prueba
```

[9]Siempre teniendo en cuenta a los archivos ejecutables, es decir programas.

2.3.5 Representación octal

Existe una manera mas ágil de representar los permisos de archivo. Teniendo en cuenta que cada grupo de 3 bits es un número binario, la representación en octal consiste en traducir cada grupo a un número octal, de tal manera que quede como resultado un número de 3 dígitos, cada dígito representando a un grupo de 3 bits.

Mejor aclarar esto con un ejemplo:

```
rwx rw- r--  representación escrita
111 110 100  representación binaria
 7   6   4   representación octal
 |   |   |
 |   |  otros
 |  grupo
usuario
```

La tabla 2.1 da una guía de la traducción de números binarios a octales.

Binario	*Octal*
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Tabla 2.1: Traducción de binario a octal

Entonces se puede concluir que los siguientes comandos son equivalentes:

```
chmod u+rwx go-rwx nombre-de-archivo.txt

chmod 700 nombre-de-archivo.txt
```

2.4 Variables de entorno

Las *variables de entorno* tienen la funcionalidad de configurar ciertos aspectos del entorno del intérprete de comandos y otros programas, que pueden cambiar con el tiempo. Estas variables se establecen cuando se abre una sesión, y la mayoría son configuradas por los *scripts* de inicio del intérprete de comandos[10].

Aunque se pueden establecer nombres de variables con minúsculas, por costumbre se utilizan nombres en mayúsculas, el comando para establecer las variables de entorno se llama `export`[11], y se utiliza de la siguiente forma:

```
export VARIABLE=valor
```

Para ver el contenido de una variable, se puede usar el comando `echo` de la siguiente manera:

```
echo $VARIABLE
```

Para eliminar una variable, se utiliza el comando interno del intérprete `bash`, llamado `unset` pasándole como parámetro el nombre de la variable.

Es importante notar que una vez que se sale de una sesión, las variables establecidas se pierden. Es por eso que si se necesita disponer de variables específicas cada vez que se abra una sesión en GNU/Linux, es imprescindible agregar dichas configuraciones a los archivos de inicio del intérprete de comandos.

Otro uso común de estas variables es en los *scripts*, programas hechos en el lenguaje del intérprete; las variables de entorno son de gran ayuda para establecer configuraciones fácilmente cambiables en dichos programas.

2.5 Redirección de entradas y salidas

2.5.1 Entrada, salida y error estándar

Haremos una breve introducción a los conceptos que definen los componentes de un programa en línea de comandos.

Una traducción no oficial de

```
$ info "Text utilities" "Opening the software toolbox" "Tool-
box introduction"
```

«Mucha gente lleva una navaja de la Armada Suiza en los bolsillos de sus pantalones (o cartera). Una navaja de la Armada Suiza es una herramienta útil de tener: tiene varias hojas de cuchillo, un destornillador, pinzas, palillo para dientes,

[10]En el caso del intérprete `bash`, estos scripts incluyen el `.bashrc`, `.bash_profile`, y otros.

[11]En `bash`.

sacacorchos y probablemente unas cuantas cosas más. Para trabajos chicos misceláneos de todos los días donde se necesita una herramienta de propósito general, es la herramienta indicada.

En la otra mano, un carpintero experimentado no construye una casa utilizando una navaja de la Armada Suiza. En reemplazo, posee una caja de herramientas llena de herramientas especializadas–una sierra, un martillo, un destornillador, etc. Y conoce exactamente dónde y cuando utilizar cada herramienta; no lo vas a agarrar martillando clavos con el mango del destornillador.»

La filosofía de Unix (en línea de comandos) cree que un único programa *especializado* para hacer todas las tareas no es bueno, al menos no para usuarios avanzados o administradores.

Los usuarios finales sin conocimientos en áreas informáticas pueden preferir tener un sólo programa para todo, pero este único programa tiene dificultades para el mantenimiento y modificación. Se vuelve monstruoso y complicado.

En reemplazo se prefiere la *navaja de la Armada Suiza* para tareas cotidianas, o sea un conjunto de programas chicos de gran simplicidad que en conjunto se potencian.

Para que los programas trabajen en conjunto se utiliza el concepto de ***flujo*** como una corriente de bytes.

Al igual que con las tuberías reales (digamos caños, los grifos o canillas, duchas, etc.) de una casa, se conectan unos a otros donde cada uno tiene una *entrada*, una *función* y una *salida*. El concepto de «tuberías» lo veremos en la sección 2.5.2 por ahora vamos a distinguir los otros tres aspectos con un ejemplo.

El comando `sort` puede ordenar por orden alfabético. Esta sería la *función* del programa. Pero £qué ordena? bueno aca es donde interviene la *entrada*. Que sin utilizar la magia de las tuberías, será el **teclado** la entrada.

Haremos una prueba:

```
$ sort
El
comando
sort
puede
ordenar
por
orden
alfabético
```

... luego presionamos Ctrl-D que significa "fin de archivo" en la mayoría de los casos, apareciendo en pantalla:

```
alfabético
comando
El
orden
ordenar
por
puede
sort
```

Con este ejemplo nos damos cuenta que la *salida* es la **pantalla**

Para esta altura varios se pueden preguntar “Pero no es la única entrada que posee mi programa”; es verdad que un programa puede tener muchas entradas y muchas salidas. Por ejemplo cuando lee un archivo, éste es una entrada más. Pero los conceptos que venimos estudiando son entradas y salidas especiales, llamadas *entrada estándar* y *salida estándar*.

También existe un tipo de salida adicional que es el *error estándar*. Por este flujo se canaliza todos los mensajes de error o avisos del programa. Facilitando varias tareas. Una muy común es reunir todos los errores en un archivo separado en caso de problemas, para su posterior análisis.

En nuestro ejemplo si `sort` intentaba abrir un archivo y no existía, es preferible que escriba:

```
Fichero o directorio no existe
```

Antes que lo canalice como salida estándar y diga

```
directorio
existe
Fichero
no
o
```

Teniendo en claro los conceptos vamos a ver como utilizamos las tuberías para «pegar» los programas entre si.

2.5.2 Tuberías (pipes)

Podríamos graficar cada programa como una «caja negra» que tiene una entrada y una salida que se pueden unir entre ellos.

El ejemplo que utilizamos se encuentra esquematizado en la figura 2.1 siendo la entrada estándar el teclado y la salida estándar la terminal o por simplicidad la pantalla.

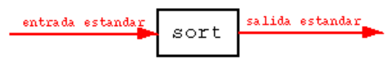


Figura 2.1: Esquema de entrada y salida estándar del ejemplo

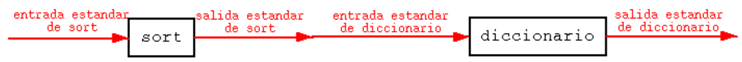


Figura 2.2: Esquema de entrada y salida estándar del ejemplo 2

Vamos a suponer un caso ficticio donde necesitamos todas las definiciones de cada palabra en un texto. Primero las ordenamos alfabéticamente, luego utilizamos un comando ficticio llamado `diccionario` que toma palabras de la entrada estándar y las reescribe junto a su significado en la salida estándar.

Su esquema se ve en la figura 2.2. En este caso nombramos por separado las entradas y salidas estándares de los dos programas, pero la «unión» entre ambos programas se puede considerar como un solo «tubo».

En ese tubo el flujo está en un estado intermedio, donde está ordenado pero no tiene las definiciones de diccionario.

En la linea de comandos esto se escribe de la siguiente manera:

```
$ sort | dicccionario
```

Donde el caracter | representa el la conexión entre la salida estándar de un programa y la entrada estándar de otro.

Con este fuerte y simple concepto se pueden concatenar gran cantidad de programas como si fuera una línea de producción en serie para generar resultados complejos.

Para mejorar nuestro ejemplo sacaremos las palabras repetidas, antes de mostrarlas con definiciones. Suponiendo que exista un programa llamado `sacar-repetidas`, la linea de comando sería:

```
$ sort | sacar-repetidas | diccionario
```

Simple, utilizando herramientas sencillas logramos algo un poco más complicado. El inconveniente que tenemos en este ejemplo es que hay que tipear aquello a procesar. Normalmente queremos utilizar archivos como entrada de nuestros datos. Es necesario un comando que envíe a salida estándar un archivo, así se procesa como la entrada estándar del `sort` y continua el proceso normalmente. Este comando es `cat`. La sintaxis es simple `cat` *`nombre-de-archivo`*.

Quedando nuestro ejemplo:

```
$ cat archivo.txt | sort | sacar-repetidas | diccionario
```

... arma un glosario de las palabras que se encuentren en `archivo.txt`

La combinación de comandos es incalculable y brinda posibilidades enormes veremos algunos ejemplos en la ejercitación.

2.5.3 Redirección

Si bien en nuestro ejemplo ilustrativo es bueno ver los resultados en pantalla; repetidas veces en la vida de un sistema es mejor tener todo en archivos, ya sea para guardar historial de algo o para automatizar ciertas funciones dentro de scripts.

Para almacenar o sacar información de archivos y vincularlas con entradas o salidas estándares se utilizan *Redirecciones*.

La redirección se expresa con los símbolos «Mayor» (>) y «Menor» (<). Se pueden utilizar en forma simple (>) o en forma doble (»).

Escritura

Para escribir un archivo se utiliza (>). Hay que tener mucho cuidado de no borrar un archivo sobreescribiéndolo. Cuando se utilizan redirecciones, debido a su utilidad en los scripts, **no se realizan confirmaciones**. Si el archivo a escribir existe y posee información valiosa, aplicar `> archivo-importante` lo sobreescribe con el contenido del flujo.

En cambio el operador (») realiza un *agregado*[12] de los datos en el flujo.

No hay nada mejor que un ejemplo esclarificador:

```
$ escribe-en-salida-estandar > archivo.txt
```

El (falso) comando `escribe-en-salida-estándar` justamente hace eso, escribe unas cuantas cosas en salida estándar. Puede ser un `ls`, un `cal` o cualquier comando antes visto, como también una combinación de comandos por tuberías.

En este punto el contenido de `archivo.txt` es lo mismo que saldría en pantalla. Si ejecutamos otro comando redireccionado a `archivo.txt`, este pierde su contenido y el resultante de la operación pasa a estar en el archivo.

Cuando se necesita tener una lista de acontecimientos, no se quiere que un acontecimiento nuevo borre a todos los anteriores. Para lograr esto *agregamos* en vez de sobreescribir.

```
$ echo Este es el acontecimiento Nro. 1 > bitacora.log
$ echo Este es el segundo acontecimiento >> bitacora.log
```

[12] *append* en inglés

Va a escribir dos líneas en el archivo `bitacora.log` sin eliminar nada.

Ejemplo Si queremos combinar el ejemplo de las tuberías en la sección 2.5.2 con lo aprendido recientemente podríamos tipear:

```
$ cat archivo.txt | sort | sacar-repetidas | diccionario >> glosario.txt
```

Lectura

Para la lectura es el símbolo inverso (<) y se utiliza de la siguiente manera:

```
$ comando-que-acepta-stdin < archivo-de-entrada.txt
```

Debido a que preferimos el comando `cat` para estas operaciones por una mayor legibilidad.

Práctica - 2.5

1- Con los comandos aprendidos, mostrar de un directorio de varios archivos los primeros 10.

2- Modificar el ejercicio anterior para mostrar los 10 archivos de mayor tamaño, ordenados alfabéticamente.

3- En un directorio con varios archivos, mostrar solo los que tienen una determinada terminación como por ejemplo `.txt` utilizando `grep` y `find`.

4- Utilizando `find` o una composición de varios comandos por tuberías, mostrar sólo los *enlaces simbólicos* existentes, en caso de no poseer, crear varios en varios subdirectorios con el comando `ln`

2.6 Archivos de configuración del intérprete de comandos `bash`

2.6.1 Introduccion

La configuración de un intérprete de comandos consiste mayormente en establecer las variables de entorno, los *aliases* de comandos y el formato del *prompt* que se necesiten. Dependiendo del tipo de uso que se le dará al intérprete de comandos, habrá diferentes necesidades de configuración de estos valores.

Existen dos modalidades de uso del *intérprete de comandos* bash:

interactiva es la comun

no interactiva es propicia para scripts

Por esto,

2.6.2 `.bashrc`

2.6.3 `.profile`

2.6.4 `.bash_profile`

======= El intérprete de comandos tiene unos cuantos archivos que se ejecutan al inicio.

Los archivos pueden categorizarse en:

no■login cada vez que se ejecuta el bash (incluso desde el interprete de comandos), se leen estos archivos. Es el caso más común, cada vez que se abre una `konsole`, `xterm`, `gnome-terminal` o equivalente, se ejecutan.

login sólo cuando el usuario comienza la sesión se ejecuta. En tiempos de terminales y consolas era fácil identificar cuando el usuario se *logueaba*. Hoy en día es muy popular el login gráfico, el cual no carga inmediatamente un `bash` como antes.

2.6.5 .bashrc

Este archivo es del tipo no-login. Primero se carga el archivo global al sistema `/etc/bashrc` y luego sea pasa a al archivo `.bashrc` en el directorio personal del usuario.

Es un archivo que probablemente llame a otros como por ejemplo `.profile` y establezca las variable se entorno.

2.6.6 .bash_profile

Este archivo es del tipo login, por lo que se ejecuta una sola vez en una sesión.

Todos los archivos son scripts por lo que en la sección 2.7 se ven ejemplos de programación en `bash`. Permitiendo personalizar el `bash` enormemente u obligando a los usuario a ejecutar ciertas tareas administrativas en cuanto ingresen al sistema o bien ejecutan el interprete de comandos.

2.7 Programación en shell

Uno de las grandes ventajas que ofrece un interprete de comandos es la programación en un lenguaje rústico pero poderoso para automatizar infinidad de tareas.

2.7.1 Introducción

Como todo lenguaje, posee reglas sintácticas que establecen la forma a escribir las sentencias a ejecutar.

Para quienes poseen conocimiento de otros lenguajes de programación, el signo «punto y coma» (;) es utilizado frecuentemente como separador o terminador de sentencias. En `bash` no es necesario y puede ser reemplazado por `Enter`. Es común encontrar una línea de este tipo:

```
# comando1 ; comando2
(ejecución de comando1 seguido de comando2)
```

es equivalente a:

```
# comando1
(ejecución de comando1)
# comando2
(ejecución de comando2)
```

En el primer ejemplo con una sola línea se ejecutan ambos comandos. Es muy buen ejemplo cuando se quiere encadenar tareas que consumen mucho tiempo y tienen que ser seguidas.

Hay que tener presente que no se ejecutan en paralelo. Recién cuando termina de ejecutarse `comando1` empieza a ejecutarse `comando2`.

2.7.2 Códigos de salida

Cada programa una vez que termina puede brindar al entorno un *Código de salida* para que otros programas o el intérprete sepan como concluyó la aplicación.

Tomemos un ejemplo de la vida de un administrador. Es común que la administración sea remota, por lo que vamos a considerar que en nuestra tarea no tenemos conocimiento de lo que esta pasando en una maquina alejada en la que se esta ejecutando `arreglar-base-de-datos`.

El script `arreglar-base-de-datos` es un script que corrige posibles errores en una hipotética gran base de datos. Y el resultado de esa corrección interesa, especialmente, si no se pudo arreglar.

Vamos a suponer que hay 2 posibles alternativas:

Salida exitosa La base de datos no tuvo ningún error. En este caso sólo hay que agregar al archivo `/var/log/BD.registro` una línea con la fecha de chequeo y el responsable en ese momento.

Se detectaron errores pero no se repararon Esta situación es peor. Hay que escribir información detallada en `/var/log/BD.registro` y además enviar e-mail a una lista de encargados y directivos de la empresa.

Para diferenciar cada caso se asigna un *código de salida* a cada uno. Luego de ejecutar `arreglar-base-de-datos` se verifica en base al código, los comandos a ejecutar.

El algoritmo sería algo similar a:

```
if arreglar-base-de-datos
then
  date >> /var/log/BD.registro
```

```
  echo $RESPONSABLE_BD >>  /var/log/BD.registro
else
  informar-errores.sh >> /var/log/BD.registro
  enviar-mail "Errores en BD" lista-encargados lista-directivos
fi
```

¿Y dónde están los códigos de salida? Bueno, el *comando interno*[13] `if` analiza el código de salida, y si es `0` (cero) ejecuta la lista de comandos después del `then`, en caso contrario (y si existe) la lista de comandos después del `else` hasta encontrar un `fi`.

Por lo tanto el script `arreglar-base-de-datos` tiene que devolver `0` en caso de éxito. Este es el comportamiento normal de la mayoría de los comandos en Linux y otros Unixes, y un valor para varios errores.

Las paginas `man` suelen tener una sección llamada **Exit Status** que contiene los códigos que devuelve el programa.

2.7.3 El comando `if`

Ya vimos un ejemplo sencillo utilizando `if`, que a su vez puede ser de gran utilidad. Ya hablamos de la equivalencia entre el «;» y el «Enter» pero hay veces que desapercibido el detalle de que `if` y `then` están en diferentes lineas por lo que:

```
# if COMANDO then COMANDO fi
```

Este último ejemplo va a dar error de sintaxis. La forma correcta de expresar es:

```
# if COMANDO; then COMANDO ; fi
```

o bien:

```
# if COMANDO
> then COMANDO
> fi
```

Esta programación se puede ir realizando «en línea» o sea, en una consola o terminal tipear cualquiera de estos ejemplo (siempre que existan los comandos).

Muchas veces es necesario hacer comparaciones o comprobaciones para tomar decisiones. Por ejemplo "Si el usuario no posee el archivo `/.configuracion` con la configuración por defecto" o bien "Si el numero de archivos es mayor a 20 escribir `no se puede transferir`"

[13]*built-in command* en inglés.

Existe el comando `test` para hacer estas evaluaciones y en base al resultado, su código de error de `test` será `0` u otro número. Por ejemplo, para saber si un archivo `.configuracion` existe en el *home* del usuario el comando puede ser:

```
# test -e $HOME/.configuracion
```

para facilitar la notación dentro del comando `if` se hace un *enlace simbólico*[14] a un comando llamado `[`. Parece extraño llamar a un comando con un corchete abierto pero veamos un ejemplo:

```
if test -e $HOME/.configuracion
```

Puede traducirse a:

```
if [ -e $HOME/.configuracion ]
```

donde el `]` *(corchete cerrado)* final no tiene importancia y la programación queda menos engorrosa.

Podríamos utilizar lo aprendido para crear un script que "Si el usuario no posee el archivo `/HOME/.configuracion` con la configuracion por defecto" en unas pocas líneas:

```
if [ -e $HOME/.configuracion ]
then
  crear-configuracion >> $HOME/.configuracion
fi
```

El comando test permite la composición de condiciones con AND y OR lógicos con los modificadores `-a` y `-o` respectivamente y el modificador NOT con `!`. Se podría agregar a la linea del `if` anterior la condición "y además no posee el archivo `SinConfiguracion`" de la siguiente forma

```
if [ -e $HOME/.configuracion -a ! -e SinConfiguracion]
```

Ejemplos mucho más interesantes de analizar se pueden encontrar en el directorio `/etc/rc.d/init.d`[15].

[14] *symbolic link* en inglés, utilizando el comando `ln -s`
[15] Este directorio puede variar según las distribuciones, también puede ser `/etc/init.d`

2.7.4 El comando `while`

El comando `test` se utiliza cuando se itera con el comando `while`. En este comando es muy útil la comparación de valores.

`test` puede comparar números al igual que cadenas de caracteres.

```
while [ ${CANT_USUARIOS} -le 1 ]
do
  echo Todavia no hay suficientes jugadores
  sleep 1
done
echo Ahora hay más de 1 usuario
```

Este ejemplo compara la variable CANT_USUARIOS si es menor o igual (`-le` significa *less or equal* en inglés) a uno, de ser así, repite cada 1 segundo, «`Todavía no hay suficientes jugadores`» en cuanto la cantidad de usuarios sea mayor a 1 sale del ciclo.

También es posible hacer un ciclo infinito utilizando `test` (o bien llamado `[`) para que devuelva siempre verdadero (con `[ 1 ]`). Se recomienda usar el comando `true` que devuelve un código de salida exitoso (cero) y el `while` no termina a menos que se le envíe una señal con `Ctrl-C`.

```
while true
do
  clear
  mailq
  sleep 2
done
```

Este simple algoritmo muestra el contenido de la «bandeja de salida» del `sendmail` cada 2 segundos. Vemos que con pocos conocimientos en `bash` se pueden lograr infinidad de cosas.

2.7.5 El comando `for`

Para quienes programan en otros lenguajes el comando `for` se comporta distinto a la clásica sentencia *for*. Este comando asigna *de* una lista de elementos, el valor *a* una variable y repite una lista de comandos con esa variable.

Si bien la explicación pudo ser un poco confusa, el concepto es bastante fácil de entender al ver un ejemplo.

```
for cantidad in dos tres cuatro cinco seis siete
do
  echo ${cantidad} elefantes se balancaban sobre la te-
la de una araña
  echo como veian que resistia fueron a llamar a otro elefante...
done
```

dos (...) siete son los elementos.

cantidad es la variable que iteración a iteración va tomando los valores de la lista de elementos

do; echo (...);done es el bloque de comandos a iterar.

Esta es la forma más simple de utilizar el comando `for`, pero con pocas variaciones se puede realizar cosas muy útiles, por ejemplo:

```
for archivo in `ls`
do
  touch ${archivo}
done
```

La lista de elementos se obtiene de el resultado del comando `ls`. Es decir, primero se ejecuta `ls`, el cual dará el listado de todos los archivos de un directorio, y a todos esos archivos se les aplica un `touch`[16].

[16] El comando `touch` cambia la fecha de modificación de un archivo a la fecha actual

Práctica - 2.7

1- Hacer un script que compile un programa[17] y SOLO en el caso de que la compilación sea exitosa, realice el enlazado (o *linkeado*) del mismo[18].

2- Suponiendo que el comando `cant-mb-libres` retorna la cantidad de MB libres en el disco, hacer un script que compruebe la capacidad disponible, y si es mayor a 640MB, copia el directorio `/usr/local` al directorio `/backup`. En caso contrario mandar un mail a `administrador@lejos.ch`, también utilizando un hipotético comando: `enviar-mail`.

3- En base al listado del directorio `/backup`, enviar a la impresora (o agregar al archivo `/dev/lp0` que es equivalente) los primeros 30 elementos. Si existía mayor cantidad de archivos escribir una línea final que diga "y mas...". Este ejercicio realizarlo con `while` o `for` en vez de tuberías.

[17]Se puede realizar con `gcc -c main.c`
[18]Aquí podemos realizar `gcc main.o -o main`

Capítulo 3

Correo Electrónico

3.1 Introducción

El correo electrónico es, y ha sido siempre, el servicio de mayor uso en Internet. Para muchas personas, el correo electrónico forma una parte significativa de su vida, ya que es un medio de comunicación confiable, rápido y barato.

Es por esto que creemos conveniente citar algunas de las *reglas de etiqueta de la red*[1], las cuales nos ayudarán a tener una correcta actitud frente a este servicio, y frente a los demás usuarios del mismo.

Algunas de las reglas mas importantes, las listamos a continuación:

- El enviar correo con propaganda no solicitado, o también conocido como correo SPAM, está muy mal visto y a muchas personas le molesta. En la actualidad las listas de miles de cuentas de correos se utilizan como moneda de cambio con las cuales se negocian para que los *spammers*[2] puedan enviar propaganda en forma masiva.

- No se debe asumir que por ser personal, los correos electrónicos no podrán ser leídos por otra persona. Antes de llegar a destino, cada correo electrónico pasa por al menos dos servidores en Internet, y el único que conocemos es el servidor de nuestro proveedor, los demás no los conocemos, entonces se debe tener extremo cuidado con lo que se envía por este medio. Afortunadamente exiten métodos para asegurar que los mensajes enviados sólo los entienda el receptor del mismo; esto se hace mediante la encriptación y el uso de programas como el **GnuPG** ó **GPG**[3], esto lo veremos mas adelante en la sección 3.3.

[1] En inglés: **netiquette**, su página web es la siguiente `http://www.fau.edu/rinaldi/netiquette.html`

[2] Aquellas personas que generan correo SPAM

[3] Las siglas significan *GNU Privacy Guard*

- Cuando se cita a la otra persona, debe evitarse citar mas de lo mínimamente necesario para dejar claro a que parte del correo se está respondiendo. Las citas no deben utilizarse para otra cosa que dar contexto a las respuestas que uno envía.
- Utilizar la línea de «subject» en los correos. El resumir en pocas palabras lo que se refiere todo el mensaje es muy conveniente, ya que muchas personas reciben gran cantidad de correos electrónicos diarios, y dan prioridad de lectura dependiendo del «subject».
- En el caso de uso de firmas en los correos enviados, no debería usarse una firma que tenga mas de 4 o 5 líneas, las firmas gigantescas muchas veces son mal recibidas, ya que generan tráfico innecesario en la red y además ofenden visualmente.
- El escribir todo en mayúsculas es considerado como hablar a los gritos y debería ser evitado en lo posible.
- Tener cuidado con los clientes de correo que se utilizan, con respecto al formato de salida de los mensajes. Muchos clientes de correo modernos utilizan el formato **HTML** como método de dar mejor imágen a los correos, pero existen otros tantos clientes de correo que no soportan este formato, y por consiguiente, los usuarios de estos programas deben realizar un esfuerzo tal para poder leer aquellos mensajes, que muchas veces los eliminan sin leerlos por el sólo hecho de ser prácticamente ilegibles.

Esta lista puede seguir, pero se han citado algunos puntos, los más importantes que se deben tener en cuenta a la hora de interactuar con un par vía el correo electrónico.

En este capítulo, se verá el uso de dos clientes de correo, el `pine` y el `mutt`, ambos son muy utilizados. Además se verá, como se ha dicho anteriormente, como encriptar la información de manera que de ser interceptado un correo, no se lo pueda descifrar a menos que sea el receptor del mismo. Otro tema importante que se dará, es la manera de enviar correos automatizados, muchas veces útil para dar aviso de algún evento en el sistema. También se verá como configurar el `fetchmail` para poder recibir correos vía un servidor remoto, y además el uso del `procmail` para el filtrado y selección de correos.

3.2 Uso de clientes de correo

Se han seleccionado estos dos clientes de correo por ser ambos en caracteres, lo que permite su uso en un equipo con o sin gráficos. Además, estos dos programas

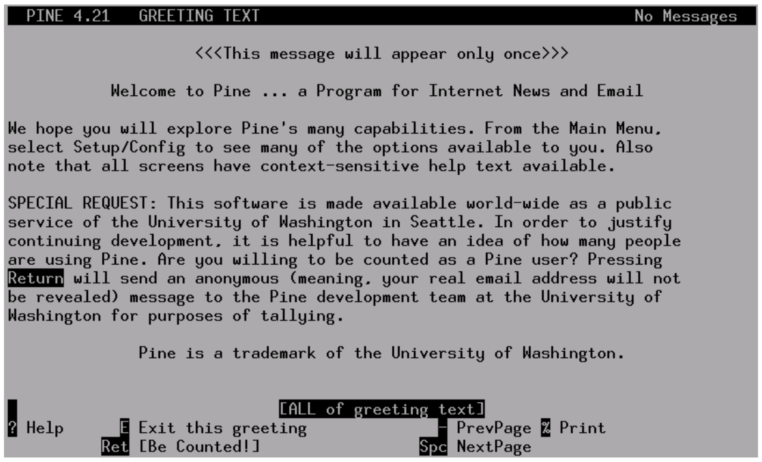


Figura 3.1: Pantalla inicial del `pine`

ya tienen un tiempo de desarrollo respetable, lo que los hace muy confiables y eficientes a la hora de manejar correo en gran cantidad.

Primero se comenzará con el `pine`, este capítulo pretende dar una idea básica del uso de estos programas, existen muchas opciones de configuración que no se mencionarán aquí, pero la base dada en este curso permitirá al alumno seguir investigando. Luego se seguirá con el `mutt`, un programa que en estos últimos años ha cobrado bastantes adeptos por su facilidad de uso.

3.2.1 Pine

La mayoría de las distribuciones (excepto Debian, virtualmente todas) incluyen este programa de correos. Es un cliente de correo y grupos de noticias muy avanzado, que posee una enorme cantidad de opciones.

La primera vez que el usuario ejecute el `pine`, el programa se iniciará con una pantalla como se muestra en la figura 3.1, donde se explica que este programa se mantiene en la Universidad de Washington y el proyecto sigue funcionando en la medida que aquellas personas sepan que el programa se utiliza, es por eso que se le pide al usuario oprimir `Enter` para ser contado entre los usuarios de `pine`. Si uno no quiere ser contado, puede salir de esa pantalla pulsando la tecla `E`, esa pantalla no se volverá a mostrar otra vez.

El `pine` dividide su área de trabajo en dos: la parte inferior, que funciona a modo de barra de menú, y la parte superior toma varios formatos dependiendo de la sección donde se encuentre el usuario. En la barra de menú, se irán mostrando las diferentes funciones y sus teclas asociadas, como generalmente existen en cada sección mas opciones que lugar físico donde distribuirlas, pulsando la tecla

O se mostrarán mas comandos. Aquella función que aparezca encerrada entre corchetes, es la que se ejecutará por defecto al pulsar Enter.

Una vez que se ha salido de la pantalla de bienvenida, aparecerá la pantalla principal, o *Main Menu*, se puede acceder a la pantalla principal desde las demás secciones del programa pulsando M. Como vemos en la figura 3.2, la pantalla principal se divide en varias secciones:

HELP Es el sistema de ayuda del `pine`, posee toda la documentación en línea explicando cada detalle del programa, es realmente recomendable utilizar esta ayuda para descubrir todas las posibilidades que provee este cliente de correo

COMPOSE MESSAGE Mediante el uso de esta opción desde la pantalla principal o pulsando C desde otras secciones, el programa activa su modo de edición para enviar un mensaje nuevo. Esta funcionalidad la veremos mas adelante.

MESSAGE INDEX Si seleccionamos esta opción, iremos al índice de mensajes de la carpeta que tengamos seleccionada. Apenas arranca el programa, la carpeta por defecto es `INBOX`, pero se pueden agregar carpetas adicionales para organizar los mensajes por temática. Esta opción puede accederse en otras secciones del `pine` pulsando la tecla I. El manejo de carpetas también se verá con en detalle mas adelante.

FOLDER LIST Esta opción lleva al usuario al índice, pero de las carpetas que existan en el programa, la tecla L activa esta opción.

ADDRESS BOOK Para evitar tener que recordar grandes cantidades de cuentas de correo, o tener que escribirlas en papeles que siempre luego se pierden, el programa posee esta funcionalidad a modo de libreta de direcciones. En ella se pueden configurar además listas de distribución, ya se verá mas adelante como funciona.

SETUP Seleccionando esta opción se ingresa a un submenú con varias posibilidades de configuración y personalización del `pine`. Mas adelante se verán algunas de las opciones de configuración mas importantes.

QUIT Esta opción es mas que obvia, se utiliza para salir del programa, en otras secciones del mismo, pulsando Q se activa.

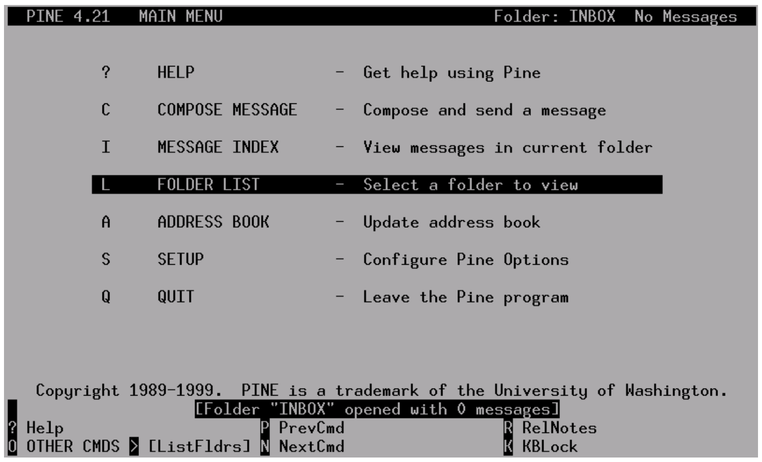


Figura 3.2: Pantalla principal del `pine`

Configuración básica

Al seleccionar `Setup`, o al pulsar la tecla `S`, el programa da al usuario una lista de ítems que pueden ser configurados, los mas importantes, y los que se verán en este curso son dos: `Config` y `Signature`, para el resto se invita al alumno a leer la documentación del programa.

Primero lo mas simple, al seleccionar la opción `Signature`, `pine` provee al usuario de un editor (el mismo editor que se utilizará para componer mensajes) para poder crear una firma que luego se agregará a cada mensaje que se escriba. Por favor recordar lo que se dijo en la sección 3.1, no es conveniente una firma muy grande, manteniendola de 4 o 5 líneas no habrá problemas. Como se indica en el menú de esta sección, para salir, se debe pulsar `Ctrl-X`; el programa preguntará si se quiere guardar los cambios.

La sección `Config` (que se accede mendiante la tecla `C`), presenta al usuario una enorme lista de opciones para configurar cada aspecto del comportamiento del programa, la mayoría de las opciones por defecto que trae configuradas el `pine` están correctas, pero se nombrarán aquellas opciones que conviene cambiar.

personal-name Por defecto el programa utiliza el nombre que el usuario tiene configurado en el sistema, en el caso de que el nombre del usuario no esté correcto, se puede editar esta línea y corregir el error. En aquellos campos donde el `pine` no tenga ningún valor, se verá una etiqueta diciendo `No Value Set`, cuando el programa asigna un valor por defecto a un campo no configurado, además agregará el valor por defecto.

user-domain El `pine` por defecto trata de averiguar qué dominio tiene configurado el equipo donde está ejecutandose, para utilizarlo en la dirección de origen en los mensajes que el usuario escriba. En algunos casos[4], el dominio de la dirección de correo del usuario no coincidirá con el dominio del equipo local, entonces es conveniente configurar este valor para que la dirección origen en los mensajes que se escriban, salgan correctamente.

smtp-server Si se tiene un servidor de correo configurado en el equipo local, no se debería ingresar ningún valor en este campo, ya que el programa por defecto utilizará el servidor local como servidor de correo saliente. Puede darse el caso que en el equipo local no se encuentre un servidor de correo configurado, entonces hay dos opciones: si el equipo es propio, es muy conveniente instalar un servidor de correo local, pero si el equipo lo administra otra persona, habrá que consultar con ésta para averiguar el valor a ingresar en este campo.

character-set Con esta opción se configura el conjunto de caracteres que utilizará el programa al mostrar y escribir mensajes. Por defecto, el `pine` utiliza el valor `US-ASCII`, que corresponde al conjunto de caracteres que se utilizan en los Estados Unidos de Norteamérica. Al tener un lenguaje mas rico, debemos avisarle al programa que se necesitará soporte para nuestro conjunto de caractares, para lograr esto, se debe ingresar `iso-8859-1` en ese campo y con esto se tendrá soporte para los acentos, las eñes, y demás caracteres extendidos.

Hay muchas otras opciones que se pueden cambiar, pero como se ha dicho antes, no son de tanta importancia como las nombradas anteriormente. El usuario puede siempre pulsar la tecla `?` para pedir ayuda acerca de cualquier opción de configuración.

Para salir de la sección de configuración, como dice en el menú, se debe pulsar `E`, el programa preguntará si se quieren guardar los cambios.

Carpetas

El uso de carpetas se hace evidentemente necesario cuando uno comienza a recibir grandes cantidades de correo. Hemos hablado ya del *correo SPAM*, y por mas que tengamos cuidado en no dejar en muchos lugares nuestra dirección electrónica, tarde o temprano caerá en manos de un empresario inescrupuloso que la utilizará para enviar su molesta publicidad. Es una actitud un tanto pesimista, pero es lo que la experiencia a través de los años ha demostrado en mas de una ocasión.

[4]Normalmente, en aquellos casos en que el sistema no está conectado a Internet mediante un enlace dedicado

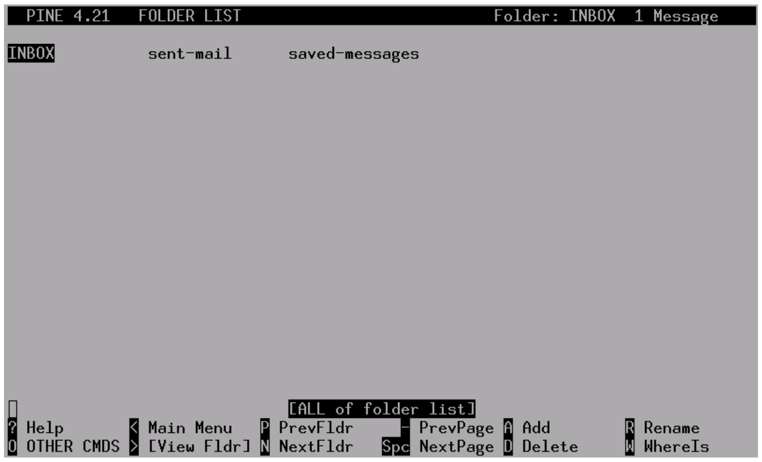


Figura 3.3: Lista de carpetas del `pine`

Tarde o temprano recibiremos cientos de mensajes diarios, y gran parte de ellos serán de SPAM, ¿cómo evitar desesperarse con tanta cantidad de mensajes, si los interesantes sólo son algunos pocos?. La respuesta está en el filtrado de mensajes, y la división de los mismos en carpetas; el filtrado es algo que se explicará mas adelante, por ahora nos abocaremos a la tarea de crear carpetas.

En la figura 3.3 se puede ver la lista de carpetas por defecto que existe cuando el `pine` es recién iniciado. Estas carpetas en realidad son archivos que se encuentran almacenados en el directorio personal de cada usuario, dentro del subdirectorio `mail/`[5].

De las funciones del menú, se tienen 3 que sirven para administrar las carpetas: `Add`, `Delete` y `Rename`. Está de mas explicar para que sirven.

Con las flechas del teclado se puede ir seleccionando una u otra carpeta, y pulsando `Enter` entraremos al índice de mensajes de la carpeta seleccionada.

Vale la pena nombrar la funcionalidad de las tres carpetas por defecto que el `pine` crea al inicio:

INBOX En esta carpeta se irán almacenando los mensajes que lleguen, el alumno verá mas adelante que mediante el uso de filtrado, se podrá seleccionar la carpeta donde se guardan mensajes espcíficos.

sent-mail Esta carpeta almacena los mensajes enviados por el usuario. Es útil a la hora de necesitar reenviar algún mensaje que por una u otra razón no ha llegado a destino.

[5] A excepción de la carpeta `INBOX`, que es una referencia al buzón de entrada del servidor de correos, generalmente este archivo tiene el mismo nombre de cada usuario, y se encuentra en el directorio `/var/spool/mail/`

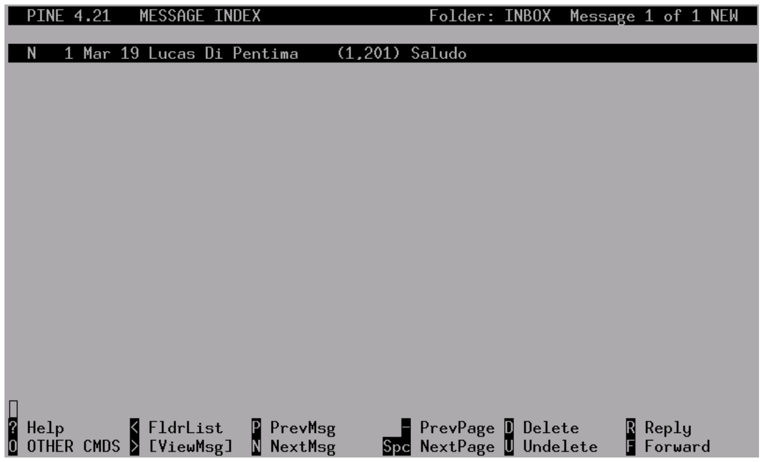


Figura 3.4: Índice de mensajes del `pine`

saved-messages Como ya se verá, los mensajes pueden ser movidos de carpeta en carpeta, la carpeta por defecto a donde son movidos los mensajes es ésta. Generalmente la carpeta `INBOX` contiene aquellos mensajes nuevos que van llegando, el usuario entonces los va leyendo, respondiendo y si cree conveniente, los guarda para un futuro, de lo contrario, los elimina; la carpeta `saved-messages` es para esto.

Leyendo y enviando correo

Entrando a una carpeta, estaremos en el índice de mensajes de la misma, viéndose algo parecido a la figura 3.4. Los mensajes que se reciban nuevos tendrán una «N» a la izquierda.

Para ver el mensaje seleccionado, naturalmente se pulsa `Enter`, como en la figura 3.5. Para responder el mail seleccionado, se utiliza la tecla `R`, el programa preguntará si se quiere citar el mensaje anterior, esto es bueno hacerlo, siempre y cuando las citas se mantengan en un límite adecuado y no se abuse de las mismas.

Si se quiere escribir un mensaje desde cero, la tecla `C` en las diferentes secciones del programa tiene la misma funcionalidad, *componer mensajes*. Al activar esta función, el programa cargará el editor de mensajes, con el encabezado del mensaje a enviar arriba, y el lugar para el cuerpo por debajo, como se puede observar en la figura 3.6.

Se supone que el alumno sabe como enviar correo electrónico, este curso no se trata de ello, así que no se explicarán las funciones de cada campo del encabezado del mensaje a componer, lo que si vale la pena aclarar, es el uso de la libreta de direcciones (addressbook) y los archivos incluídos (file attach) dentro de los

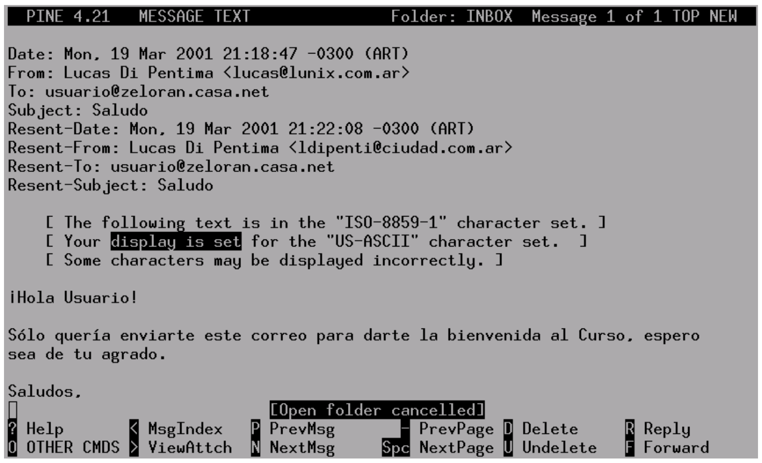


Figura 3.5: Leyendo mensajes en `pine`

Figura 3.6: Pantalla de composición de mensajes del `pine`

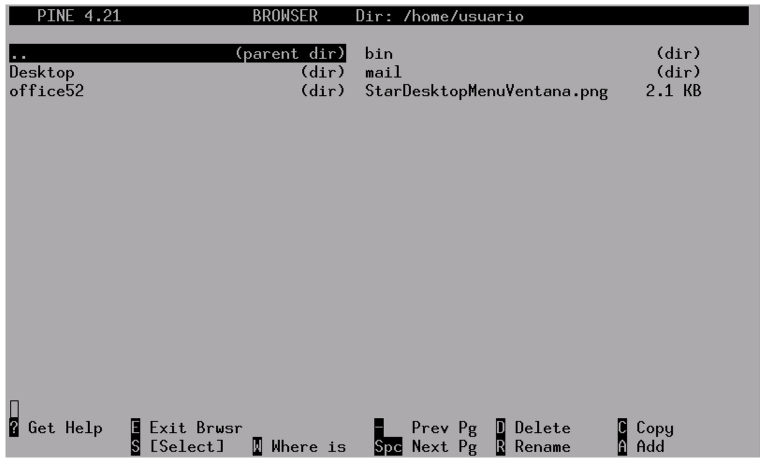


Figura 3.7: Seleccionando un archivo a incluir en el mensaje

mensajes. Cuando el cursos se encuentra en el campo `To:`, pulsando `Ctrl-T` se puede seleccionar la dirección de destino que tengamos almacenada en la libreta de direcciones.

Cuando se necesite enviar uno o varios archivos por correo electrónico, se debe posicionar el cursor en el campo `Attachmnt:` y pulsando `Ctrl-T`, se carga un navegador de disco como el que se ve en la figura 3.7.

Una característica del `pine` que no se ha visto, es la del *addressbook*, con esta libreta de direcciones se puede mantener toda la lista de contactos de forma fácil. Estando en la pantalla principal, se puede acceder pulsando la tecla `A`, y de esta forma se obtiene la lista de contactos ingresados. Primeramente se tendrá la lista vacía, cuando se necesite agregar un nuevo contacto, se utiliza (como se puede ver en el menú) la tecla `@`, apareciendo de esta manera una pantalla como la que se ve en la figura 3.8.

En el campo `Nickname:` se debe ingresar el alias de la persona que luego se utilizará en el campo `To:` cuando se compone un mensaje. De esta manera, no hará falta escribir todo el nombre de la persona.

Haciendo respaldo de las configuraciones

Como el alumno sabrá, el hacer copias de respaldo es una buena tarea para evitar disgustos. Entonces es bueno enumerar aquellos archivos que el `pine` utiliza para guardar sus configuraciones, estos archivos son manipulados por el `pine`, y aunque son archivos de texto, no se recomienda editarlos directamente.

Las configuraciones del programa se guardan en un archivo localizado en el directorio personal de cada usuario, con el nombre de `.pinerc`.

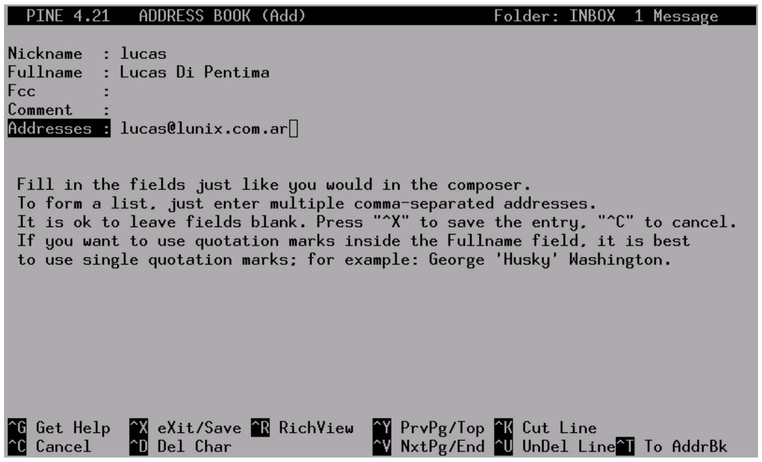


Figura 3.8: Agregando un contacto en la libreta de direcciones del `pine`

La libreta de direcciones con todos los contactos que se tengan, se almacenan en un archivo llamado `.addressbook`.

Finalmente, la firma que se haya personalizado, se almacena en un archivo con nombre `.signature`

También es bueno recordar que las carpetas con los mensajes almacenados (exceptuando `INBOX`) se guardan en el directorio personal, en un subdirectorio llamado `mail/`.

A la hora de hacer respaldo de la correspondencia electrónica, es bueno conocer estos datos. Otro punto interesante a nombrar es que el `pine`, cada fin de mes preguntará al usuario si quiere mover las carpetas `saved-messages` y `sent-mail` a otro lugar, para ir almacenando solamente los mensajes del mes actual, es una buena práctica ir guardando los mensajes viejos en otro sitio, así el `pine` no tarda tanto tiempo al intentar abrir una carpeta que tiene miles de mensajes acumulados a través de los meses o años.

3.2.2 Mutt

3.3 Protegiendo nuestra privacidad: uso del GnuPG

3.3.1 La seguridad del correo electrónico

Generalmente se cree que el correo electrónico es totalmente privado, que nadie, excepto el receptor del mensaje, puede leer el mensaje que se envía. Esto dista de ser verdad, de hecho es mucho mas fácil espiar el correo electrónico que el correo postal, dado que al fin de cuentas, el correo electrónico termina siendo un archivo en algún o algunos equipos en la red, a medida que viaja a su destino final.

Con lo dicho anteriormente, no se debe pensar que cada mensaje que se envía es leído por alguna persona distinta a la que en realidad se le envió el mensaje, pero es muy recomendable tener siempre en mente que un mensaje que viaja por la red puede ser revisado [6]. Tarde o temprano, se necesitará enviar información sensible por correo electrónico, entonces habrá que utilizar algún método para *cifrar* la información de manera tal que sólo el receptor del mensaje pueda descifrarla correctamente.

Esta sección se encarga de introducir al alumno al tema del cifrado de datos utilizando una herramienta libre: el *GnuPG (GNU Privacy Guard)*.

3.3.2 ¿Qué es el GnuPG?

El *GNU Privacy Guard* (comúnmente conocido como GPG) es una herramienta de cifrado de información, que utiliza un sistema de *cifrado asimétrico*, ¿que quiere decir esto?, que cada usuario tiene un par de claves:

Clave pública La cual se puede (y debe) hacer conocer a todas aquellas personas que quieran enviarnos información cifrada.

Clave privada Esta clave se debe guardar y en ninguna circunstancia entregar a nadie, se la utiliza para descifrar los mensajes que nos envían cifrados.

Este método se utiliza porque es mucho mas seguro que un sistema de clave única, por ejemplo: Si la persona A y la persona B quieren comenzar a mandarse correo cifrado, deberían establecer una clave con la cual cifrar (y descifrar) la información (mensajes de correo en este caso) que intercambian. La persona A establece una clave y se la envía por correo a B, pero por correo no cifrado (porque aún B al no tener la clave, no podría descifrar el mensaje de A). B al recibirla comienza a cifrar los mensajes que envía a A, y A usa la misma clave para descifrarlos y cifrar los suyos hacia B. Esto parece seguro, pero de hecho no lo es, ya que si alguna otra persona pudo interceptar el mensaje de A hacia B (no cifrado) con la clave, luego usando dicha clave podría descifrar todos los mensajes que A y B intercambien sin ningún problema.

El sistema de cifrado asimétrico soluciona este problema, ya que A y B entonces tendrían dos claves cada uno, entonces A le envía su clave pública a B y B envía su clave pública a A. Luego A utiliza dicha clave pública de B para cifrar los mensajes dirigidos a B, lo mismo hace B con la clave pública de A. Aquellos

[6] Esto obviamente es ilegal, es una violación a la privacidad y muchos países ya tienen leyes sobre este tema.

mensajes cifrados con la clave pública de B sólo podrán ser descifrados con la clave *privada* de B al igual que los mensajes cifrados con la clave pública de A, podrán ser descifrados con la clave privada de A. Como dichas claves privadas están en poder de sus dueños y nadie mas, cualquier persona que intercepte los mensajes no cifrados con las claves públicas de A y B, no podrá utilizarlas para descifrar ningún mensaje, ya que el único uso que tienen las claves públicas es cifrar (y no descifrar) información que podrá ser descifrada solo por el dueño de dicha clave pública.

3.3.3 Configurando el GnuPG

Como primera medida para utilizar el GPG, se debe configurar las claves pública y privada. El comando correspondiente al GPG se llama `gpg`, y para generar las claves se debe ejecutar como sigue a continuación:

```
usuario@maquina:~$ gpg --gen-key
gpg (GnuPG) 1.0.4; Copyright (C) 2000 Free Software Foun-
dation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistri-
bute it
under certain conditions. See the file COPYING for details.

gpg: /home/usuario/.gnupg/secring.gpg: keyring created
gpg: /home/usuario/.gnupg/pubring.gpg: keyring created
Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (4) ElGamal (sign and encrypt)
Your selection?
```

Se le solicita al usuario que seleccione el tipo de clave que se utilizará. Como el mismo GPG lo dice, algunas claves sirven solo para firmar, otras para firmar y cifrar, se recomienda utilizar la opción por defecto.

A continuación el GPG pregunta sobre la longitud de las claves a generar. Cuanto mas larga sean las claves, mas difícil será para cualquier persona intentar romper el cifrado por *fuerza bruta*[7], el usuario entonces elegirá la longitud (`gpg` recomienda una longitud no mayor a 2048 bits) dependiendo de su paranoia. Se

[7]Se llama fuerza bruta al método de probar todas las combinaciones de algo, por ejemplo una clave numérica, para adivinarla. Imagine que una clave numérica de 3 dígitos es mucho mas fácil de adivinar probando todas las 1000 posibilidades, que una clave de 6 dígitos.

debe tener en cuenta que cuanto mas larga es la clave, también mas trabajo le toma a la máquina cifrar y descifrar la información.

```
DSA keypair will have 1024 bits.
About to generate a new ELG-E keypair.
              minimum keysize is  768 bits
              default keysize is 1024 bits
    highest suggested keysize is 2048 bits
What keysize do you want? (1024)
```

Una vez elegida la longitud de las claves, el programa pide la fecha de caducidad de las mismas.

```
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
```

Generalmente no se necesitará tener una fecha de caducidad, lo recomendable es entonces elegir la primer opción. El problema de tener una clave con fecha de caducidad es que una vez que el par de claves caduca, se debe enviar el par nuevo a todas aquellas personas que usaban el viejo par, y esto puede ser bastante engorroso.

Luego habrá que ingresar los datos personales, los cuales se almacenarán con el par de claves para distinguirse de las demás claves que luego se posean. GPG necesita el nombre real del usuario, la dirección de correo electrónico y un comentario (como por ejemplo, el teléfono o nombre de una empresa).

```
You need a User-ID to identify your key; the software cons-
tructs the user id
from Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Usuario
Email address: usuario@maquina.dominio.com
Comment: 233-8847
You selected this USER-ID:
    "Usuario (233-8847) <usuario@maquina.dominio.com>"
```

```
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit?
```

Una vez que se aceptan los datos ingresados, GPG solicita un nivel mas de protección a la clave privada del usuario: una frase de contraseña. Esto es muy útil por si alguna vez la clave privada es de alguna manera robada del dueño, se necesitará saber esta frase para poder utilizar la clave privada. Se recomienda que la frase contenga letras mayúsculas, minúsculas y si es posible, números, de manera que sea muy difícil adivinarla por *fuerza bruta*[8].

Luego de ingresar la frase de contraseña dos veces, GPG necesita obtener información al azar para la generación de las claves, y para esto, toma la información de la actividad del sistema operativo, es por eso que pide al usuario que realice cualquier actividad, como mover el mouse, usar el disco rígido (por ejemplo cargando algún programa), hasta obtener la cantidad necesaria de datos.

Con esto se concluye la etapa de configuración de GPG. El programa habrá generado el directorio `.gnupg` en el directorio personal del usuario. Los datos que contengan ese directorio no deben ser publicados en ningún sitio, ya que ahí dentro se encuentra la clave privada.

3.3.4 Uso diario del GnuPG

Una vez configurado el programa de cifrado, lo primero que se debe hacer es *generar un certificado de revocación*, el cual servirá para poder anular el par de claves en el caso de olvido de contraseña, o robo de la clave privada. Esto se hace de la siguiente forma:

```
usuario@maquina:~$ gpg --output cert-revocacion.asc --gen-
revoke Usuario

sec  1024D/62B70584 2001-04-22   Usuario (233-8847) <usuario@maquina.dominio.com>

Create a revocation certificate for this key? y
Please select the reason for the revocation:
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  0 = Cancel
(Probably you want to select 1 here)
Your decision?
```

[8] Se debe recordar exactamente cual es la frase de contraseña, respetando las minúsculas de las mayúsculas, porque se necesitará ingresarla cada vez que se quiera cifrar o descifrar algún mensaje.

El argumento que se le debe pasar al parámetro `-gen-revoke` es el nombre que previamente se ingresó cuando se configuró GPG. El programa pide la razón por la cual se genera el certificado de revocación y un comentario, luego GPG genera el archivo `cert-revocacion.asc` (que es un archivo de texto) con el certificado incluido. Este archivo debería guardarse en algún medio de almacenamiento y esconderse donde nadie tenga acceso, una de las opciones válidas puede ser imprimir el archivo y guardar la hoja en algún lugar seguro, luego borrar el archivo.

Por cada persona a la que se quiera enviar mensajes cifrados, se debe tener almacenada su clave pública en el *anillo de llaves* de GPG. Para ver la lista de claves públicas que se tiene en cualquier instante, se ejecutará el siguiente comando:

```
usuario@maquina:~$ gpg --list-keys
/home/usuario/.gnupg/pubring.gpg
--------------------------------
pub  1024D/62B70584 2001-04-22 Usuario (233-8847) <usuario@maquina.dominio.com>
sub  2048g/7459EB6A 2001-04-22
```

Inicialmente se tendrá únicamente la propia clave pública en el anillo. Cuando se reciba alguna clave pública de otra persona, se la debe agregar al anillo de la siguiente manera:

```
usuario@maquina:~$ gpg --import <nombre_archivo>
```

Donde `<nombre_archivo>` corresponde al archivo donde se tengan la o las claves públicas a ingresar.

De manera similar, se necesitará exportar la propia clave pública a un archivo de texto para poder enviarla a aquellas personas que quieran enviar mensajes cifrados. Esto se hace como sigue:

```
usuario@maquina:~$ gpg --armor --output <archivo_exportado> -
-export <nombre>
```

Donde `<archivo_exportado>` será el archivo donde se almacenará la clave pública en formato de texto, y `<nombre>` deberá corresponder con el nombre o la dirección de correo electrónico propia que se haya ingresado en los datos del programa. De hecho, este comando sirve para exportar cualquier clave pública, la propia o la de cualquiera que esté en el anillo, proveyendo el correspondiente nombre o dirección electrónica.

La clave pública exportada tendrá aproximadamente el siguiente formato[9]:

[9] De hecho esta clave que se incluye es la clave pública de unos de los autores de este manual, Lucas Di Pentima.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.4 (GNU/Linux)
Comment: For info see http://www.gnupg.org

mQGiBDq6OwIRBADXaEdL7bTUR8HtgfNYz+Bzm5oEGM/vm3tUB1yDgMGZAkugCZt4
b41HX2LXCqmfMLQclS+0B6gYSfYGNG1v/VdGFdXEkrWzVV3QcTEFVDC79qac06eD
zerdIgjgVaPKjwIY8b9i1I2OisjY4Ey5gw2w1CoepCj9DMZZZ7d0tLJhhwCg5OB4
UXQtitHd8L6ASkc9Eyjg6x0EAKg61fRranXcCWhaVDoPvKvjThCTf53wVaw9eSVy
CJwy4f18cMRC+MYT5J9wni4dC2I1YkLGuNrUgb0SVVS2TZkcaI/4LZvIzxSupMww
yMedstQfYe1kzjY/ODnE3OYXyW6k5eEfBopNGO/J300/YDZ/OCXj/zq/TOb9Ztls
SEGSA/9XeoEUqe6lolXMJMAK1eO0TkgFm1B5R+mjnkXrh8z/Ofgw+HuD0Pr9imQi
YMk9ymT+swhC08hv5lDPe0iOHc5biyvKh6t654vIiJF8sds8hkiwW5RYmiVeB3hG
R4dBfVuOEGQXYNabwGSiqECJpXeCl2q7NVvijIUJFqsLTPcBd7Q7THVjYXMgRGkg
UGVudGltYSAoVGVsOiA1NCAzNDIgNDU5MzEyMikgPGx1Y2FzQGx1bml4LmNvbS5h
cj6IVwQTEQIAFwUCOro7AgULBwoDBAMVAwIDFgIBAheAAAoJEI+YP89qpU/Jo18A
oJQ0OyVfioYgCxxbU6f1buN6uKsfAJ4nVyK+YW+RnofcbHJga/SsxAJ2drkBDQQ6
ujsGEAQA8g+dqkNhhywhkj3d8dCqVicq8JjEFQyWIQkP1/Qg0CshljSWjeX1D/bn
fEOkxqt/oy/+ClqMMKABO764NEcu3B7zz16OK/uLuvddY3vfZiA82XJdxu5wXqk4
dKA+iorx5xtE2eNxVAFydXnR7KkiN4HOoVDNxTNGKA22wra3ND8AAwUD/iSd3NoP
zPdhF7/laBefp7vdDo7LRn3iLe7m1NbvXvtYMtNvtWP9LIjq7q1iHqsZw+5Xymkl
LbhaMhfUIoZhqaLr1L1IajuLZA8wUmYeHK/ZswLoEK0bJBYfVxS8gbpJG89PiQXK
PFPwWSHmesR13+nKUtoAsEsOaxRM/f4PPPlziEYEGBECAAYFAjq6OwYACgkQj5g/
z2qlT8ka+ACg24eZNxUhYJ+FF6P7Cd3CPZ/dlhsAnj8NrXcAEN6BbOdTWaS6FDmt
a3FW
=SP6W
-----END PGP PUBLIC KEY BLOCK-----
```

Cifrando información

Suponiendo que una persona quiere enviarle correo cifrado a uno de los autores de este curso, Lucas Di Pentima, una vez que la clave pública de este personaje esté en el anillo de claves públicas, se verá algo parecido a lo siguiente:

```
usuario@zeloran:~$ gpg --list-keys
/home/usuario/.gnupg/pubring.gpg
--------------------------------
pub  1024D/62B70584 2001-04-22 Usuario (233-8847) <usuario@maquina.dominio.com>
sub  2048g/7459EB6A 2001-04-22

pub  1024D/6AA54FC9 2001-03-22 Lucas Di Pentima (Tel: 54 342 4593122) <lucas@lunix.com.ar>
sub  1024g/9252D0E4 2001-03-22
```

Entonces para cifrar el archivo /home/usuario/mensaje.txt se deberá ejecutar el siguiente comando:

```
usuario@maquina:~$ gpg --output mensaje.gpg --encrypt -
-recipient lucas@lunix.com.ar mensaje.txt
```

Una clave pública puede llegar a tener la certificación de otras personas de que es válida, no ahondaremos en este tema, pero es oportuno hacer notar que si una clave pública que se utiliza para cifrar un mensaje, no está certificada por otros, significa que no es seguro que sea realmente de la persona que dice ser. Si esta clave pública la recibimos de fuentes confiables (por ejemplo, directamente del dueño), se la puede utilizar de todas formas, aunque GPG avise que no es totalmente confiable.

El archivo con el mensaje cifrado será, obviamente, el archivo mensaje.gpg.

Descifrando información

Cuando recibamos algún mensaje o archivo cifrado con nuestra clave pública, se debe ejecutar el siguiente comando para descifrarlo:

```
usuario@maquina:~/gpg --output archivo.txt --decrypt archivo.gpg
```

Donde archivo.gpg es el archivo cifrado y archivo.txt es el descifrado resultante.

Conclusión

Este curso aborda el tema del cifrado como una herramienta de comunicación segura con otras personas a través del uso de un servicio no seguro, como lo es el correo electrónico. Esto no quiere decir que GPG sólo sirva para cifrar y descifrar mensajes de texto, al contrario, GPG sirve para cifrar cualquier tipo de archivo, no necesariamente que se deba enviar a otra persona, por ejemplo si el alumno tiene un conjunto de documentos que quiere almacenar en un CD-ROM como copia de seguridad, pero que no quiere que ninguna persona que tenga acceso a ese CD-ROM pueda usar esos documentos, puede cifrarlos con su propia clave pública y almacenar en CD-ROM los archivos cifrados resultantes, teniendo la seguridad que nadie podrá descifrarlos al no tener la clave privada correspondiente.

A continuación se describirán los pasos necesarios para integrar GPG con el cliente de correo pine para poder usar el cifrado en los mensajes de correo electrónico de manera fácil y cómoda.

3.4 Integración de GPG con `pine`

En la sección 3.3 se ha explicado a grandes rasgos los conceptos de la herramienta de cifrado GPG y su uso genérico con archivos de cualquier tipo. Quizás el método descrito anteriormente no es el más cómodo para usarlo con un cliente de correo, ya que al necesitar cifrar un mensaje se debería hacerlo aparte en un archivo de texto, para luego importarlo al cliente de correo; cuando se necesitan enviar varios mensajes cifrados, se convierte en una tarea muy tediosa.

Pero el `pine` tiene la posibilidad de configurarse para tratar con mensajes cifrados o firmados[10], a continuación se describirán los pasos para configurarlo correctamente:

1. Suponiendo que el archivo ejecutable `gpg` esté en `/usr/bin/gpg`, se debe ejecutar lo siguiente:

   ```
   usuario@maquina:~$ ln -s /usr/bin/gpg ~/.gnupg/encrypt
   usuario@maquina:~$ ln -s /usr/bin/gpg ~/.gnupg/gpgsign
   ```

   Lo cual genera enlaces simbólicos del archivo ejecutable al directorio de instalación de GPG en el directorio personal. Esto es para que `pine` pueda distinguir el mismo ejecutable de las dos funciones: firmar y cifrar.

2. Ejecutar el `pine` ingresando en la sección de configuración del programa. Buscar la opción de configuración denominada `display-filters` e ingresar lo siguiente en ese campo:

   ```
   _LEADING("-----BEGIN PGP ")_ /usr/bin/gpg
   ```

   Esto le indica al `pine` que ejecute GPG en caso de detectar la presencia de información cifrada o firmada en el cuerpo de un mensaje.

3. En el campo siguiente, el denominado `sending-filters`, se deben ingresar dos valores, que se incluyen a continuación:

   - `~/.gnupg/encrypt -eastr _RECIPIENTS_`
   - `~/.gnupg/gpgsign -ast`

[10] Los mensajes firmados no son mensajes ilegibles para todos excepto el destinatario, sino que cualquiera lo puede leer, y al tener una firma, se puede constatar que dicho mensaje lo ha escrito la persona quien dice ser.

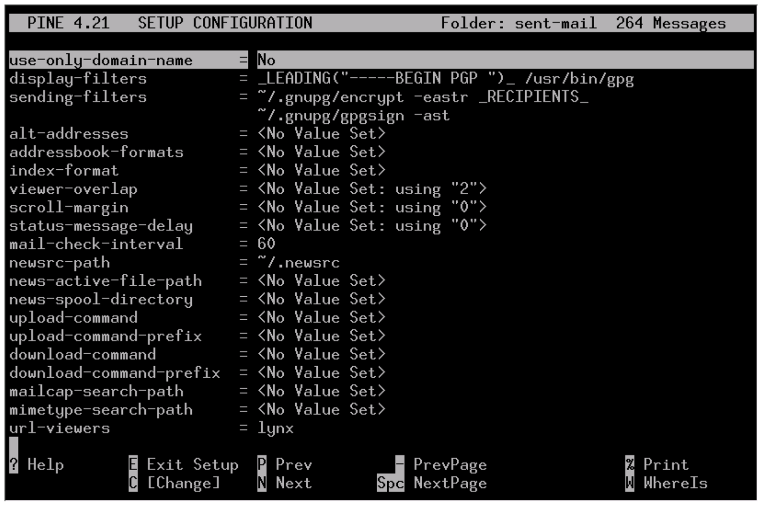


Figura 3.9: Configurando `pine` para uso de GPG

> El primer valor indica a `pine` que cifre (cuando se lo requiera el usuario) con el comando dado y a todos los receptores del mensaje en cuestión. Recordar que se debe poseer la clave pública de aquel al que queramos enviar un mensaje cifrado, de lo contrario el mensaje no se enviará.
>
> El segundo valor indica a `pine` que firme digitalmente el mensaje a enviar. GPG será ejecutado y pedirá la frase de contraseña para realizar la firma, que se agregará al final del mensaje.

La figura 3.9 muestra como debe quedar la configuración de `pine` para usar GPG.

3.5 Envío de correo automatizado con el comando `mail`

3.6 Herramientas de tratamiento de codificación MIME, UUE, ROT13, etc.

3.7 Configuración y uso de `fetchmail`

3.8 Configuración de filtros con `procmail`

Capítulo 4

Edición de archivos

4.1 Diferencias entre vi y emacs

4.2 Configuración y uso básico del editor vi

4.3 Configuración y uso básico del editor emacs

4.4 Configuración y uso del editor pico

4.5 Sistemas de documentación

4.5.1 TeX

4.5.2 LaTeX

4.5.3 groff

4.5.4 Texinfo

4.5.5 SGML

4.5.6 DocBook

Capítulo 5

Redes

5.1 Uso básico de un cliente de FTP

5.1.1 ftp

5.1.2 ncftp

5.2 Uso básico de un cliente de IRC

5.2.1 ircii

5.2.2 BitchX

5.3 Uso básico de un cliente de web

5.3.1 lynx

5.3.2 links

Capítulo 6

Programación

6.1 Conceptos sobre bibliotecas compartidas

6.2 Uso de compiladores

6.2.1 C

6.2.2 C++

6.2.3 Fortran

6.2.4 Java

6.2.5 Perl

6.2.6 Python

6.2.7 Configuración del Apache para uso de PHP

6.3 Automatización de compilaciones con Makefile, autoconf, etc.

6.4 Uso de depuradores

6.4.1 gdb

6.4.2 xgdb

6.4.3 ddd

Capítulo 7

Inicio del sistema GNU/Linux

7.1 LILO

7.2 Carga del kernel

7.3 Scripts de inicio

Capítulo 8

Particiones

8.1 ¿Por qué conviene tener más de una sola partición?

8.2 Algunas propuestas de esquema de particiones

Capítulo 9

Distribuciones

9.1 Distribuciones mejor diseñadas para servidores

9.2 Distribuciones mejor diseñadas para estaciones de trabajo

9.3 Diferencias básicas

Capítulo 10

El comando `su` y el uso de la *fuerza*

Capítulo 11

Tareas administrativas básicas

11.1 Administración de usuarios y grupos

11.2 Personalización y compilación del núcleo

11.2.1 Introducción

¿Que es el núcleo[1]?, ¿que es lo que hace realmente? El núcleo es el corazón del sistema operativo, todo sistema operativo tiene un núcleo, los hay de varias formas: monolíticos, modulares, de tiempo real, *microkernel*, etc. Nuestro Linux (así se llama el núcleo, por eso al sistema completo lo llamamos GNU/Linux) puede hallarse de dos formas, modular o monolítico, pero por ahora no hay que preocuparse por esto ya que lo veremos mas adelante.

Un núcleo (cualquiera) administra los recursos de hardware del equipo, algunos lo hacen de una manera chapucera y desordenada, de modo tal que se producen los famosos *colgazos* y *pantallas azules de la muerte*, otros (como Linux) lo hacen de manera ordenada, y muy eficiente. Un núcleo se encarga de manejar la memoria, los discos, el vídeo, el módem, y demás dispositivos físicos, y da acceso al usuario para que disponga de estos dispositivos de forma controlada.

La administración de la memoria y el procesador que realiza el núcleo Linux permite la ejecución de programas en *pseudo-paralelismo*. No es paralelismo puro a menos que el equipo posea varios procesadores y cada uno esté ejecutando un programa a la vez. El pseudo-paralelismo consiste en dar a cada programa en ejecución[2] una porción de tiempo definida en el procesador, que normalmente son

[1] En inglés se lo llama **kernel**.

[2] A los programas en ejecución se los llama procesos, y así es como los llamaremos de aquí en adelante

unos pocos milisegundos, cuando se acaba el tiempo asignado de procesador[3], el núcleo pausa el proceso para darle prioridad a otro que está esperando su turno. Este intercambio de procesos se hace tan rápidamente que no lo notamos, y es por eso que parece que el equipo ejecuta muchos procesos a la vez.

Como la mayoría de los sistemas operativos tipo UN*X, Linux es un núcleo monolítico[4], esto significa que todos los manejadores de dispositivos deben estar incluidos dentro del código del núcleo al momento de encender el equipo. Esto es a veces poco práctico en el sentido de que al necesitar soporte para algún dispositivo nuevo, se debe *recompilar* el núcleo agregando el nuevo manejador, proceso que toma cierto tiempo y requiere reiniciar el equipo.

Como se dijo anteriormente, el núcleo Linux puede también ser modular, esto significa que muchos manejadores pueden estar en archivos separados del archivo principal del núcleo, y por lo tanto, pueden ser cargados y descargados de memoria dinámicamente, evitando así la necesidad de reiniciar el equipo. Esta es la manera preferible de compilar los núcleos, pero hay ciertas ocasiones cuando se debe si o si compilarlo en forma monolítica.

En esta sección veremos cómo dar soporte a nuevos dispositivos, cómo y donde buscar las actualizaciones de los núcleos nuevos, el manejo de los módulos del núcleo y la configuración del arrancador `LILO`.

11.2.2 Soporte de hardware

En un principio, el hardware que el núcleo Linux soportaba se debía exclusivamente al arduo trabajo de unos cuantos hackers del núcleo, que pasaban incontables horas de trabajo intentando descubrir como una pieza de hardware funcionaba (realizando lo que se llama *ingeniería inversa*), para luego escribir un manejador para que Linux pudiera utilizar ese dispositivo. La mayoría de las empresas fabricantes de dispositivos no entregaba la información necesaria a los programadores del núcleo, y el soporte para los nuevos dispositivos muchas veces se tardaba un tiempo.

Hoy en día, con el aumento de colaboradores en el desarrollo del núcleo, y con la creciente cantidad de empresas que se han dado cuenta que GNU/Linux vale la pena, el soporte para nuevos dispositivos no se hace esperar demasiado, y es por eso que el núcleo Linux soporta:

- Tarjetas de vídeo VGA, SVGA, Monocromo, etc.
- Controladores de discos IDE, EIDE, MFM, RLL, RAID, SCSI.

[3] En la jerga de los sistemas operativos esto se llama *timeslice*.

[4] No es totalmente cierto, como veremos mas adelante

- Controladores de puertos seriales.
- Tarjetas multipuerto.
- Adaptadores de red Ethernet, ISDN, Frame Relay, Inalámbricas, X25, SLIP, PPP, ARCnet, TokenRing, FDDI, AX.25, ATM.
- Tarjetas de sonido.
- Unidades de cinta.
- Unidades de CD-ROM.
- Unidades grabadoras de CD-R.
- Unidades removibles, como por ejemplo Zip, Jaz, Bernoulli y tantas otras.
- Mouse serie, PS/2 y otros.
- Módems normales, y también algunos *winmódems*.
- Impresoras matriciales, de inyección de tinta, y láser.
- Plotters.
- Cámaras digitales.
- Capturadoras de vídeo.
- Unidades DVD-ROM.
- Puertos y dispositivos USB.

La lista es demasiado grande como para detallarla en este curso, pero para tener una idea mucho mas detallada, basta con leer el «Hardware-HOWTO», normalmente localizado en `/usr/doc/HOWTO`.

También debemos tener en cuenta que no todo el soporte del hardware listado es trabajo exclusivo del núcleo Linux. Como ejemplo tenemos el caso de las impresoras, en el núcleo no se necesita definir explícitamente que clase de impresora se tiene conectada al equipo, solamente se necesita activar el soporte para el puerto paralelo (siempre y cuando se utilice una impresora de puerto paralelo obviamente), y el resto del trabajo lo hará un programa a través del núcleo; en el caso del ejemplo, el programa que se encarga de enviar datos a la impresora se llama `lpr`.

11.2.3 Actualización del núcleo

¿Por qué la necesidad de actualizar el núcleo? La primer respuesta que quizás venga a la mente, es por la mayor cantidad de hardware soportado, y seguramente esta respuesta es válida, pero además debemos tener en mente, que con cada nueva versión del núcleo, se solucionan problemas y fallas que versiones anteriores poseían, y muchas veces los núcleos mas avanzados funcionan mas rápido que los de las versiones anteriores. Por esto, es muy recomendable actualizar el núcleo con cada versión *estable* que salga disponible.

Y ya que nombramos las versiones estables, es hora de explicar como se conforma el número de versión de un núcleo, ya que es de suma importancia a la hora de elegir el núcleo a actualizar. Cada versión del núcleo está formado por 3 números, por ejemplo la última versión del núcleo estable hoy en día[5] es la versión 2.4.2.

El primer número (el que se encuentra mas a la izquierda) simboliza el «número mayor de versión», éste cambia cada varios años, cada cambio de este número simboliza un cambio muy importante en la estructura interna del núcleo.

El segundo número simboliza el tipo de núcleo: un número par indica una versión estable, un número impar una versión inestable o de desarrollo. Normalmente las distribuciones de GNU/Linux instalan por defecto una versión estable del núcleo, y es lo recomendable para equipos que funcionarán como servidores o en cualquier ambiente de trabajo. Los núcleos de desarrollo generalmente poseen mas amplio soporte de hardware y funcionalidades nuevas, ya que en el futuro se convertirán en núcleos estables con estas nuevas características, pero la estabilidad no está garantizada en estos núcleos y no es recomendable utilizarlos en máquinas que posean datos importantes o que realicen tareas que requieran de un funcionamiento constante. Mucha gente utiliza los núcleos de desarrollo, normalmente se utilizan en alguna máquina personal, cuando se necesita soporte para un dispositivo nuevo, o cuando se está probando el núcleo para reportar cualquier tipo de error a los programadores, pero en otros casos, no es recomendable usarlos.

El tercer número simboliza el número de revisión del núcleo. Cada revisión nueva sale cada pocos meses, a medida que se van corrigiendo errores y mejorando la eficiencia.

La tarea de actualización del núcleo comprende varias etapas:

1. Conseguir la última versión estable del núcleo. El sitio «oficial» del núcleo Linux es `http://www.kernel.org`, en este sitio se podrán conseguir tanto las versiones estables como inestables del núcleo, así como también una lista de sitios alternativos (servidores espejo) de donde descargar estos

[5] 27 de marzo de 2001

archivos en caso de que el sitio principal esté muy cargado. El tamaño aproximado del paquete del núcleo 2.4.2 es de 20 Mb.

2. Desempaquetar el núcleo. Generalmente el código fuente del núcleo se debe colocar en el directorio `/usr/src/linux`, pero se debe tener cuidado antes de desempaquetar el nuevo núcleo que no esté el código fuente de un núcleo anterior instalado por la distribución. En caso de existir, es conveniente mover el código del núcleo anterior a otro directorio antes.

3. Seleccionar las opciones del núcleo y compilarlo. Esto dependiendo de la máquina donde se haga puede demorar un tiempo. Aunque la potencia de los equipos que hoy se consiguen es suficiente, en los días de las PC-AT 386 y 486, uno aprendía a desarrollar la paciencia, ya que una compilación normal podía tomar entre una hora hasta días si el equipo contaba con poca memoria RAM[6].

4. Instalar el núcleo y sus módulos. Esto implica copiar el archivo del núcleo y los módulos a donde corresponde, y configurar el `LILO`, el programa que cargará el núcleo nuevo cuando se reinicie el equipo.

5. Reiniciar el equipo y probar el nuevo núcleo. Esta última etapa puede también incluir opcionalmente alguna plegaria a su santo preferido.

11.2.4 Compilación del núcleo

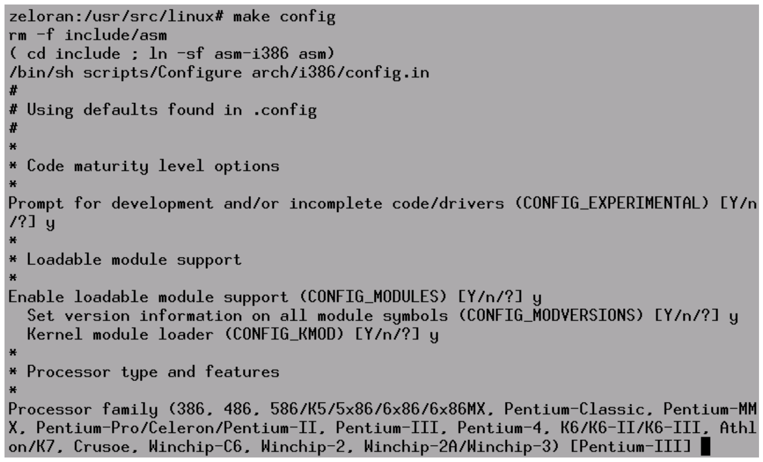
Habiendo enumerado los pasos para la actualización del núcleo en el sistema, vamos a abocarnos a una de las etapas quizás mas complicadas del proceso.

Un detalle muy importante a tener en cuenta a la hora de compilar un núcleo, es que se deben tener instalados en el sistema los paquetes de desarrollo. El núcleo Linux está programado en casi su totalidad en C, por lo tanto se debe tener instalados la biblioteca C de desarrollo y el compilador de dicho lenguaje.

Dependiendo de la distribución que se utiliza, el método de instalación de los paquetes difiere un poco. Dejando de lado el formato de paquetes y la distribución, se debe revisar si los paquetes `gcc` y `libc6-dev` se encuentran instalados, estos paquetes siempre tienen un nombre similar en cualquier distribución.

Existen 3 formas de configurar las opciones y características a incluir en el núcleo, una es usando un programa de línea de comandos, que línea a línea va preguntando las diferentes opciones, este método era el que se utilizaba hace algunos años, los demás métodos son mas agradables. El segundo método es mediante el uso de una interfaz de texto, pero a pantalla completa, con uso de la biblioteca

[6]Ni hablar de cuando uno se olvidaba de incluir una opción al núcleo y debía recompilar todo, la industria de los calmantes habrá ido en alza en esa época.



```
zeloran:/usr/src/linux# make config
rm -f include/asm
( cd include ; ln -sf asm-i386 asm)
/bin/sh scripts/Configure arch/i386/config.in
#
# Using defaults found in .config
#
*
* Code maturity level options
*
Prompt for development and/or incomplete code/drivers (CONFIG_EXPERIMENTAL) [Y/n
/?] y
*
* Loadable module support
*
Enable loadable module support (CONFIG_MODULES) [Y/n/?] y
  Set version information on all module symbols (CONFIG_MODVERSIONS) [Y/n/?] y
  Kernel module loader (CONFIG_KMOD) [Y/n/?] y
*
* Processor type and features
*
Processor family (386, 486, 586/K5/5x86/6x86/6x86MX, Pentium-Classic, Pentium-MM
X, Pentium-Pro/Celeron/Pentium-II, Pentium-III, Pentium-4, K6/K6-II/K6-III, Athl
on/K7, Crusoe, Winchip-C6, Winchip-2, Winchip-2A/Winchip-3) [Pentium-III]
```

Figura 11.1: Configurando el núcleo con la interfaz de línea de comandos

ncurses, si se quiere utilizar este método, se deben tener instalados en el sistema dicha biblioteca (normalmente se instala por defecto), cuyo paquete generalmente se denomina `libncurses`. El tercer y último método es mediante el uso de una interfaz gráfica en las X; este método utiliza las bibliotecas *Tcl/Tk*, que también normalmente se instalan por defecto en el sistema.

El primer método es el que se utilizaba con las primeras versiones del núcleo, como se puede observar en la figura 11.1, no es muy amigable pero a veces puede resultar útil cuando no se tiene disponible una interfaz gráfica ni las bibliotecas *ncurses*. Estando en el directorio del código fuente del núcleo, se ejecuta el siguiente comando:

```
root@maquina:/usr/src/linux# make config
```

Siempre hay que tener en cuenta el usuario que se está utilizando en esta tarea, si bien el compilar un núcleo no requiere de utilizar el usuario administrador, dependiendo del directorio donde esté alojado el árbol de fuentes será o no necesario usar la cuenta *root*. Como generalmente el directorio donde se aloja el código fuente del núcleo es `/usr/src`, se necesitará usar la cuenta de administrador, a menos que dicho directorio posea permisos especiales para algún otro usuario del sistema.

El segundo método mencionado corresponde al uso de la interfaz de texto a pantalla completa, como se ve en la figura 11.2. Este método se usará en el resto del curso.

Esto se obtiene ejecutando el siguiente comando:

```
root@maquina:/usr/src/linux# make menuconfig
```

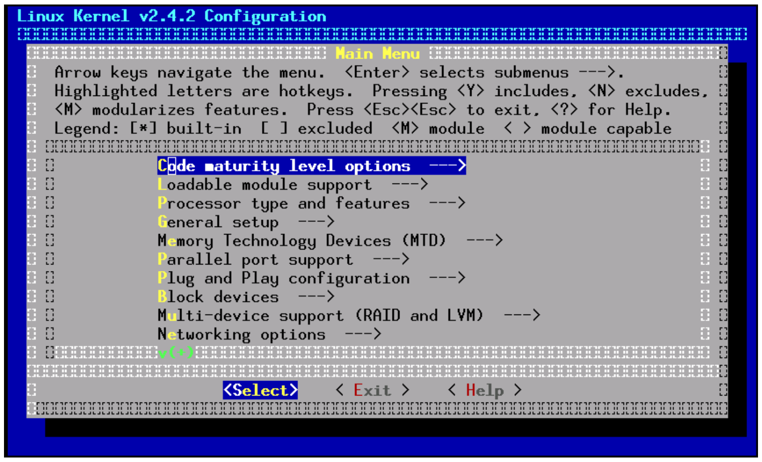


Figura 11.2: Interfaz de texto a pantalla completa

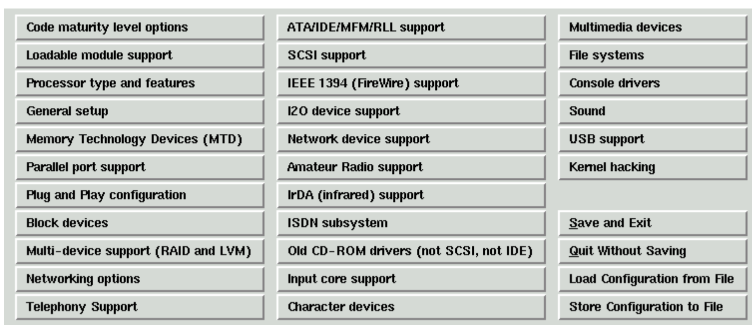


Figura 11.3: Interfaz gráfica para configurar el núcleo

El tercer método utiliza una interfaz gráfica en las X (figura 11.3). Es el método mas práctico, siempre y cuando se disponga de las X, claro está. Por esa razón hemos seleccionado el anterior método para las demás figuras, aunque se debe saber que los tres métodos poseen los mismos contenidos, solo cambia la manera de desplegar la información.

Este tercer método se obtiene ejecutando desde una terminal X lo siguiente:

```
root@maquina:/usr/src/linux# make xconfig
```

Una vez presentados los métodos de configuración del núcleo, nos toca seleccionar las opciones y características que el nuevo núcleo incluirá. Lo que siempre se recomienda al agregar opciones en el núcleo, es que aquellas opciones que tengan la posibilidad de ir como módulos, que lo hagan. Con esto, obtendremos un núcleo mas pequeño, y los módulos que se han compilado se irán cargando y descargando automáticamente, a medida que se necesiten activar ciertas funcionalidades, y luego ya no se necesiten. Al tener menos cantidad de código en memoria, no sólo se produce un ahorro de RAM (que es prácticamente insignificante en estos días ahorrar unos 100k de RAM), sino que lo mas importante es el tener menos posibilidades que algo falle. El sentido común dice que un núcleo de 400Kb en memoria mas 200Kb de módulos no cargados tendrá menos posibilidades de falla que el mismo núcleo, pero con sus 600Kb completos en memoria.

¿Cómo saber cual función debería ir en módulo y cual en el núcleo? Lo que se recomienda es tener en cuenta la necesidad del uso de cada funcionalidad del núcleo para realizar esta decisión. Por ejemplo, una máquina que posee discos rígidos IDE, los va a necesitar usar en todo el tiempo que esté encendido el equipo (salvo raras ocasiones), entonces el soporte para discos IDE debería ir en el núcleo y no como un módulo, ya que de ir como módulo, estaría siempre cargado. Un servidor de red necesitará el manejador de la tarjeta de red siempre activo, entonces conviene compilarlo en el núcleo, pero el uso del módem de dicho equipo es muy esporádico, en este caso el manejador del módem o los puertos serie podría ir como módulos. También hay que tener en cuenta algo importante: el núcleo necesita saber como manejar un disco rígido para poder iniciar todo el sistema (si el sistema está almacenado en un disco rígido, como normalmente ocurre), si se compila el soporte de discos rígidos (cualquiera sea el tipo) como módulo, el núcleo necesitará poder manejar el disco rígido al iniciar el equipo para poder cargar el módulo para poder manejar el disco rígido[7], y obviamente este tipo de casos causan problemas.

Resumiendo, hay dos casos en los cuales una funcionalidad del núcleo no debería ser compilada como núcleo:

- La funcionalidad en cuestión es esencial para el correcto arranque del sistema.

- Dicha funcionalidad va a utilizarse en todo momento del funcionamiento del sistema.

Para los demás casos es conveniente compilar como módulo. El sistema se encargará de cargar el módulo cuando se necesite, y cuando ya no se use, se descargará para liberar memoria.

Cuando se ejecuta el comando `make menuconfig` se presenta una serie de secciones donde se agrupan las diferentes capacidades del núcleo, esta lista que sigue proporciona una breve explicación de los contenidos de cada una de estas

[7] La recursión es algo muy interesante, pero en estos casos no aporta nada útil.

secciones[8]:

Code maturity level options Se presenta una única opción, al activarla, será posible activar aquellas funcionalidades experimentales que tenga el núcleo, generalmente esto no es recomendable cuando se está compilando un núcleo para algún servidor de producción, a menos que se tenga la certeza de que tal o cual opción es estable y funciona bien (o sea imprescindible).

Loadable module support Las opciones que se incluyen en esta sección tienen que ver con el manejo de los módulos del núcleo. En algunos casos no será necesario o no se podrá tener módulos, entonces se pueden desactivar. En este método de configuración del núcleo, aquellas opciones del mismo que sea posible compilar como módulo tendrán la casilla de selección a la izquierda de esta forma: `< >`, cuando se llega a alguna opción con este tipo de casilla de selección, oprimiendo `Y` se la selecciona para compilarse dentro del núcleo, si se presiona `M` se la seleccionará para compilarse como un módulo.

Processor type and features Dentro de esta sección encontraremos opciones de optimización del núcleo para el tipo de procesador que se posea. Con cada nueva serie de procesadores, características nuevas van siendo disponibles para ciertas funciones, lo que aprovecharlas es interesante para aumentar el desempeño general del sistema. El núcleo Linux soporta también equipos con mas de un procesador, en caso de estar compilando el núcleo para un equipo de este tipo, en esta sección se debe activar la opción `Symmetric multi-processing support` para sacar el máximo provecho.

General setup Como su nombre lo indica, esta sección contiene opciones de configuración general del núcleo, entre ellas se encuentran opciones especificando la marca y el modelo del *chipset* de la placa madre del equipo, así como también si el núcleo utilizará el sistema de administración de energía que provee el BIOS, el formato en el cual el núcleo será compilado, si se soportarán dispositivos PCMCIA, etc.

Memory Technology Devices (MTD) Esta sección en nueva de los núcleos de la serie 2.4.x, posee una opción para dar soporte a los dispositivos de discos en memoria, para poder manejar sistemas de archivos de estado sólido en dispositivos embebidos. Normalmente esta opción no se utilizará.

[8] Se toma la versión 2.4.2 del núcleo para las explicaciones, pero se supone que durante toda la serie 2.4.x estas secciones y sus contenidos no cambiarán demasiado

Parallel port support Dentro de esta sección se encuentra la opción que activa el soporte para el puerto paralelo. Generalmente utilizaremos el puerto paralelo si en la máquina se tendrá alguna impresora de puerto paralelo instalada, alguna unidad de disco Zip, o cualquier otra unidad de disco por puerto paralelo (grabadoras de CD-R, por ejemplo) o en el caso de utilizar un cable paralelo para conectarse con otra máquina. Se debe tener en cuenta que esta opción se refiere solamente al soporte de base para todos aquellos dispositivos que utilicen el puerto paralelo, en otras secciones se debe después confirmar el soporte individual para aquellos dispositivos específicos.

Plug and Play configuration Hace unos años se ha impuesto como estándar una norma para periféricos de tal modo que no tengan que ser configurados sus parámetros (IRQs, direcciones de E/S, etc.) manualmente, sino que el sistema lo pueda manejar lo mas convenientemente posible, de esta forma, el sistema operativo podría lidiar con los conflictos y solucionar los problemas que puedan surgir, liberando al usuario de estas tareas que a veces suelen ser complicadas de solucionar. Este estándar se conoce como *Plug&Play* (o abreviado, PnP), y aunque inicialmente los dispositivos PnP no funcionaban muy bien, hoy en día es mas común ver este tipo de periféricos que los otros. El núcleo Linux tiene soporte para estos dispositivos mediante la opción dentro de esta sección. Además, se cuenta con un paquete de software llamado `isapnptools` que es muy útil para configurar los antiguos dispositivos PnP ISA (normalmente, los módems).

Block devices En esta sección se presentan opciones para dar soporte a varios tipos dispositivos de bloques, incluyendo unidades de discos flexibles, discos IDE por puerto paralelo, discos en memoria RAM, y otros mas.

Multi-device support (RAID and LVM) Esta sección contiene opciones de configuración que muchas veces se utilizan en equipos servidores. RAID es una norma que sirve para generar arreglos de múltiples discos de manera de obtener redundancia de información y evitar problemas de corrupción de datos en el caso de que algún disco del arreglo tenga una falla. RAID tiene varios modos de funcionamiento, generalmente en los servidores se utilizan placas controladoras de discos RAID (soporte de hardware) que funcionan en forma transparente y el sistema operativo no se da cuenta, sin embargo el núcleo Linux da la posibilidad de poder utiliza discos comunes como discos RAID (soporte por software) teniendo de esta manera un elemento mas de seguridad a bajo costo. LVM es otro método de combinación de discos, pero para la generación de volúmenes o grupos de volúmenes de lógicos, cuyo tamaño puede variar en tiempo de corrida del sistema, sin tener que reiniciar ni parar servicios. LVM no se utiliza para dar redundancia de datos, sino para combinar varios discos (de distintos tipos) y utilizarlos como si fuera un gran disco grande.

Networking options En esta sección se encuentran las opciones de soporte base de redes. Se podrán activar los soportes de varios protocolos, con sus opciones específicas, el mas común en estos días es el protocolo *TCP/IP*, uno de los protocolos que se utiliza en Internet. Otra opción bastante novedosa en los núcleos, es la opción QoS (Quality of Service) la cual se utiliza para definir los algoritmos de planificación de paquetes de red, pudiendo partir un enlace en varios sub-canales por decirlo de alguna manera, y de esta forma poder asignar cierto ancho de banda a diferentes clientes. Tener en cuenta que esta sección no se trata acerca de soporte a periféricos de red, sino a protocolos, para tener soporte de una placa de red en especial, se debe activar en otra sección.

Telephony Support En esta sección se da soporte a los dispositivos VoIP (Voice over IP), que se utilizan para hablar por teléfono a través de redes de datos, en vez de redes de telefonía. Estos dispositivos cada vez se están viendo con mas frecuencia, quien sabe, en unos años quizás podremos hablar al otro lado del mundo sin tener que pagar altísimos costos.

ATA/IDE/MFM/RLL support Aquí se da soporte a los discos IDE, sean discos rígidos, flexibles, CD-ROMs, etc. También hay varias opciones para especificar la marca de controladora de disco que se posee, y otros detalles para mejorar la performance. El núcleo Linux puede manejar sin problemas los nuevos discos con UDMA.

SCSI support También existe soporte para los dispositivos SCSI, en esta sección están las opciones para definir los tipos de dispositivos SCSI que se dará soporte, mas otras opciones de control. Un ejemplo común para esta sección es el soporte para las unidades de discos Zip por puerto paralelo, En realidad estas unidades trabajan con la norma SCSI, y es por eso que parte de las opciones que se necesitan para poder hacer funcionar estos periféricos, se encuentran en esta sección.

IEEE 1394 (FireWire) support FireWire es una nueva clase de bus de datos, de alta velocidad que normalmente se utiliza para transmitir vídeo. Originariamente en las Macintosh, esta clase de conexiones se las utiliza para conectar una cámara de vídeo al equipo y capturar vídeo sin problemas de lentitud. Actualmente existen placas con puertos FireWire para PCs compatibles con IBM.

I2O device support La arquitectura *I2O* se ha pensado para dividir los manejadores de dispositivos I2O en dos partes: una dependiente del sistema operativo y otra no, de manera de que el fabricante del dispositivo en cuestión solamente deba hacer un solo manejador (la parte no dependiente del sistema operativo) y de esta forma poder utilizarlo en cualquier sistema operativo compatible con I2O. El autor[9] no conoce actualmente ningún dispositivo de esta clase, seguramente con el tiempo irán siendo mas comunes.

Network device support Esta sección engloba a todas las opciones de soporte de interfaces de red, desde ethernet de 10, 100 y 1000 Mbps, pasando por SLIP, PPP, PLIP, hasta interfaces WAN, Token Ring y otras. Normalmente se deberán activar las opciones de soporte para placas ethernet e interfaces PPP (si es que se utilizará módem).

Amateur Radio support Los aficionados a las comunicaciones radiales no quedan de lado en lo que respecta a soporte en el núcleo. Linux tiene soporte para varios TNCs, o para simular un TNC usando una placa de sonido, de tal forma de poder conectarse con otros equipos mediante el protocolo AX.25.

IrDA (infrared) support Esta sección generalmente es para las notebooks, que poseen un puerto infrarrojo. IrDA es un protocolo de transferencia de datos vía infrarrojo que se encuentra normalmente en PDAs (como las Palm) permitiendo así sincronizar los datos de la PDA en una notebook sin necesidad de conectarla con un cable serie o USB.

ISDN subsystem ISDN (o RDSI, como se lo llama en España) es un tipo especial de servicios digitales de telefonía. Normalmente se lo utiliza para conectarse a Internet, teniendo una velocidad mucho mayor que conectando se con módem. Actualmente se está dejando de lado ISDN en favor de ADSL, en Argentina por ejemplo, ISDN no se ha visto, por su alto costo.

Old CD-ROM drivers (not SCSI, not IDE) En esta sección se da soporte a aquellas unidades de CD-ROMs antiguas que no eran compatibles con las normas IDE ni SCSI. Generalmente eran unidades que había que conectar a placas de sonido, actualmente no se ven esta clase de unidades de CD-ROM.

Input core support Las opciones que esta sección provee, permiten la configuración de dispositivos de entrada del tipo USB, tales como teclados, mouses, etc.

Multimedia devices Esta sección contiene las opciones necesarias para dar soporte a dispositivos de captura de vídeo y tarjetas de radio.

[9] En realidad, uno de los autores (Lucas Di Pentima)

File systems Los sistemas de archivos son los formatos que se le dan a las particiones para poder almacenar archivos, administrar los atributos de cada archivo, etc. Esta sección posee las opciones de soporte de sistemas de archivos tanto nativos de GNU/Linux como de otros sistemas operativos, como por ejemplo FAT32. Es importante tener en cuenta que el soporte para el sistema de archivo que se esté utilizando en el sistema debe estar compilado dentro del núcleo y no como módulo, normalmente este sistema es el *Second extended filesystem* (e2fs). En los últimos núcleos se puede configurar el soporte para un nuevo sistema de archivos nativo de GNU/Linux: el *Reiser filesystem* (reiserfs), el cual es un sistema de archivos jornalizado, que provee ciertas características que lo hacen mas seguro que el e2fs. Además se provee opciones para dar soporte a varios sistemas de archivos de red, que permiten utilizar discos remotos como si fueran locales al equipo, como por ejemplo NFS (generalmente se utiliza en los UN*X), SMB (plataformas Windows) y NCP (volúmenes NetWare).

Console drivers Aquí se encuentran las opciones para configurar el tipo de consola que se usará en el sistema. Dependiendo de la tarjeta de vídeo que se posea, se puede configurar para que la consola tenga el típico formato de 80x25 caracteres, o mas resolución. También se puede configurar el núcleo para soportar mas de una placa de vídeo, pudiendo tener dos monitores, uno como consola de texto y otro como consola gráfica.

Sound Esta sección contiene las opciones de sonido, pudiendo configurar una gama de tarjetas de sonido bastante amplia.

USB support Esta sección provee soporte para distintos dispositivos USB, desde escáneres, dispositivos de almacenamiento de datos, reproductores de MP3, módems, impresoras hasta adaptadores de red.

Kernel hacking Esta sección contiene una opción que se utiliza cuando se prueban núcleos en desarrollo, generalmente no se necesitará (ni tampoco será muy seguro) activar esta opción, a menos que se esté colaborando con el desarrollo del núcleo.

Una vez seleccionadas todas las opciones, se debe salir del sistema de configuración del núcleo y salvar los cambios. Las opciones seleccionadas del núcleo se guardarán en el un archivo llamado `.config` dentro del árbol de fuentes del núcleo. Esto es interesante saberlo para poder copiarlo en algún sitio como respaldo y no perder las configuraciones del núcleo si necesitamos alguna vez borrar todos los archivos (por cuestión de espacio, por ejemplo).

Una vez que se ha salido del sistema de configuración del núcleo, antes de compilar se deben generar las dependencias con el siguiente comando:

```
root@maquina:/usr/src/linux# make dep
```

Luego se compila el núcleo:

```
root@maquina:/usr/src/linux# make bzImage
```

Esto puede tardar algunos minutos, dependiendo de la potencia del procesador y la cantidad de memoria RAM del equipo.

La opción *bzImage* (tener en cuenta las mayúsculas) le dice al compilador que genere el archivo del núcleo y lo comprima con `bzip2`. Antes se utilizaba la opción *zImage* que generaba un núcleo comprimido con `gzip`, pero tenía un límite de tamaño que actualmente se ha sobrepasado por la cantidad de opciones que posee el núcleo. `bzip2` comprime mejor que `gzip`.

Luego de generar el núcleo, se deben generar los módulos, con el comando:

```
root@maquina:/usr/src/linux# make modules
```

Esta etapa quizás tome mas tiempo que la anterior, dependiendo de la cantidad de opciones que se hayan elegido como módulos.

Una vez finalizado, se deben instalar los módulos de la siguiente manera:

```
root@maquina:/usr/src/linux# make modules_install
```

Y por último, se debe copiar el archivo del núcleo a algún sitio donde estén alojados normalmente los núcleos, como por ejemplo el directorio `/boot`. El archivo del núcleo, suponiendo que el conjunto de archivos fuente del núcleo se haya almacenado en el directorio `/usr/src/linux`, se encuentra en `/usr/src/linux/arch/i386/boot/bzImage`.

Como último paso, se debe configurar el `LILO` para poder probar el nuevo núcleo, pero esto se verá en la sección 11.2.5.

A modo de información, es conveniente comentar el uso de una utilidad bastante práctica. El comando `dmesg` imprime en pantalla los diferentes mensajes de inicio que el núcleo muestra cuando se arranca el sistema. Esto puede ser muy útil cuando se necesita chequear mensajes de error que no se han podido leer bien en el arranque.

11.2.5 LILO

Al hablar del núcleo es inevitable hablar del cargador de Linux o bien conocido como LILO[10] que es el encargado de cargar en memoria el núcleo y largarlo a correr.

[10] LInux LOader: Cargador de Linux

El núcleo es un archivo mas. Normalmente se encuentra en el disco rígido[11]. Similar a un archivo ejecutable, algún proceso debe ser el encargado de cargarlo y luego ejecutarlo. Como todavía no se encuentra nada en memoria, la BIOS ejecuta código de un lugar especial en el disco, llamado *boot sector*, que contendrá a LILO.

Una de las grandes funciones de LILO es la selección de núcleo a usar. Normalmente al compilar diferentes núcleos hay que elegir, por ejemplo, entre alguno que tenga soporte para *clusters*[12] o para emular *SCSI* con un dispositivo *IDE*[13].

También se pueden tener núcleos de otros sistemas operativos (en el caso de algún problema serio neurológico) como OS/2, toda la gama de Windows, otros UNIX, etc.

El núcleo de Linux acepta parámetros para personalizarlo o en el caso de que no pueda auto detectar ciertos dispositivos o recursos. Estos parámetros deben darse antes de que se cargue el núcleo en si. Un ejemplo sería:

```
LILO: linux mem=256M
```

en este caso se saltea la auto detección de cantidad de memoria realizada por Linux y se presume que existen 256 MB de memoria.

Una lista más detallada de estos parámetros se encuentra en `/usr/src/linux/Documentation`. En este directorio está toda la documentación de los desarrolladores de núcleo, separada por módulo.

Configurando LILO

Toda la configuración de LILO se encuentra en `/etc/lilo.conf` el contenido es similar a algo así:

```
boot=/dev/hda
install=/boot/boot.b
default=linux
prompt
timeout=5
message=/boot/message
image=/boot/vmlinuz
        label=linux
        root=/dev/hda6
```

[11] Es probable que en otros cursos veamos como arrancar una máquina sin disco rígido a través de la red

[12] Cluster es una forma de utilizar varias computadoras para que todas calculen al mismo tiempo como si fuera una sola

[13] Es muy común cuando se desea utilizar una grabadora de CD-R

```
        append=" hdc=ide-scsi ide1=autotune ide0=autotune"
other=/dev/hda2
        label=windows
        table=/dev/hda
```

Por ahora esto puede parecer inentendible pero vamos a analizar linea a linea:

```
boot=/dev/hda
```

Significa que el dispositivo de arranque es `/dev/hda`[14]. El *sector de arranque* o *boot sector* de ese dispositivo contendrá a LILO cuando inicie el equipo.

```
install=/boot/boot.b
```

`/boot/boot.b` es un archivo usado como nuevo sector de arranque.

```
default=linux
prompt
timeout=5
```

Con estas tres opciones se especifica que:

prompt Pregunte que núcleo hay que utilizar (el caso contrario puede ser que haya uno sólo y no se quiera elegir).

default En caso de no poner nada se elija “linux”.

timeout tiempo en segundos a esperar si no se elije nada.

```
message=/boot/message
```

Se muestra un mensaje que es el archivo `/boot/message` que puede contener algo como:

```
Bienvenido a LILO, el selector de SO de arranque!

Elija un sistema operativo de la lista.
O espere 5 segundos para que arranque el sistema predeterminado.
```

[14] `/dev/hda` el disco maestro de la controladora IDE primaria

Luego vienen las configuraciones de los núcleos en si. En el ejemplo existen 2 núcleos, uno de linux y el otro es un *Windows*.

Las dos configuraciones son distintas pero tienen una linea en común. Esta es `label`. `label` es el identificador de núcleo para LILO, es de suponer que tiene que ser único. Puedo tener varios núcleos de linux pero no con `label=linux` en mas de uno de ellos. Simplemente habrá que asignarlos de distinta manera como por ejemplo `label=linux-2.2.19` y `label=linux-2.4.3`.

Si por un momento repasamos este concepto, nos vamos a dar cuenta que `default=linux` hace referencia al núcleo que posee `label=linux`. Cuando cambiemos de configuración a `label=linux-nuevo` recordemos cambiar `default` también.

Para correr un sistema no sólo necesitamos el núcleo, sino también archivos, que componen el árbol de directorios que surge de la *raíz* o *root*.

Por eso,

```
root=/dev/hda6
```

especifica que se va a usar la sexta partición del disco como *directorio raíz* o simplemente *raíz*, es decir que, todo lo que esté en esa partición va a pasar a ser el directorio `/` donde estarán `/bin`, `/etc`, `/home`, `/usr`, `/lib`, etc.

En este concepto independizamos el núcleo de los archivos que maneja. Una vez que está el núcleo corriendo, los archivos se pueden obtener de diferentes lugares. Por ejemplo particiones, otros discos, discos flexibles, hasta un dispositivo que se encuentra a través de una red[15]. Casi cualquier archivo/dispositivo puede ser `root` siempre que este *formateado* correctamente.

Por ultimo la linea:

```
append=" hdc=ide-scsi ide1=autotune ide0=autotune"
```

pasa parámetros al núcleo para ajustar configuraciones, las cuales dependerán de cada sistema.

Instalando LILO

Una vez que está correctamente configurado (mediante el archivo `/etc/lilo.conf`), es necesario escribir el sector de arranque del dispositivo[16].

Un error común es pensar que sólo editando el archivo se guarda la configuración, hay que recordar que es un archivo más, incluso se puede utilizar otro archivo.

Para grabar hay que ejecutar:

[15] un hipotético `/dev/red` o comúnmente denominado `/dev/nfsroot`

[16] Utilizamos la palabra *dispositivo* en vez de *disco* porque puede ser que estemos configurando otra alternativa de arranque.

```
root@maquina:/root# lilo
Adding linux *
Adding windows
root@maquina:/root#
```

Y listo. Si es que no surgió ningún problema.

El asterisco (`*`) indica que núcleo se cargará por defecto (*default*). En nuestro caso es la entrada que contiene `label=linux`.

La tecla `TAB` muestra todas las posibilidades de nucleos a cargar. Obviamente muestra el contenido de `label`.

11.2.6 Módulos

Una mejora extraordinaria al núcleo fue la *modularización* del mismo. En un principio el núcleo era *monolítico*, es decir, un gran archivo que contenía todos los controladores para los dispositivos.

Un núcleo monolítico es más eficiente que uno modularizado, en parte porque toda referencia se conoce en tiempo de compilación y por otro lado el sistema entero está en memoria siempre. Como desventaja tiene su gran tamaño, poca flexibilidad de incorporar nuevos controladores y no acepta cambios en el código existente.

Los módulos como contrapartida, se pueden cargar y descargar de memoria en cualquier momento. Dando la libertad de poder utilizar sólo lo necesario. Y si estamos programando un controlador para cualquier periférico, compilamos el controlador, lo cargamos a memoria, lo probamos y luego se puede sacar de memoria, recompilar y seguir probando. Todo esto sin rearrancar el sistema, ni cerrar los programas que estamos usando.

Cómo generar un módulo

Cuando compilamos el núcleo debemos especificar cuales controladores se compilaran como módulos. Esto es muy sencillo, sólo hay que poner la letra `M` en el menú.

Siempre que terminamos de configurar la opciones del núcleo hay que ejecutar `make dep`.

Ejecutando `make modules` se compilarán todos los módulos que sean necesarios. Esto puede tardar desde unos pocos segundos hasta una hora, dependiendo del hardware, la configuración (cuantos módulos se eligieron) y la versión del núcleo.

Lo único que falta es copiar los módulos recién compilados al lugar indicado (el directorio `/lib/modules/(versión del Núcleo)/`). Esto se puede hacer *manualmente* con `cp` o tipear `make modules_install`.

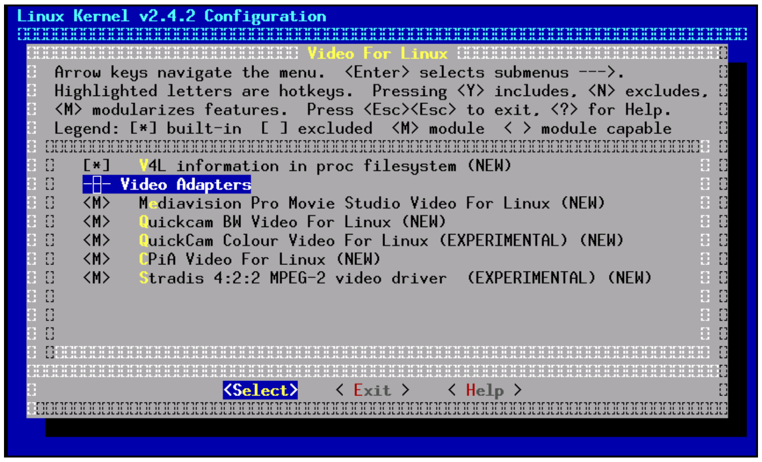


Figura 11.4: Seleccionando opciones como módulos

Mostrando los módulos cargados

Una vez que tenemos varios módulos en el directorio `/lib/modules/(versión del Núcleo)/`[17] podemos listar aquellos que están siendo usados.

El comando `lsmod` muestra los módulos usados. Una salida podría ser:

```
root@maquina:/root# lsmod
Module                  Size  Used by
loop                    9600   2  (autoclean)
lockd                  32208   1  (autoclean)
sunrpc                 54640   1  (autoclean)  [lockd]
autofs                  9456   2  (autoclean)
8139too                12064   1  (autoclean)
via82cxxx_audio         9024   0
soundcore               2800   2  [via82cxxx_audio]
ac97_codec              7088   0  [via82cxxx_audio]
ip_masq_vdolive         1440   0  (unused)
ip_masq_cuseeme         1184   0  (unused)
ip_masq_quake           1456   0  (unused)
ip_masq_irc             1664   0  (unused)
ip_masq_raudio          3072   0  (unused)
ip_masq_ftp             4032   0  (unused)
nls_cp437               3952   5  (autoclean)
```

[17]Este curso muestra como generarlos, pero, la mayoría de las distribuciones ya vienen casi todos los módulos compilados

```
vfat                    9408   2  (autoclean)
fat                    30432   2  (autoclean) [vfat]
supermount             14224   3  (autoclean)
ide-scsi                7664   0
reiserfs              128592   2
root@maquina:/root#
```

Como también podría estar vacía. Si es que ningún módulo se cargó o si el núcleo es monolítico.

Tomemos el caso del módulo `soundcore`:

```
soundcore               2800   2  [via82cxxx_audio]
```

El tamaño en memoria del módulo es de 2800 bytes. Y el módulo `via82cxxx_audio` lo está usando. Esto quiere decir que para sacar de memoria a `soundcore` primero hay que sacar a `via82cxxx_audio`. Y viceversa, si necesitamos agregar `via82cxxx_audio` primero tendremos que agregar `soundcore`.

Podemos darnos cuenta de que existe un árbol de dependencias entre módulos. Y en algún lugar debe estar. Bueno así es, es el archivo `/lib/modules/(versión de núcleo)/modules.dep` y es generado en la compilación.

Agregando módulos

La forma de agregar un módulo es relativamente simple. El comando es `insmod` y su sintaxis es:

```
insmod modulo [parametros]
```

Siendo `modulo` el nombre del módulo y `parametros` los parametros de ese modulo, que configuran al dispositivo que controla. La documentación de los parámetros se encuentran en `/usr/src/linux/Documentation`.

El gran inconveniente de `insmod` es que no controla las dependencias necesarias, sólo intenta cargar el módulo, si la operación no tiene éxito, finaliza su ejecución.

Debido a que es casi imposible tener en mente todo el árbol de dependencias, existe una utilidad que realiza comprobaciones. Esta utilidad es `modprobe`. `modprobe` utiliza `insmod` en el orden correcto y su sintaxis es:

```
modprobe modulo [parametros]
```

Retirando módulos de memoria

También se pueden retirar módulos de memoria. El comando es `rmmod` y es similar al `insmod` en lo que respecta a comprobaciones en el árbol de dependencias.

Un módulo puede estar siendo utilizado por otro módulo, pero también puede estar siendo utilizado por un programa de usuario. En este caso el usuario va a tener que liberar el dispositivo/recurso antes de poder retirar el módulo de memoria.

11.2.7 Automatizando un poco mas los módulos

Configuraciones en `/etc/modules.conf`

Como estamos operando frente a una máquina trataremos de automatizar lo más posible las tareas rutinarias. Se supone que los cambios de configuración en el hardware se hacen infrecuentemente. Entonces los parámetros en la carga de módulos es siempre la misma.

La gran mayoría de los módulos auto detectan su configuración, pero en ciertas ocasiones hay que parametrizarla. Una alternativa poco elegante sería crear un script que cargue al módulo con los parámetros correspondientes. Pero se vuelve engorroso tener un script por módulo.

En reemplazo a eso, los comandos `insmod` y `rmmod` utilizan una archivo de configuración: `/etc/modules.conf`.

Este archivo puede contener una línea del estilo

```
option nombre-modulo opt-1 [opt-2 .. opt-n]
```

Donde `opt-1`, `opt-2`, etc. son las opciones o parámetros del módulo.

Una configuración interesante es la creación de sobrenombre o *alias* a los módulos. Sirve para no tener que recordar nombres como `via82cxxx_audio` y en reemplazo usar `placa_sonido` por ejemplo. Su sintaxis es:

```
alias sobre-nombre nombre-modulo
```

Algunos scripts de configuración del sistema tienen palabras definidas para cargar los módulos correspondientes y que el usuario edite `/etc/modules.conf` y asigne el modulo.

Un ejemplo clásico sería:

```
alias sound sb
alias eth0 ne2k-pci
```

que asigna el seudónimo `sound` a un módulo de *Sound Blaster* y `eth0`[18] al módulo `ne2k-pci` para la placa de red.

[18] Corresponde a la primera placa de red Ethernet

Auto cargado de módulos

A medida que se utilizaban los módulos, era conveniente tener una utilidad que cargue el o los módulos necesarios para hacer una tarea. Esta utilidad está contenida en el núcleo, se llama `kmod` y se configura en

Loadable Module Suport → *Enable Loadable Module Support* → *Kernel Module Loader*.

Con sólo incluir eso, prácticamente no hay que cargar «a mano» ni ningún módulo.

Práctica - 11.2

1- Compilar un núcleo monolítico y uno modular con las mismas opciones de configuración, comparar los tamaños (tener en cuenta el tamaño de los módulos). Sacar conclusiones.

2- Utilizar LILO para hacer múltiples entradas a ambos núcleos, además de los núcleos que ya existían en el sistema.

3- Probar los núcleos y verificar si existen errores al iniciar. Si da errores, ¿qué tipo de errores da?. Si uno de ellos únicamente tiene errores y ambos fueron configurados con las mismas opciones, ¿por qué cree que da errores?.

4- Arrancar el sistema con ambos núcleos y comparar los tamaños de éstos en memoria. ¿Cuál de ellos tiene ventaja en la utilización de recursos?. ¿Y en el tiempo de ejecución?. **Nota**: utilizar `lsmod` para calcular el tamaño de los módulos en ejecución y `dmesg | grep Memory` para el tamaño del núcleo

11.3 Instalación y actualización de paquetes utilizando RPM/DEB

11.3.1 ¿Qué paquetes instalar para un servidor promedio?

11.3.2 ¿Por qué no debo instalar todo?

11.3.3 ¿Qué me conviene para una estación de trabajo promedio?

11.4 Programación de tareas con `cron` y `at`

11.5 Creación y chequeo de sistemas de archivos

11.6 Copias de respaldo, utilidades de archivado y compresión

Capítulo 12

Conocimientos básicos de redes

12.1 Nociones de TCP/IP.

12.2 Conceptos básicos de interfaz de red y ruteo

12.3 Conexiones punto a punto

12.3.1 PPP

12.3.2 SLIP

12.3.3 PLIP

12.4 Interfaces ethernet

12.5 Conceptos básicos de filtrado de paquetes (ipfwadm, etc...ver kernel 2.4)

12.6 Conceptos básicos de enmascaramiento de IPs

Capítulo 13

Conceptos y configuración básica del servidor DNS

Capítulo 14

Configuración básica de Samba

Capítulo 15

Consiguración básica de Apache

Capítulo 16

Configuración básica de Squid

Capítulo 17

Configuración de Leafnode/INN.

17.1 Leafnode.

17.1.1 Introducción.

El servidor Leafnode se utiliza para redes pequeñas que necesiten un servidor de noticias offline, esto es, leer las noticias desconectados de la red, o cuando se tiene una conexión lenta.

Leafnode se utiliza para replicar grupos de noticias que obtiene desde otro servidor ya que, por el momento, no es capaz de generar grupos locales, aunque parace que la próxima versión soportará esta eventualidad.
Su mayor ventaja está en su pequeño tamaño y su fácil instalación. Por contra, sólo se puede utilizar para pocos usuarios puesto que no es escalable. Si se necesita dar servicio de noticias a una gran cantidad de usuarios hay mejores alternativas, como INN.

17.1.2 Obtención e Instalación.

Se puede obtener Leafnode desde la página oficial del programa `http://www.leafnode.org`. Si se quieren bajar paquetes en formato rpm se puede acudir a `http://rpmfind.net` para el formato Debian se deberá acudir a `http://www.debian.org`. Aunque normalmente el paquete de Leafnode viene con la mayoría de las distribuciones.

Su instalación es sencilla:

- Paquetes RPM: desde la línea de comandos ejecutaremos la orden `rpm -i leafnode-1.9.17-1.rpm`, teniendo en cuenta que la versión puede

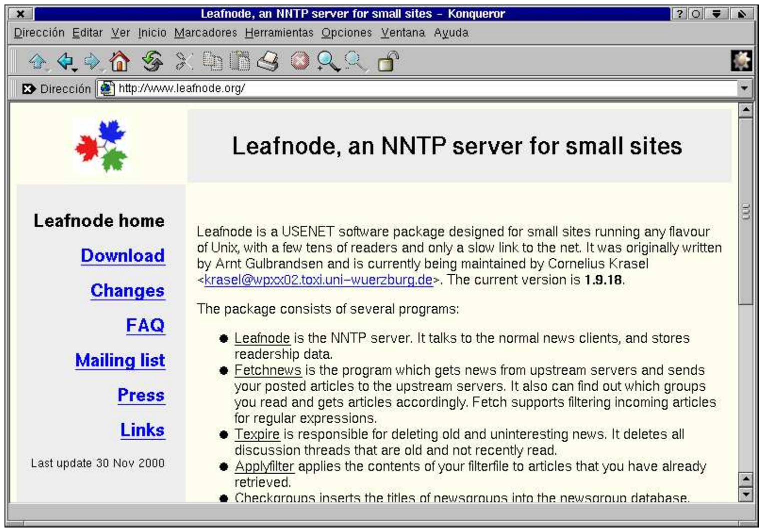


Figura 17.1: Página de Leafnode

ser distinta, en este caso, 1.9.17-1.

- Paquetes Deb: desde la línea de comandos ejecutamos `apt-get install leafnode` y con esto nos aseguramos que se instalan todas las dependencias necesarias.

- Código fuente: procederemos a desempaquetar el archivo con las fuentes con la orden `tar xvzf archivo` y buscaremos el fichero README para ver la información de compilación.

Una vez realizada la instalación, habrá que verificar la existencia del usuario news. Dentro del fichero /etc/inetd.conf debe existir:

```
nntp stream tcp nowait news /usr/sbin/tcpd /usr/local/sbin/leafnode
```

Estas comprobaciones no son necesarias porque el sistema de instalación las trae automatizadas, pero servirán para descartar problemas si el servicio no funciona correctamente.

El siguiente paso lógico es editar el fichero de configuración con el objeto de adpatar el servicio a nustras necesidades e indicarle a Leafnode una información vital para su correcto funcionamiento.

El último paso sería automatizar los comandos para bajar los grupos y las noticias. Para esto se puede poner el `fetchnews` dentro del script ip-up para bajarlos cuando hagamos la conexión a Internet o bien añadiendo una línea dentro del cron. Aunque siempre nos queda el método manual, esto es, ejecutarlo desde la línea de comandos.

Reiniciamos el demonio inetd o xinetd:
`kill -HUP 'cat /var/run/inetd.pid'`
En el fichero /etc/hosts.deny pondremos:
`leafnode: ALL EXCEPT LOCAL`
Ejecutamos `fetchnews` después de haber hecho los cambios en el fichero de configuración, con lo que se bajarán la lista de grupos disponibles. Luego nos conectaremos desde nuestro cliente favorito, teniendo en cuenta que el servidor será local y no se necesitará usuario ni contraseña. Ejecutamos `fetchnews` por segunda vez y comenzarán a bajar las noticias de los grupos suscritos hasta un máximo que ha sido definido en el ficher de configuración.

17.1.3 Configuración.

La configuración de Leafnode es relativamente sencilla, se concentra en un sólo archivo. Para RedHat lo encontraremos en `/etc/leafnode/config`, que es donde se instala por defecto. Para Debian en `/etc/news/leafnode/config`. Este archivo tiene varias opciones que vamos a explicar a continuación:

- server: el servidor al que se conectará Leafnode.
- supplement: un servidor de apoyo para bajar aquellas noticias que no encontremos en el principal o cuando este no esté disponible.
- expire: el tiempo máximo que permanecerán las noticias en el servidor Leafnode.
- username: nombre de usuario para acceder al servidor.
- password: la clave para permitir el acceso.
- port: puerto al que se conectará Leafnode para bajar las noticias si es diferente del 119.
- timeout: máximo tiempo de espera si no se conecta.
- nodesc: aquí decidimos si queremos bajar los grupos con descripciones, valor 1, o sin descripciones, por defecto, valor 0.
- filterfile: fichero en el que se encuentran los filtros.
- groupexpire: tiempo de caducidad de las noticias dentro de un grupo determinado.
- maxfetch: número máximo de noticias que se bajarán desde el servidor en una sesión.

- initialfetch: el número de noticias que se obtienen la primera vez que Leafnode baja un grupo de noticias. Útil cuando se trata de grupos que tengan mucho tráfico, de esta manera se bajarán los más actuales.
- delayboy: para bajar sólo los encabezados en un primer momento y, posteriormente, los cuerpos de aquellas noticias que hayamos seleccionado con el lector de noticias. Hay que tener cuidado porque da problemas con el lector de noticias del Netscape.
- maxcrosspost: número máximo al que se ha enviado una noticia, si excede el número, la noticia no será almacenada.
- maxage: con esta opción indicamos la antigüedad máxima de las noticias a bajar.
- maxlines: número máximo de líneas que puede contener la noticia.
- maxbytes: tamaño máximo de las noticias, para evitar la descarga de binarios.
- timeout_short: indica el número de días que Leafnode mantendrá un grupo de noticias antes de darlo de baja cuando no tiene a nadie suscrito al mismo.
- timeout_long: es el número de días que Leafnode seguirá bajando noticias a un grupo que nadie está leyendo, luego lo dará de baja.
- timeout_active: con esta opción indicamos a Leafnode un intervalo en días para que actualize la lista de grupos desde los servidores.
- create_all_links: para aquellos que usen lectores de noticias que permitan puntuar/matar, valor 1 para activar, 0 para desactivar.
- hostname: nombre del servidor de noticias que saldrá en la noticia en el caso de que Leafnode no lo resuelva de forma satisfactoria.

Leafnode consta de una serie de programas que tienen como objetivo el control del servidor:

- fetchnews: con este comando se prepara a Leafnode para albergar los grupos de noticias y para bajar las noticias cada vez que queramos. Presenta una serie de modificadores que nos ayudan a controlar su funcionamiento. Los modificadores más destacados son v, f, x, l, n y P. Con el modificador `v` hacemos que pase a modo de depuración, de esta manera irá mostrando en pantalla los mensajes que va generando, con lo que podemos ver el estado actual o los mensajes de error si los hubiera. La profundidad de depuración viene marcada por la cantidad de `v` que se pongan. El modificador `f` se utiliza para volver a leer las listas de grupos desde los servidores, puede tardar tiempo y se utiliza cuando Leanode comienza a dar problemas al servir las noticias. El modificador `x #` nos permite bajar `#` noticias extras desde el exterior. El modificador `l` se usa para que Leafnode no utilice los servidores suplementarios. Con el modificador `n` se puede comunicar a Leafnode que no elimine los grupos que no se leen, esto es, cuando nos inscribimos en un grupo que nos interesa pero que no leemos desde hace tiempo. El modificador `P` se utiliza para mandar las noticias que hayamos recibido localmente a los servidores externos pero no bajar ninguna noticia nueva.

- texpire: este es el programa que retira del servidor todas aquellas noticias que superen el tiempo dado en la directiva expire o groupexpire. Sus modificadores son dos, `v` y f. El modificador `v` es el mismo que en el comando anterior. El modificador `f` se utiliza para forzar la caducidad de las noticias independientemente del tiempo de acceso.

- applyfilter: con este programa se busca dentro de las noticias almacenadas en el directorio `news` todas aquellas que coincidan con los patrones dados dentro del fichero filters para ser retiradas. Su sintaxis es `applyfilter grupo de noticias`, ha de notarse que se debe dar el nombre completo del grupo, por ejemplo, para el grupo misc de la jerarquía ecol ha de ponerse `applyfilter es.comp.os.linux.misc`.

- checkgroups: se utiliza para actualizar las descripciones de los grupos.

- newsq: con este vemos las noticias que hay en la cola listas para ser enviadas al servidor en la próxima conexión.

17.1.4 Los filtros.

Aquí es dónde apunta la directiva `filterfile` del archivo de configuración. En este archivo se colocan todas aquellas reglas destinadas al filtrado de mensajes que no queramos recibir. Dentro de Usenet hay una práctica que se va extendiendo y que es perjudicial, el denominado Spam, con el filtrado de las direcciones desde las que se remiten los mensajes no solicitados se consigue que los mensajes que leamos se ajusten a la materia del grupo de noticias al que estemos apuntados. Otra manera de controlar estos mensajes es con la directiva `maxcrosspost` del archivo de configuración que evita leer mensajes que vayan a más de una determinada cantidad de grupos a la vez, el denominado crosspoting, y que es muy

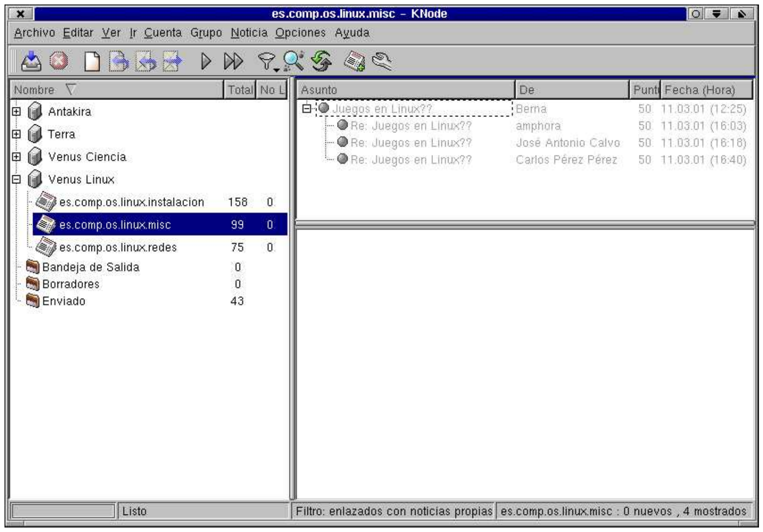


Figura 17.2: Viendo las noticias en knode

utilizado por los spammers.

Las expresiones para el fichero de filtros siguen la misma estructura que las expresiones regulares de Perl.
Con la siguiente línea filtramos cualquier noticia que tenga como remitente de correo `todosexo@sex.com`:

```
^(i?:from):.*[< ]todosexo@sexo.com(>|$| )
```

Ejemplo de filtro para el apartado Asunto (Subject en inglés):

```
^Subject.*\[Cursos-linux\]*.
```

Con esta regla filtramos todas las noticias que tengan en el asunto [Cursos-linux].
De esta forma podremos evitar bajar noticias que contengan unas determinadas palabras o unos remitentes molestos. En `http://www.escomposlinux.org/spam/` se pueden encontrar consejos para filtrar el spam y filtros constantemente actualizados que filtran los spammers dentro de la jerarquía es.comp.os.linux.*.

17.2 INN

17.2.1 Obtención e instalación

Capítulo 18

Configuración de servidor FTP

Capítulo 19

Agentes de Transporte de Correos

19.1 Introducción

19.2 Configuración básica de MTAs

19.3 Configuración de herramientas anti-spam